SYSC 4907

# Come and Chat (C&C): Fourth Year Engineering Final Report Draft 1

Carleton University

## Group Members:

Yunzhou Liu 101027110

Shizhong Shang 101115304

Zirui Qiao 101100225

## Supervisor:

Lynn Marshall

30 November 2022

# Abstract

Instant messaging (IM) technology is a type of online chat allowing real-time text transmission over the Internet or another computer network. There are many famous IM software, such as Discord, WhatsApp and Facebook Messenger. To develop the web based instant messaging software --- Come and Chat (C&C), a series of research, analysis, design, implementation and test will be done.

This report tracks the processes of developing the C&C system from the research phase to the very beginning of the implementation phase because later stages still need to be started. After a survey of three of the most famous instant messaging software, Discord, WhatsApp and Facebook Messenger, many unexpected techniques were learned. Followed by intense discussions and rigorous analysis, we extracted many common and valuable requirements from the three famous IM software and added some new features of our own conception. A detailed requirements document was then created. Use Case Diagram and Description are generated based on the requirement documents, then the Flow Diagram, Overall Structure, ER Diagram, Back-end UML, Front pages prototype, Front-end UML and finally, the API specification document.

Then, a series techniques that will be used in implementation are identified through discussion and justification about choosing them is provided in the report. Some estimation of challenges and risks that may encounter in the future are listed, as well as the timetable.

# Table of Content

# 1. Introduction

## 1.1 Purpose

Instant messaging is an efficient way to deliver information, and today, instant messaging software has become an integral part of our lives. For some of the most famous instant messaging software, such as Discord, WhatsApp and Facebook Messenger, we enjoy the convenience they offer while occasionally complaining about their shortcomings. This has led us to take a keen interest in such software.
Based on the instant messaging software that we have studied, in this project, we will build an imitated, web-based instant messaging software called Come and Chat (C&C). This proposal will outline the proposed plan and objectives to complete a working prototype of the project in early January 2023.

## 1.2 Scope

The C&C will provide mechanisms for signing up/in users, searching users, adding/deleting friends, sending/receiving messages to/from friends, building group chats, recording chat history, and allowing users to post messages as their status, allowing users' friend to see users' status.
More functions could be added after the functions above are implemented. Customized parts are always considered after the procedures above.

## 1.3 Plan

## 1.4 Accomplishments

To design a reasonable C&C system, researches about Discord, WhatsApp and Facebook Messenger are done. A lot of unexpected techniques are learned from them.

A detailed requirements document is generated according to our researches and thinking. And the design part is finished in high quality by following the design part in subsection 1.3.

Through discussion and further investigation, main techniques that will be used in implementation is determined. Learning about these techs is ongoing.

Finally, some possible challenges in implementation were identified. Draft solutions to these challenges are also proposed.

## 1.5 Overview of Report

In this Progress Report, the research about Discord, WhatsApp and Facebook Messenger will be shown, followed by the requirements documents. The complete design section of the C&C will then be explained in depth. The justifications for choosing specific techniques for C&C implementation will also be shown. Timetable and possible risks are shown, and corresponding solutions are provided. Finally, a comparison between the requirements document of this report and the previous proposal is made, and the conclusion is stated.

# 2. Engineering Project

## 2.1 Health and Safety

According to the Health and Safety Manual, Yunzhou Liu, Zirui Qiao, Shizhong Shang are laboratory workers and Lynn Marshall is a supervisor. In this project, there is no need to use the school's laboratory and anything other than computers. Laboratory staff will report all events in the project to the supervisor, such as the specific implementation process of the project and the required resources. The work place for completing the project is located at the home of the laboratory staff and the school library. In the library, the laboratory staff only need to use their own laptops to work. It will not involve the use of any chemicals when working.

## 2.2 Engineering Professionalism

Team 6 consists of 3 people. The team is not managed by a single person but negotiation among the three people. Before authority and cyber security problems are solved, the software will not be published to the public, thus neither of Professional Misconduct nor Code of Ethics will be violated. All of three people attended the weekly meeting on time. All of the team members are polite to each other in any communication and share opinions actively.

## 2.3 Management Methodology

Water Fall model is selected for our project management. We spent tons of time on designing, it is unlikely for us to miss many important requirements. Therefore, the requirements for our project is tend to be explicit and constant. In our project, only one iteration is estimated to be proceed.

Rapid Prototyping model is an excellent model for software projects, it allows flexible design and development.

Developers can make any changes in no either designing phase or developing phase, and the end user can “see” the system requirements as they are being gathered by the project team. For our project, there is no end user in our designing nor developing phase. The whole team designed the project together and discussed various aspects of the project, changes will occur, but not much. Another reason for not using the rapid prototyping model is that this model requires rapid iterations, so our team must put tons of time into the project. All three of us planned to graduate in 2023, and this model is not friendly for 4th-year students like us.

Spiral Model is very attractive for our team. It handles change through iteration and minimizes risk. Still, rapid iteration is not a good choice for our team. Doing risk analysis takes time and require a large team, 3 people team is never considered as a large team.

Rapid Application Development model is also a good model, it divides the whole project into several small projects and use waterfall model on each. The start of the second small project does not require a complete of the first small project, such developing model could work very fast. However, performing such a development process requires a very familiar team with the corresponding area, which has to be large enough to split into smaller groups for different small projects. Our team of three did not meet any of the above points, so we could not choose this model either.

The main idea of using the incremental model is to keep the whole team working; however, for a small group of only three team members dealing with such a complex C&C project, most of the work is already planned, and there is not much free time. Therefore, there is no need to choose incremental model despite its excellence.

In conclusion, the Water Fall model best suits our situation.

## 2.4 Relation to Degree Program

The title of the courses are described in Appendix 1.

### 2.4.1 Yunzhou Liu

My major is computer system engineering. I learned the basic programming languages Java, python and c. The back-end part I am responsible for in the project is written in the java language. I also learned about database structure, which can be applied to the creation of project databases.

I learned SYSC 3020, which introduces the structure of software engineering, and helps me build the overall structure of the project. The various drawing techniques I used in the project, such as UML diagrams, and the hierarchical knowledge required for a complete project come from this course. In the two courses SYSC 3303 and SYSC 3310, I learned about real-time systems, which can help me solve the concurrent problem of multiple people sending messages at the same time in the project. SYSC 4801 taught about security issues in information transmission. I will use this knowledge to ensure the safety of information transmission in the project.

ECOR 2050 and ECOR 4995 have taught me the processes and specifications to pay attention to when making projects.

### 2.4.2 Shizhong Shang

I am majoring in Software Engineering, which mainly studies program designing, programming and communication skills. In the final project, we will use HTML [15], CSS [16] and JS [17] to build the front-end of this web app, and the back-end is java based, the database is built with MYSQL and Redis.

I have learned Java code from SYSC 2004, SYSC 3110 and SYSC 4806, these three courses teach java code. In SYSC 3110, we are divided into groups to do a game project, which will help with scheduling and team working. In SYSC 4806, I chose to make an online server as the project, it will help us with the use of database, HTML, CSS and JS.

SYSC 3120 and SYSC 4106 will help us with the beginning of the designing of this web app, such as the structure, use cases and diagrams. And by digging into the project, we will get a better understanding of how to organize this project and risk management. CCDP 2100 has trained us on how to communicate with each other, especially when the team members are from different backgrounds.

### 2.4.3 Zirui Qiao

My major degree is Software Engineering which taught me about programming skills, data structure and, most important - project management. In this project, we will build the front-end part with Vue3 [18], the back-end part with Java [19] and the database part with MYSQL [20] and Redis [21].

I learned about Java and used it as a tool to build projects in SYSC2004, SYSC3110 and SYSC3303. These courses helped me build a firm base for Java which would be very helpful in this project. SYSC2100 taught me about data structure and algorithms, and COMP3005 taught me about the operation of Databases; these two courses will reduce the hindrance in database development. The design and management of a project are always

essential, SYSC3120 and SYSC4106 are great courses, and I will practice the knowledge in this project. This semester, SYSC4101 is a course on software validation, and I believe I will use what I have learned in this class in this project.

Communication skills can be easily overlooked, but with the help of CCDP2100 and ECOR4995, I am aware of the importance of communication skills and will apply them in this project.

## 2.5 Individual Contribution

### 2.5.1 Report Contributions

| Name | Progress Report Contribution |
|---|---|
| Yunzhou Liu | 2.1 Engineering Project - Health and Safety<br>2.4.1 Engineering Project - Relation to Degree Program - Yunzhou Liu<br>3.1 Background - Discord<br>3.2 Background - Database<br>5.1 Design - Use Case Diagram<br>5.2 Design - Use Case Descriptions<br>5.3 Design - Flow Diagrams<br>5.5 Design - Back-end<br>6.1 Tools Selected and Justification - Java8<br>6.2 Tools Selected and Justification - Spring<br>6.3 Tools Selected and Justification - MYSQL<br>7.1.1 Implementation - Backend - MYSQL<br>7.1.2 Implementation - Backend - Springboot<br>8.2 Validation and Verification - API Testing |
| Shizhong Shang | 2.4.2 Engineering Project - Relation to Degree Program - Shizhong Shang<br>3.1 Background - Discord<br>5.6 Design - Front-end<br>5.7 Design - API Specifications<br>5.4 Design - Overall Sequence Structure |

| | |
|---|---|
| | 6.5 Tools Selected and Justification - Vue3<br>6.6 Tools Selected and Justification - Element-Plus & Tailwind<br>7.2.3 Implementation - Frontend - Multi language<br>7.2.4 Implementation - Frontend - Components and Views<br>7.2.5 Implementation - Frontend - Stores<br>12 Conclusions |
| Zirui Qiao | Abstract<br>1. Introduction<br>2.2 Engineering Project - Engineering Professionalism<br>2.3 Engineering Project - Management Methodology<br>2.4.1 Engineering Project - Relation to Degree Program - Zirui Qiao<br>3.3 Background - WhatsApp<br>4. Objectives<br>5.4 Design - Overall Project Structure<br>6.4 Tools Selected and Justification - Redis<br>6.7 Tools Selected and Justification - HTTP and WebSocket<br>6.8 Tools Selected and Justification - Restful API<br>7.1.3 Implementation - Backend - Redis<br>7.2.1 Implementation - Frontend - HTTP request and api<br>7.2.2 Implementation - Frontend - Router<br>7.3 Implementation - WebSocket<br>8. Validation and Verification - Unit Testing<br>10. Documentation<br>11. Discussion and Analysis<br>Appendix 2. Source Documents |
| Together | 2.5 Engineering Project - Individual Contribution<br>Reference<br>Appendix 1. Definition, Acronyms, and Abbreviations<br>Appendix 3. Proposal |

Table 1.0 Individual Contribution in Progress Report

### 2.5.2 Project Design Contribution

| Name | Project Design Contribution |
|---|---|
| Yunzhou Liu | ERDiagram V1.0-V1.2<br>Admin Back-end UML<br>Module UML V1.1 |
| Shizhong Shang | User Page Structure<br>Admin FlowDiagram<br>Admin Front-end Page<br>Admin Front-end UML |
| Zirui Qiao | UseCase Diagram V1.0-V1.3<br>User FlowDiagram V1.0-V1.2<br>User Back-end UML V1.0<br>API specification V1.0-V1.4<br>Overall Project Structure |
| Yunzhou&Zirui | Module UML V1.0 |
| Shizhong&Zirui | User Page Design<br>User Front-end UML |
| All Members Together | UseCase Description 1st, 2nd draft |

Table 1.1 Individual Contribution in Project Design

### 2.5.3 Project Implement Contribution

| Name | Project Implement Contribution |
|---|---|
| Yunzhou Liu | Back-end service package<br>Back-end dao package<br>Back-end pojo package<br>Back-end vo package |

| | |
|---|---|
| | API tests<br>Database |
| Shizhong Shang | Front-end views folder 8 files<br>Front-end local folder<br>Front-end-ad local folder<br>Front-end-ad api folder<br>Front-end-ad assets folder<br>Front-end-ad component folder<br>Front-end-ad request folder<br>Front-end-ad router folder<br>Front-end-ad views folder<br>Front-end-ad store folder |
| Zirui Qiao | Back-end common package<br>Back-end config package<br>Back-end controller package<br>Back-end config package<br>Back-end handler package<br>Back-end util package<br>Back-end unit tests<br>Front-end api folder<br>Front-end assets folder<br>Front-end component folder<br>Front-end request folder<br>Front-end router folder<br>Front-end store folder<br>Front-end util folder<br>Front-end views folder 11 files<br>Front-end other setting |

# 3. Background

## 3.1 Discord

To study the functions and implementation methods that we will use in our project, we first looked at Discord [1].

Discord is a chat software. Discord started from game voice and IM tool services, then turned to live broadcast platforms, and then opened a community platform for game stores, becoming the preferred tool for players to communicate and collaborate in games.

According to the Discord development log, Discord's internal resources include Application, Audit Logs, Auto Modification, Channels, Emoji, Guild, Guild Scheduled, Guild Template, Invite, Stage Instance, Sticker, Users, Voice, and Webhook. Each instance corresponds to the user's operation. For example, a channel. You can create a new channel. Channel settings, such as name and type, are determined by channel attributes. The user enters the information on the front-end page. After the information is transferred to the back-end, the back-end creates a channel instance to contain the input information; and then operates the database through the instance.

Our project also uses chat technology. Some internal resources of Discord are helpful to our project. The user object in Discord contains a variety of attributes, including id, username, avatar, locale, email and other attributes that we can apply to our projects. The services for corresponding users include acquiring users, acquiring current users, modifying user information, and acquiring current users' Guilds. Guild object has id, name, owner_ ID, roles, etc. The corresponding service methods include creating Guilds, obtaining Guilds, modifying Guild’s information, deleting Guilds, etc. Object is associated with a database structure [1].

This information describes the functions and specific implementation methods of Discord, and helps us define the basic function we need to have when designing projects.

## 3.2 Database

A database is a warehouse that organizes, stores and manages data according to the data structure. Each database has different APIs for creating, accessing, managing, searching, and copying saved data.

The database is implemented by MySQL technology. MySQL is developed by MySQL AB of Sweden and belongs to Oracle. MySQL is a relational database management system. Relational databases store data in different tables instead of putting all data in a large warehouse, which increases speed and flexibility [2].

## 3.3 WhatsApp

We have investigated WhatsApp in detail.

WhatsApp is a famous IM software focusing on simple, secure, reliable messaging. Its features include text messaging, voice and video calls, photo and video sharing, document sharing, voice sharing, group chat, end-to-end encryption, feasible on the web, desktop and mobile [3] and many more. These features of WhatsApp have inspired our team, and we will use some of them as a C&C requirement.

Another document about WhatsApp has also been investigated: WhatsApp Business Platform. From WhatsApp Business Platform's official website: "*The WhatsApp Business Platform gives medium to large businesses the ability to connect with customers at scale. You can start conversations with customers in minutes, send customer care notifications or purchase updates, offer your customers a level of personalized service and provide support in the channel that your customers prefer to be reached on.*" [4]. It is certainly possible to implement C&C with the help of WhatsApp Business API, but that would be too boring, and we would learn very little from it.

Instead of using the API directly, our team decided to learn from the APIs and design our APIs. One exciting thing found during the investigation is called Authentication; the WhatsApp service will authenticate the sender's identity each time a request is received. The request is responded to only when the identity has the power. Only a few operations need no authentication, such as login and register. The authentication operation is done by verifying the token in the HTTP request header, and the token should follow a specific format. The token will be generated after login and expire in 7 days, or after logout [5].

WhatsApp sends all requests in RESTful API form and only in 5 common methods: post, get, delete, patch and put. "*A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding.*" [6]. This technology is very helpful for the C&C project in HTTP request implementation.
WhatsApp Business document published their API response format which is made up of 4 components:

a) HTTP Status Code: Status codes are issued by the server in response to a request made to the server. Represent the status of the response (e.g., error, success …)
b) Payload: Actual data, only returns when the request is successful.
c) Meta: Some version data (e.g., API client version)
d) Errors: Errors' detail, only returns when errors occurred.

This is very helpful for our team to design the response data form. Except for Meta, C&C will need all the other three components in responses [7].

Capacity Rate Limits are also provided for each request, the sender cannot send requests higher than a specific rate. Our team believes this is a means of combating DDoS attacks and will not be used in this project as C&C will not be very large.

In general, WhatsApp is excellent as an IM software. Not only is it efficient and secure, but it is also prepared to deal with many unpredictable events. Our team's investigation also closely examined its client architecture [8], security [9], API reference [10] and current research [11]. Still, they are all less relevant to this project for various reasons, so no more details will be provided here.

## 3.4 Facebook Messenger

Messenger is an instant messaging service owned by Facebook. But at the same time Messenger can be used without a Facebook account and it can be used on Android and iOS devices (mobile application), works as a Web-app, and it also has a desktop version for Windows and iOS. So Messenger is a successful app that not only keeps users from Facebook, but also gathers more users whose not using Facebook and makes them know the Facebook app from using Messenger, it is a perfect example of expanding the user amount of a single app [12].

As a messaging app, it can do the basic function which allows users to send messages, short audio, short video, and pictures and it also can be used to make phone calls and video calls. After those basic functions, this app can also connect with other apps, such as sending locations from Map, sending pictures or videos from the local album, texting friends from both Facebook and Instagram, and adding bank cards and PayPal to transfer money between friends [13].

Messenger also has more functions, making this app more usable and fancier. Users can create chat groups and send messages, make group phone calls and video calls as well, then the group members can play games and do the poll in the group chat, when the user thinks the chat or the group is too annoying, they can mute them for the time length they chose. On the chat page, users can change the theme of the app, to make the app unique and more personal, when the user gets a message, they can simply replay it with emojis. Messages also support the users to share their life and make them closer to their friends, there is a function called watch together, it allows chat friends to watch shows, games, films and TVs together over video chat, so people can talk about the same

video anywhere anytime. To make this app more secure, whenever the user gets back to Messenger for other apps on their phone, their device will ask for the Face ID or the fingerprint to let the user get into the app [14]. These features increase the playability of Messenger.

Messenger is one of the most typical instant message-sending apps for the public, almost all features in the app are free. This app makes life became more convenient. It allows users to connect anywhere anytime as long as there is data from the service company or WiFi. At some point, it helps people save money by not calling others through the phone and gives the users a better life. Messenger provides our team with many new ideas for developing a new usable web app, to make our app better, some deeper research and more rigorous analysis will be needed.

# 4. Objectives

All requirements listed in each subsection of section 2 are in priority order, which means the requirement with the highest priority order in 2.1.1 is shown as the first requirement in section 2.1.1. The subsections in section 3 are also in priority order; the subsections int front are the prerequisites of the following subsections. For example, a user cannot search for friends (2.1.2 - 1) before register in (2.1.1 - 1) the C&C system.

## 4.1 Functional Requirements

### 4.1.1 Identity Functions

1. Users provide email, nickname and password to sign up for the system.
2. Users provide their nickname and password to sign in.
3. Users can modify their profiles, including changing nickname, password, email, address, avatar, phone number, and birthday.
4. Users can log out of C&C.

### 4.1.2 Friend Functions

1. Users search for other users by nickname.
2. A user can apply to be a friend of another user.
3. A user can approve/reject a friend request from another user.
4. A user can view all his/her friends in a friend list.
5. A user can delete a friend.
6. A user can mute a friend so that he/her will not receive any message from the friend until unmuted or

enters the chat room of the friend.

7. A user can hide a friend so that he/her will not be able to find the friend from friends list.
8. A user can unset a friend so that the friend is unmuted and unhidden.
9. A user can show all friends so that all friends, including hidden friends, are shown in friends list.
10. A user can search his/her friends by friend's nickname.
11. A user can view a friend's information including: avatar, nickname, email and statuses.

### 4.1.3 Group Functions

1. A user can create a group by providing a proper group name; group notice, avatar and group initial members are optional.
2. A user in a group can increase the member of a group by inviting friends.
3. A user can view all groups he/her joined in a group list.
4. A user can leave a group.
5. A user can view all the members of the group he/her has joined.
6. Users can view and change the information of the groups they have joined.
7. A user can mute a group so that he/her will not receive any message from the group until unmuted or enters the chat room of the group.
8. A user can hide a group so that he/her will not be able to find the group from group list.
9. A user can unset a group so that the group is unmuted and unhidden.

### 4.1.4 Chat Functions

1. Chat messages can only be sent to/received from a user's friends and groups.
2. Users can send texts and images in chat room as a chat message.
3. Only the user themselves and the friend/group members can view messages in a chat room.
4. All users in a group chat can view messages from all users in the group chat.
5. A user can view the chatting histories of chats and group chats.
6. A friend receiving a chat message and a user sending a chat message should happen almost simultaneously.
7. A user will be able to know whether or not unread messages have been sent by a group or friend without entering a chat room.

### 4.1.5 Status Functions

1. A user can post text and image messages as his/her status.
2. A user can only view his/her own and friends' statuses.

3. A user can ‘like’/’unlike’ a status and view the amount of ‘like’s of a status.
4. A user can write text as comments to a status.
5. A user can view all comments of a status.
6. A user can delete his/her own statuses.

### 4.1.6 Administrator Functions

1. An administrator of C&C can log into C&C by providing the correct username and password.
2. An administrator of C&C can view statistics including:
   a) total number of users,
   b) number of online users,
   c) total number of statuses,
   d) total number of comments, and
   e) total number of chat messages.
3. An administrator can search a user by nickname or user id.
4. An administrator can block/unblock a user, and the blocked user cannot log in to the C&C system until unblocked.

### 4.1.7 Optional Functions: Faction Grouping

1. There exist n factions in C&C set by the administrator
2. Every client must join 1 and only 1 faction during registration.
3. Clients within the same faction, if they are friends, can chat with each other without limits.
4. Chat messages sent from different factions will be encoded, which means one cannot chat with clients in other factions.
5. There are periodic events that allow 2 random factions to understand each other’s messages over time. These events are triggered by time.
6. Clients cannot switch faction.

## 4.2 Page Requirements

1. The list shows all friends' chats, and group chats should exist the whole time after signing in.
2. Signing in/up page should be a separate page from the chatting pages.
3. All messages sent by a user should appear with their avatar simultaneously.

## 4.3 Design Constraints

1. Performance should be reasonably fast.
2. The system shall be usable at least on Chrome.
3. The system shall support PC and mobile.
4. The system must comply with the relevant privacy legislation.

## 4.4 Functional Restrictions

1. Users can only see messages in chats between themselves and friends, and the group chat they joined.
2. Users can only view the history of messages that they can see.
3. The system can be remotely accessible to all users.
4. The system shall protect all users' information.

## 4.5 Other Requirements

1. The time plan can be divided into 2 parts, version 1 and version 2. Version 1 will be completed before Jan 22, 2023, and version 2 will be completed before Mar 14, 2023.
2. The hard deadline for the system is the end of April of next year for project completion.
3. One or many servers are required for this project. Specific requirements are still in process of investigation.

## 4.6 Measurability

When all 4.1 (Functional requirements), 4.2 (Page requirements) and 4.5 (Other requirements) are met, the project is primed for completion, which could be seen as a "B" project. When 4.3 (Design Constraints) and 4.4 (Software Quality Attributes) have been completed, the project has been fully completed, which could be seen as an "A" project. If extra functions are added the project is considered to be over-completed, and the project can then be rated as an “A+” project. For all requirements, unit tests of at least 80% code coverage ensures that the requirements are met correctly. An integration test would be necessary as well.

# 5. Design

The design methodology learned from SYSC 3120 (Software Requirements Engineering) was very useful in the design of this project. As a result, the design process followed the requirements analysis, modelling and specification learned in SYSC 3120.

# 5.1 Use Case Diagram

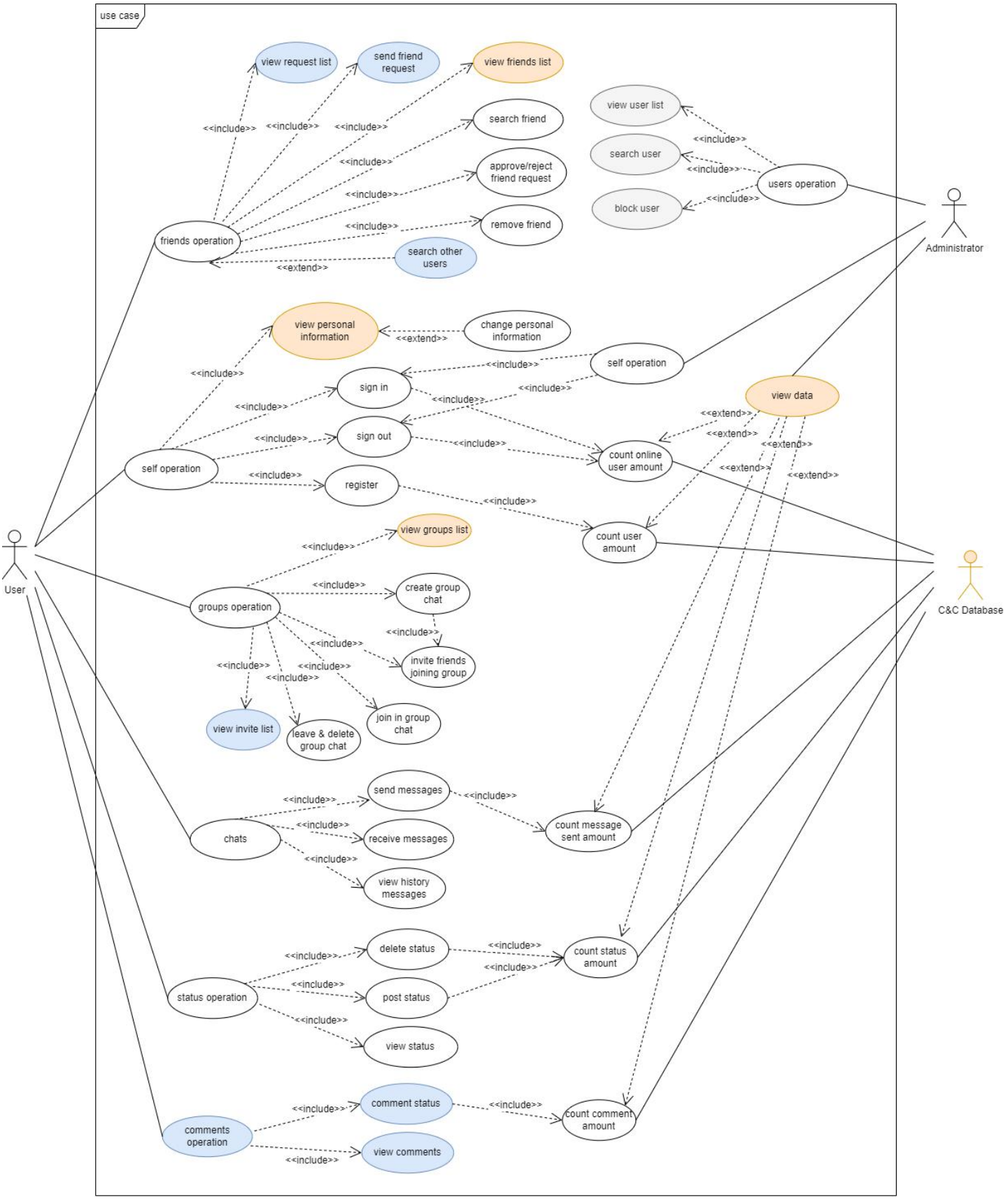



Figure 5.1.1: Use Case Diagram

Use Case Diagram Version 4

In the use case diagram, we made three modifications. Use cases with white as the background colour are designed when they are created. Use cases with an orange background are added or modified during the first modification. Use cases with blue background are added or modified during the second modification. Use cases with grey background are added or modified in the third modification.
There are three objects in the use case diagram: user, administrator and database.

Users mainly have six operations. The first case is that the user operates the friend system. Users can add, delete and find users at will. The second situation is that users operate the self information system. Users can freely change their personal data, register their accounts, log in or log out of their accounts. The third situation is that users operate the group system. Users can create groups at will, invite friends to join their own groups, leave groups, and accept group invitations from friends. The fourth situation is that users can manage their own chat system. Users can send, receive and delete chat messages with their friends or groups. The fifth situation is that users can manage their own status system. Users can create, delete, and view previous statuses. The sixth case is that users can add their comments after their friends or their status.

The administrator mainly performs three operations in this software. First, the administrator can log in or out of the account freely. Secondly, the administrator has the right to view the list of all users and lock other users. Finally, the administrator can view the database information, such as the number of registrations and online accounts.

## 5.2 Use Case Descriptions

Link on Appendix 2. Source Documents
Use Case Description Version 2

The use case description file records the information of each operation in the use case diagram in detail.
The use case description file records the information of each operation in the use case diagram in detail. Each operation is described by the following attributes.

Use case name

Brief description. A simple sentence that explains the meaning of the use case name.

Primary actor. The main roles involved in the use case

Secondary actor. Secondary actor participating in the use case

Precondition. You want to implement the precondition of the use case

Dependency. Derivative operation of use case

Basic flow. Basic process of realizing use cases

Specific alternative flow1. The response of the system when exceptional operations occur

Specific alternative flow2. The system's response when exceptional operations occur

Global alternative flows

Bounded alternative flows

Special requirements

## 5.3 Flow Diagrams

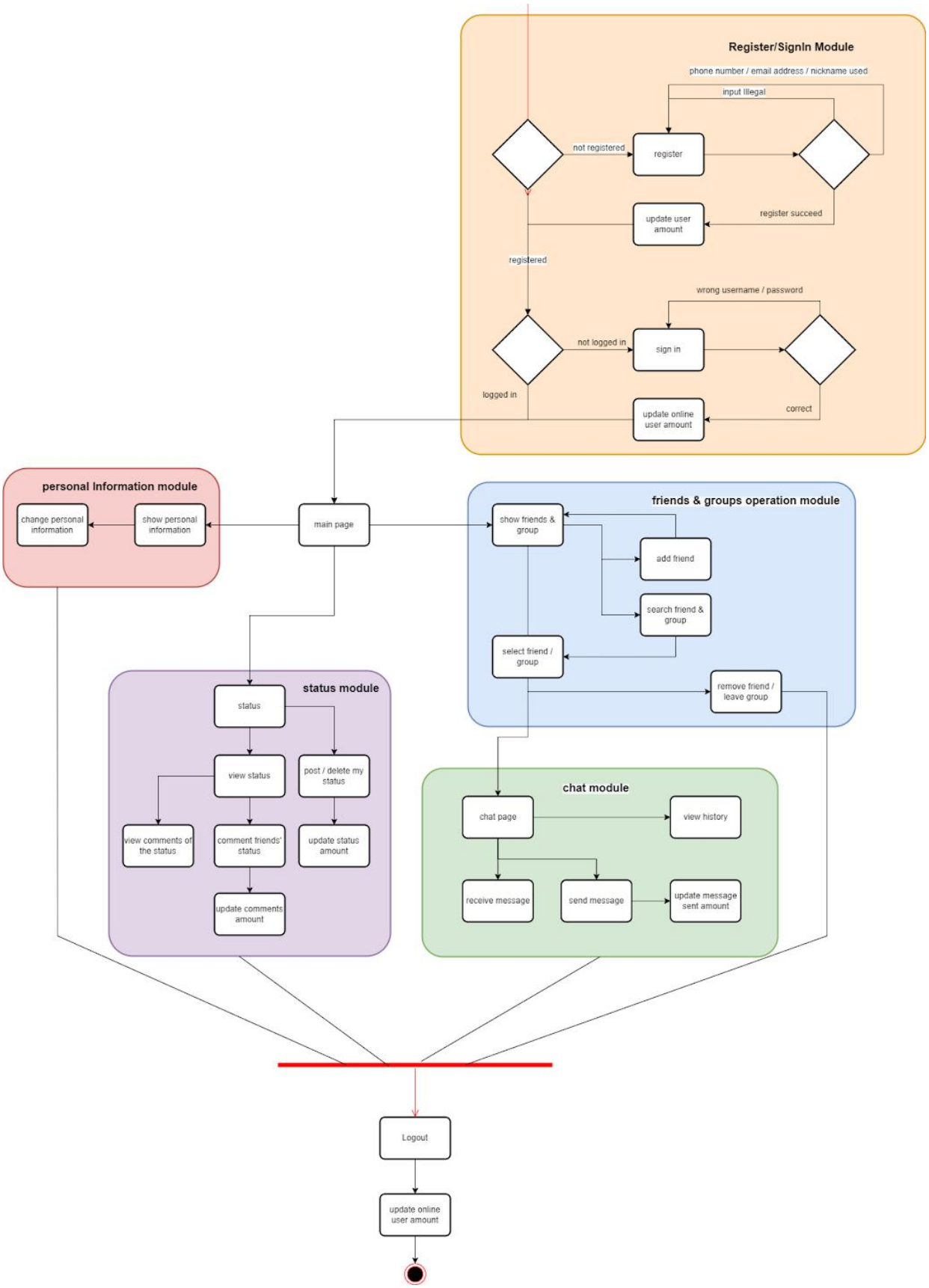



Figure 5.3.1: User Flow Diagram

Link on Appendix 2. Source Documents

User FlowDiagram Version 3

Flow Diagram shows the operation of a user from logging in to logging out of the account under normal process. First, the user enters the website page to log in. If there is an account, the user will enter the main page. If there is no account, the user will enter the registration page. The user enters the login page after successfully registering an account. Entering the correct user name and password will cause the user to enter the main page or enter the login page repeatedly after entering the wrong user name or password. After entering the homepage, users can choose to enter the personal information page, friends and groups page, chat page and status page. Users can enter the logout page to perform logout operation no matter which page they are on.

When the user enters the personal information page, the user can view and modify the user information.

When users enter the friends and groups page, they can add, delete, and view friends and groups.

When users enter the status page, they can add, delete, modify, and view the status.

When users enter the chat page, they can view past chat messages and send chat messages.

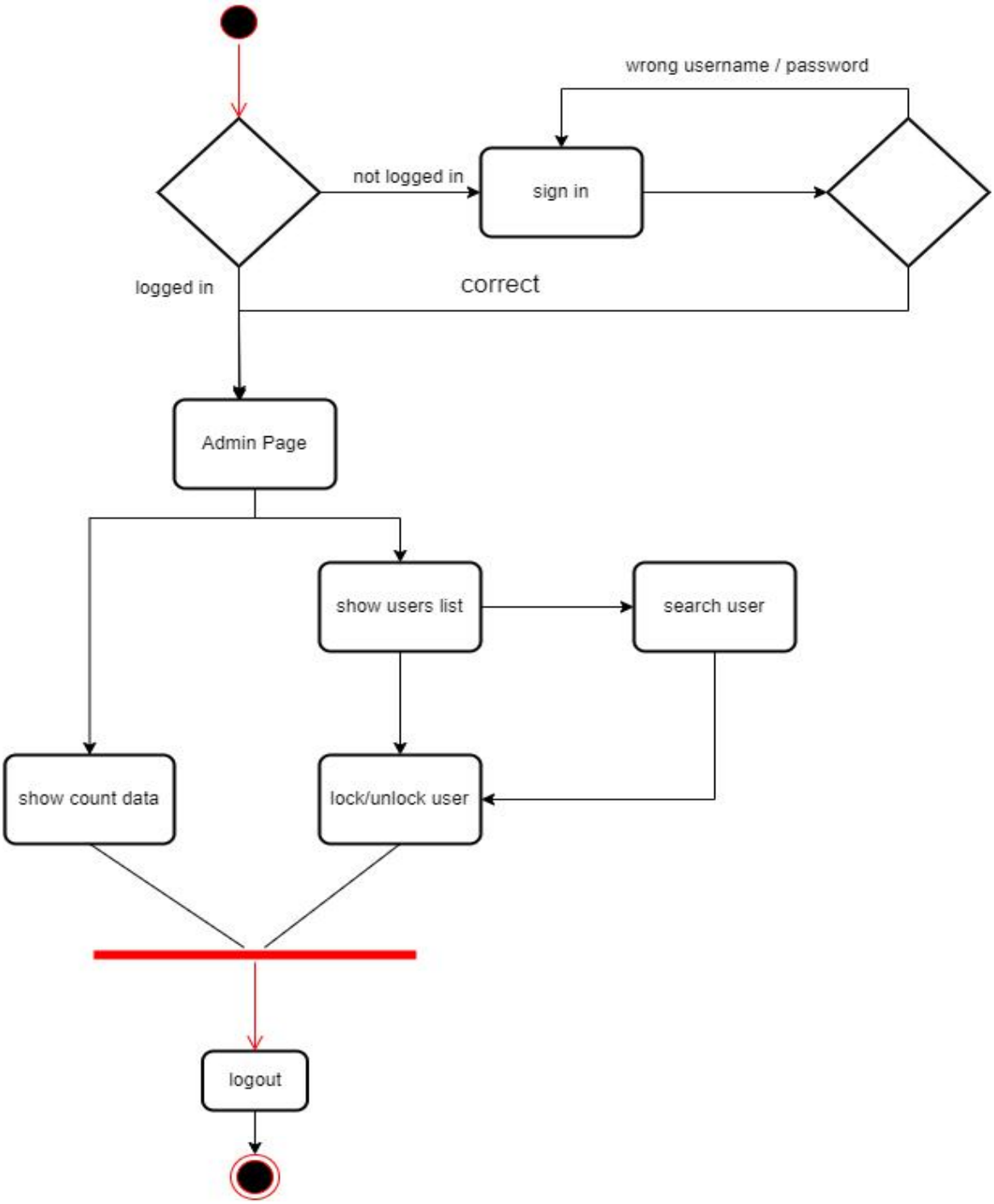



Figure 5.3.2: Administrator Flow Diagram

Admin FlowDiagram Version 1

The above figure shows the administrator flow chart. The process of the administrator when logging in the page is the same as that of ordinary users. The difference is that the administrator will enter the administrator page after passing the login page. On the administrator page, the administrator can view the user's data, such as the number of online users, the total number of public registrations, and then enter the login page. The administrator can also select to display all users and block users, and then enter the login page.

## 5.4 Overall Project Structure

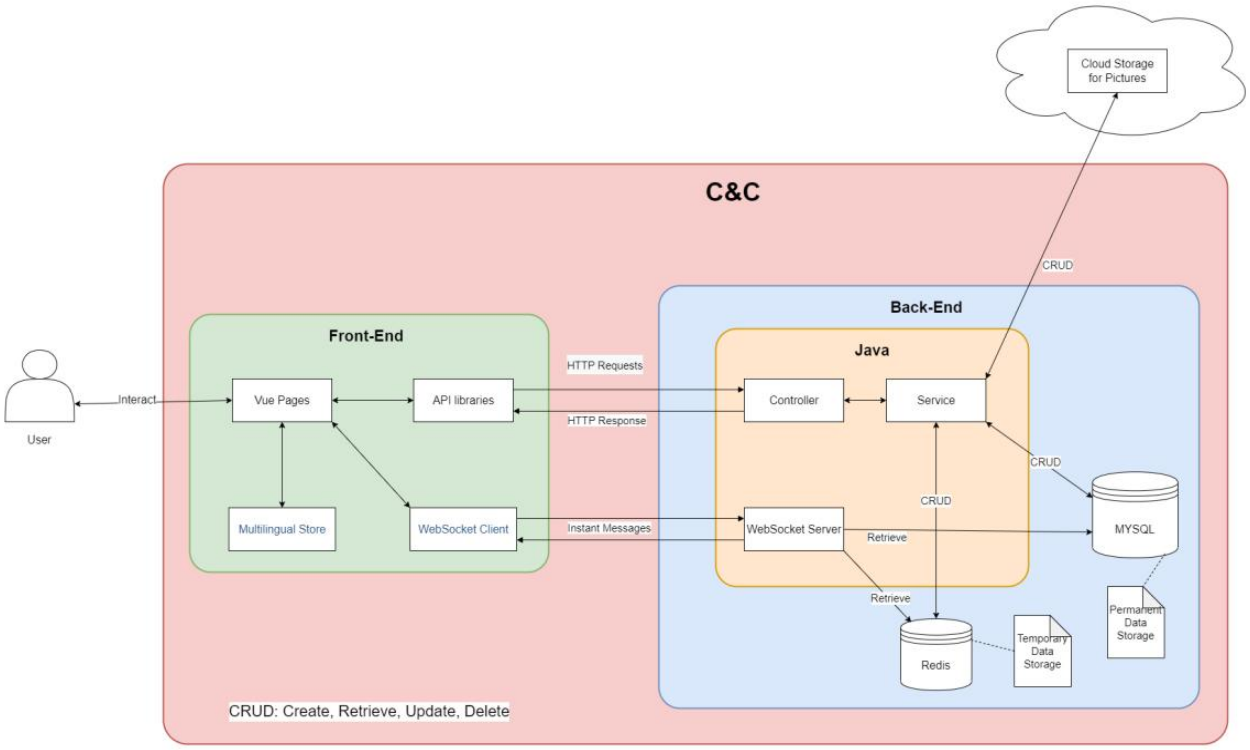



Figure 5.4.1: Overall Structure of C&C Project

An overall project structure is designed to describe the project in a more clear way.

Users can only interact with Vue pages in Front-End part. Vue pages handles page jumping by itself and provide multi-languages through the help of Multilingual Store. Most requests are sent to Back-End part through API libraries as HTTP requests. Once Back-End generate replies corresponding to requests, API libraries obtain and return the replies to Vue pages and display it to the user. Chat messages cannot be treated exactly as HTTP requests, therefore, WebSocket Client module will send and receive instant messages (WebSocket messages) to and from WebSocket Server in Back-End.

Java is the main software application in Back-End. The Controller module recognize all HTTP requests and invoke the corresponding service in Service module to handle the requests. Once service generate a result, Controller judge the result and reply the judging result to Front-End. MYSQL is a relational database which stores permanent data for C&C and Redis is a NoSql database which stores temporary data for C&C. The Cloud Storage on the top of the graph is a cloud database which can only store pictures, all pictures in C&C will be stored in the Cloud Storage module. Only Service module can perform all CRUD operations on the three

databases, CRUD stands for Create, Retrieve, Update and Delete. The WebSocket Server module is only used to accept and reply to instant messages (WebSocket messages). To obtain necessary user data, the WebSocket Server can retrieve data from both Redis and MYSQL.

C&C comprise all elements in the graph except for User and Cloud Storage. The C&C portion is labelled in red.

## 5.5 Back-end

### 5.5.1 ER Diagram

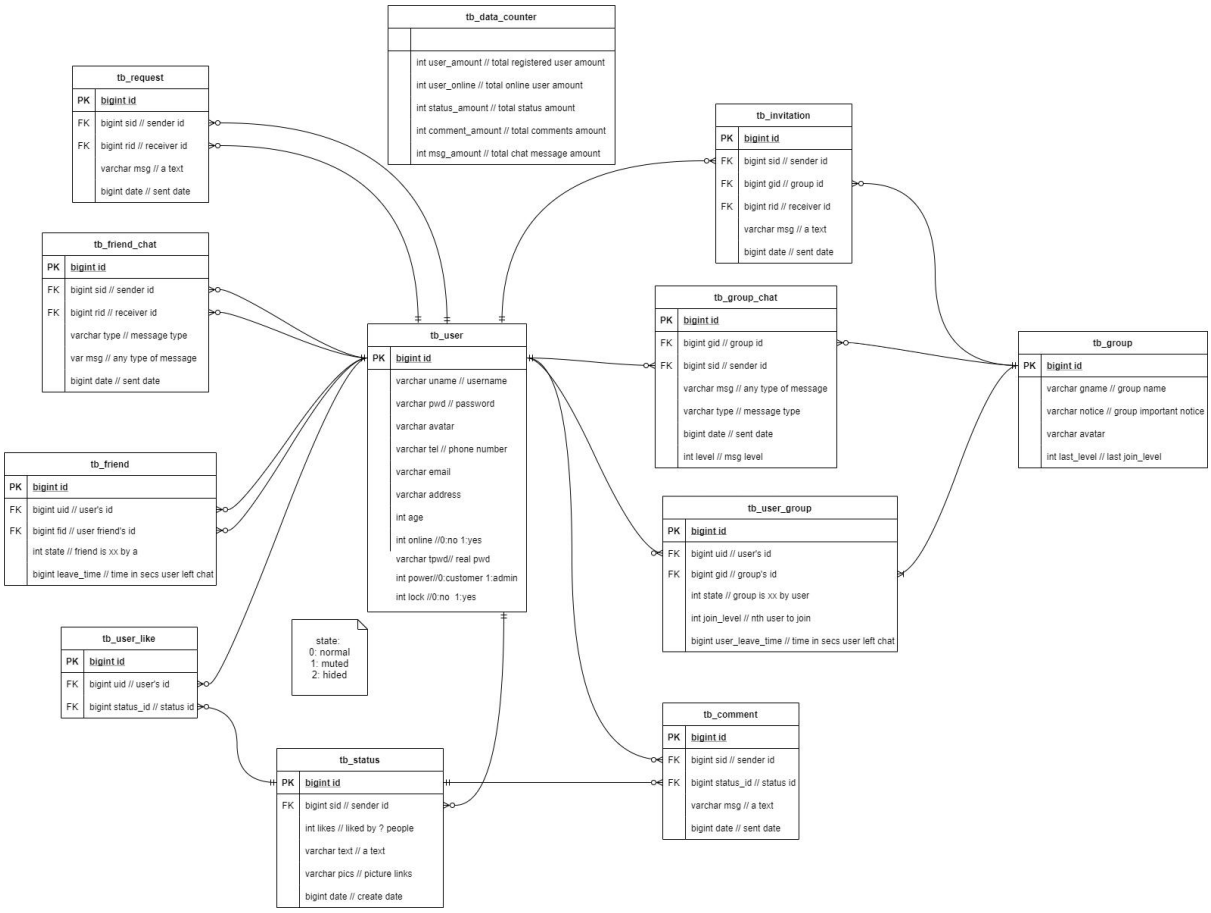



Figure 5.5.2: Database structure diagram

[ERDiagram Version 4]

Link on Appendix 2. Source Documents

The above figure describes the names and corresponding attributes of all tables in the database. The description of each attribute is labeled by annotations.

There are 12 tables in the database.The tb_user stores various user information. The online attribute

indicates whether the user is online. The pwd attribute represents the user's encrypted password. Tpwd represents the original password of the user. Lock indicates the status of the user's account and whether it is available. The tb_friend table stores friend relationships. The state attribute indicates whether the user is blocked by his friends in a friend relationship. The leave_time attribute indicates the time when the user last sent the message. tb_ friend_ Chat table stores chat information between users and friends. The type attribute indicates whether the user is sending a picture or text. The tb_ request table stores the invitation to add friends sent by users. The tb_ data_ counter table stores various information about the system. The tb_ invitation table stores the group joining invitations sent by users. The tb_group table stores all group information. The last_level attribute indicates the last member to join the group. The tb_ group_chat table stores the chat information of members in the group. The tb_ user_ group table stores group member information. The tb_ user_ like table stores users' likes of status. The tb_ status table stores all status information. The likes attribute indicates how many user tags like this status. tb_ Comment stores all comment information.

## 5.5.2 Back-end UML

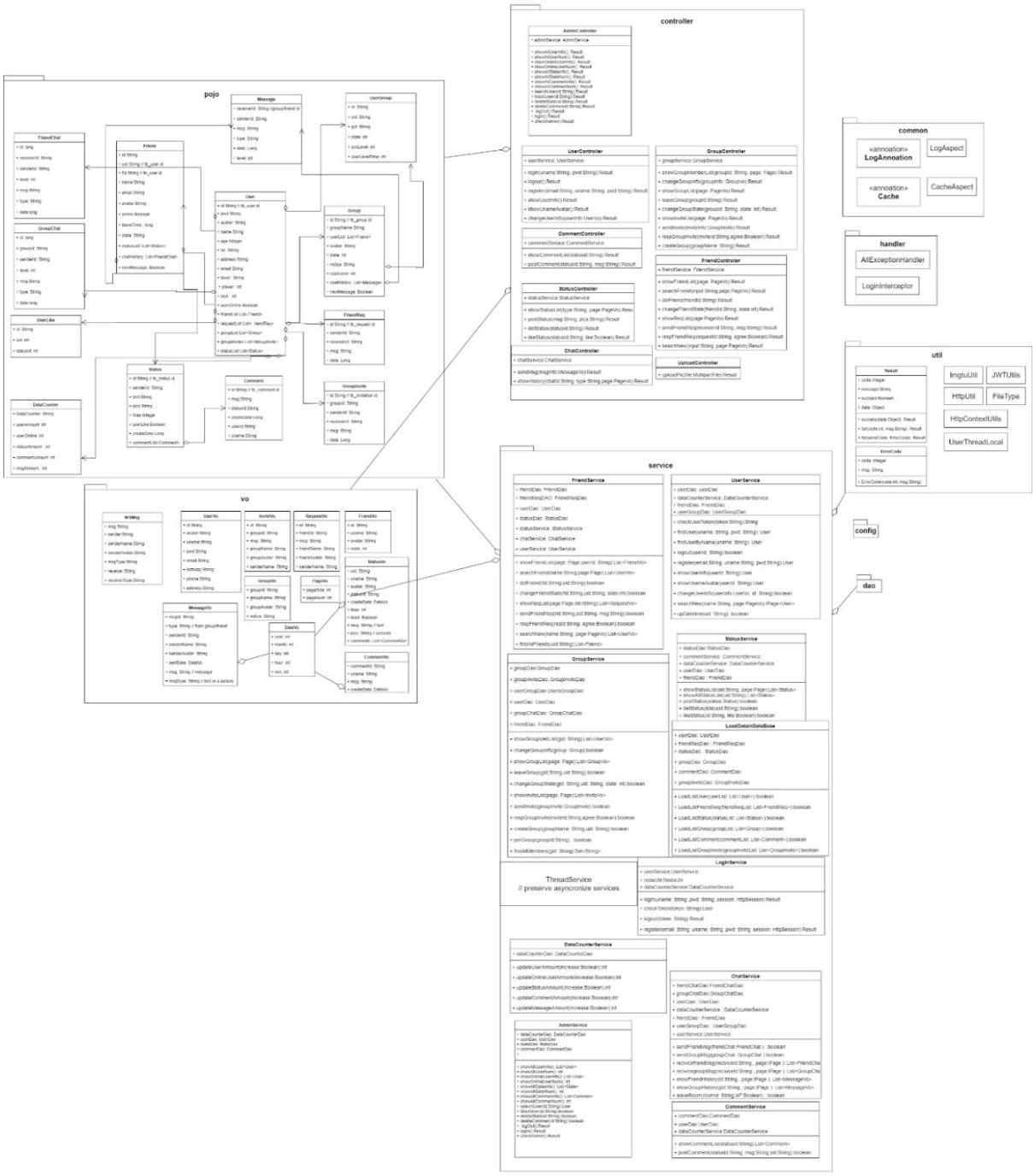
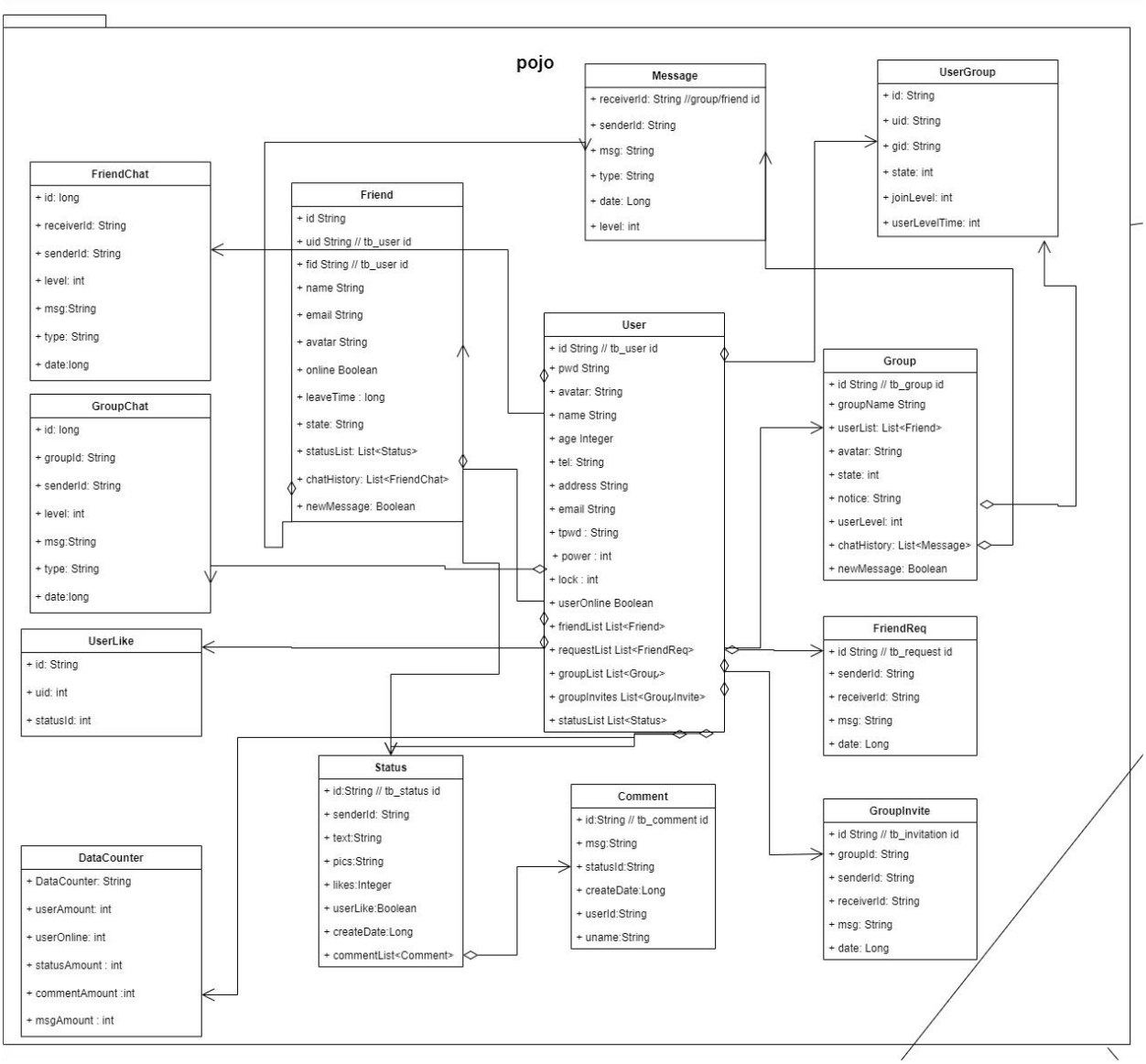



Figure 5.5.3: Detailed design drawing of back end

[User Back-End UML Version 2]

Link on Appendix 2. Source Documents

The figure above describes the specific design requirements of the control layer, service department, instance, tool layer and setting layer in the back-end.
The following series of pictures are the exploded view of Back-end UML. I will explain the details of each part in detail.

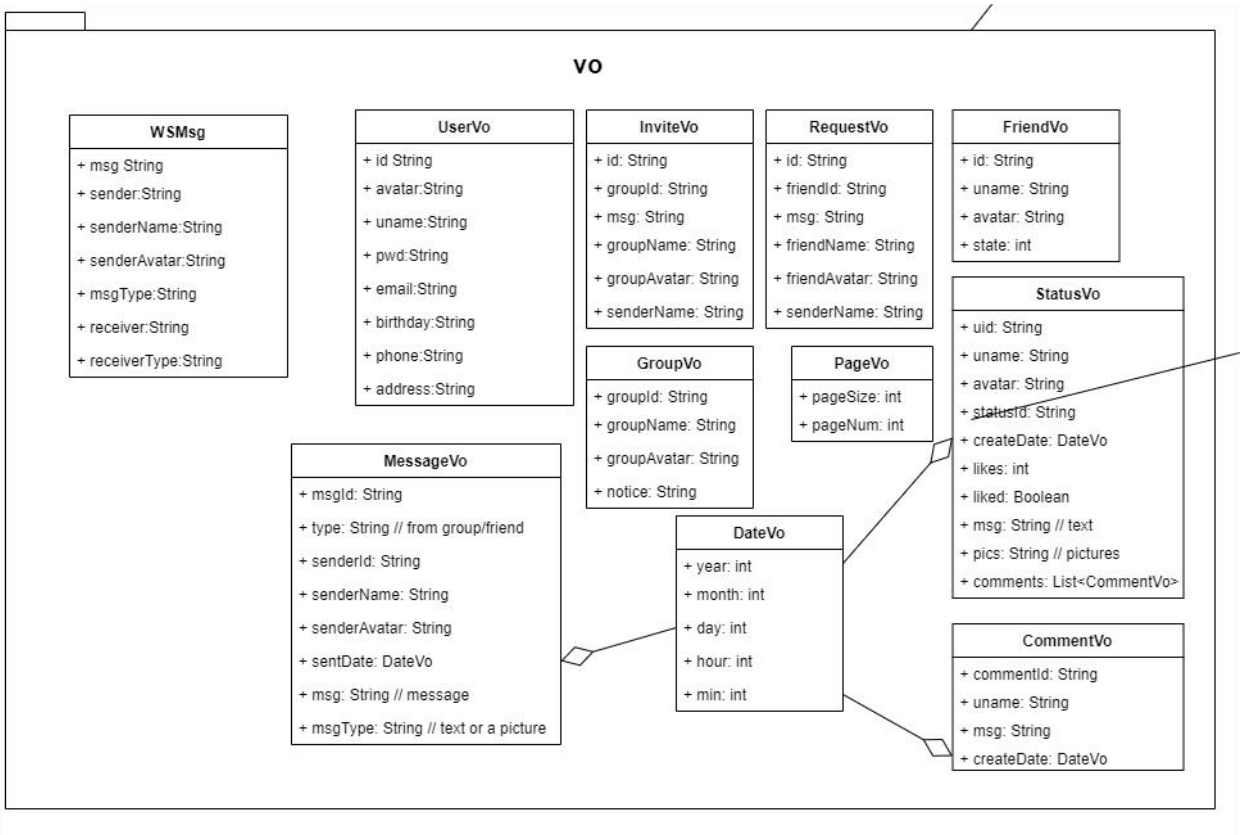



The classes in the pojo package are all objects used in the backend system. These object classes are associated with tables in the database. The dao layer can modify and find records in the database through the classes in pojo.

The user class corresponds to the tb_user  in the database.  The friendList, requestList, groupList, groupList, groupInvites and statusList attributes in the user class have no corresponding fields in the database. When calling the user class in the service layer, we need to call other classes to find the corresponding data and attach

the data to the user class. For example, a service needs to find the user's friend in the tb_friend according to the user name and store the data in the friendList.

The message class is different from other classes. This class has no corresponding table in the database. The message class is used to store the information of FriendChat or GroupChat in the message class and send it to the front end when the front end does not need detailed chat data.

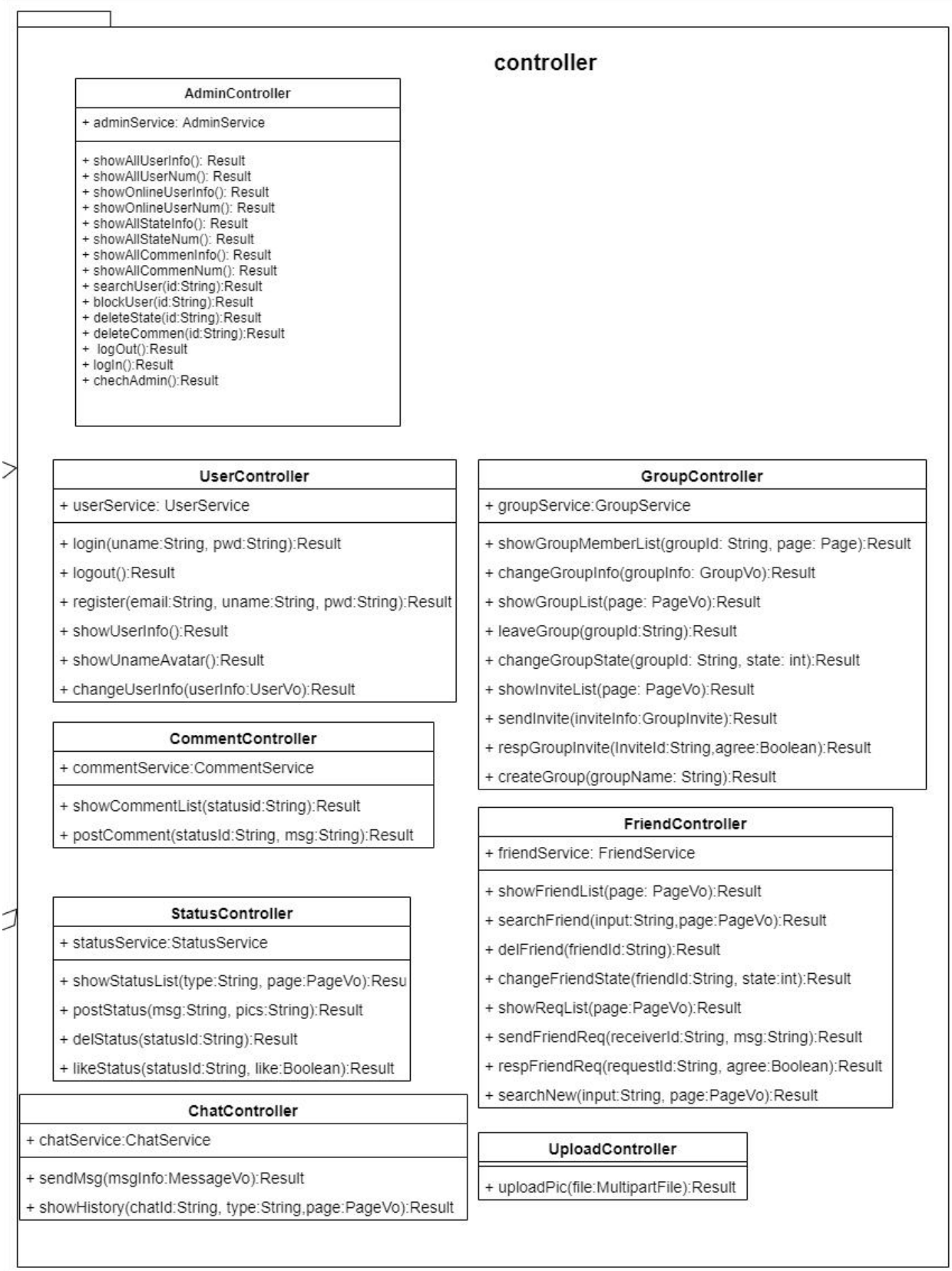



The classes in the vo package are especially used to send data to the front end. The service layer looks up the records in the database and uses the classes in pojo to store the records. Most of the data returned by operations in the service layer are in the form of classes in pojo. The data structure required by the front end is sometimes different from the class structure in pojo. At this time, the controller layer can convert the data returned by the service layer into the format of vo through the methods in the class of vo.

In addition, when the controller receives the data from the front end, part of the data is passed in in the format of the class in the vo package. For example, the front end needs to insert a new Status in the format of StatusVo. The controller layer accepts the StatusVo and extracts the information from it to form a status class, which is then inserted into the database.

PageVo is a special format that contains the size of information required by the front end. The front end passes PageVo into the controller layer. The controller extracts the pageSize and pageNum attributes of PageVo and adds them to the parameters of the service layer operation. PageVo is just a container for storing information exchanged between the front and back ends.

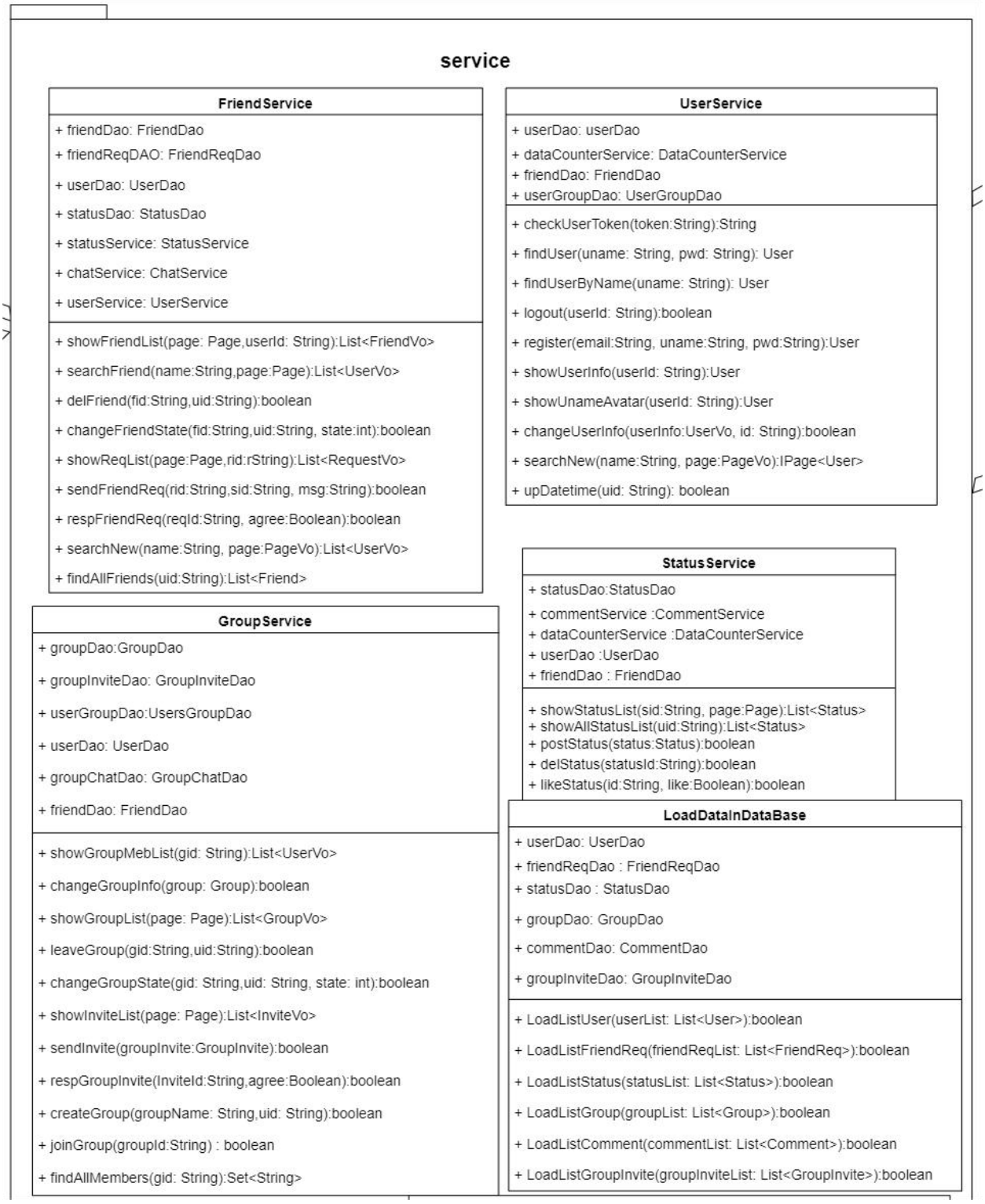
The purpose of DateVo is to provide time data. For example, the controller layer accepts the StatusVo from the front end and needs to insert it into the database. The controller layer needs to call the method in DateVo to get the current time and store the data in the status class.




The controller layer is used to receive information from the front end, and then call the method of the service layer to obtain information from the database, modify the format, and pass it to the front end.

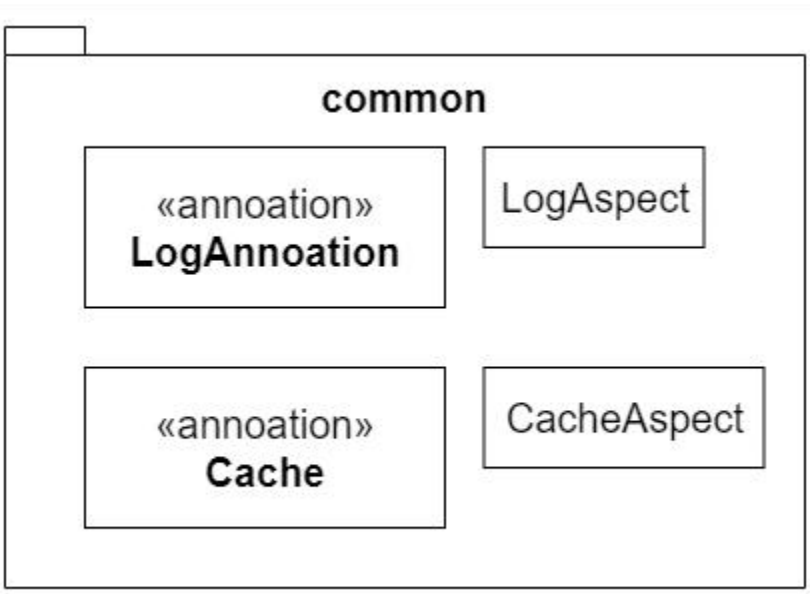
For example, commentService is instantiated first in CommentController. In the showCommentList method, the CommentController calls the method in the service layer to find the corresponding record in the database by accepting the status id information from the front end. The CommentController will package these data as the result class in the util package and send them to the front end.

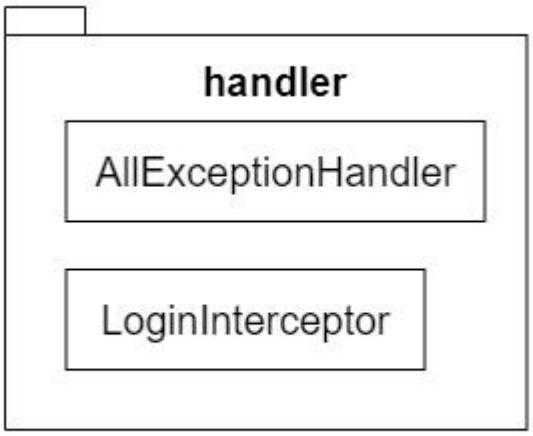
In the postComment method, the CommentController calls the method of the service layer to build a new comment and insert it into the database based on the status id and information passed in from the front end. Regardless of success or failure, the service layer will return a Boolean value. This method sets the result class based on this Boolean value and returns the front end.

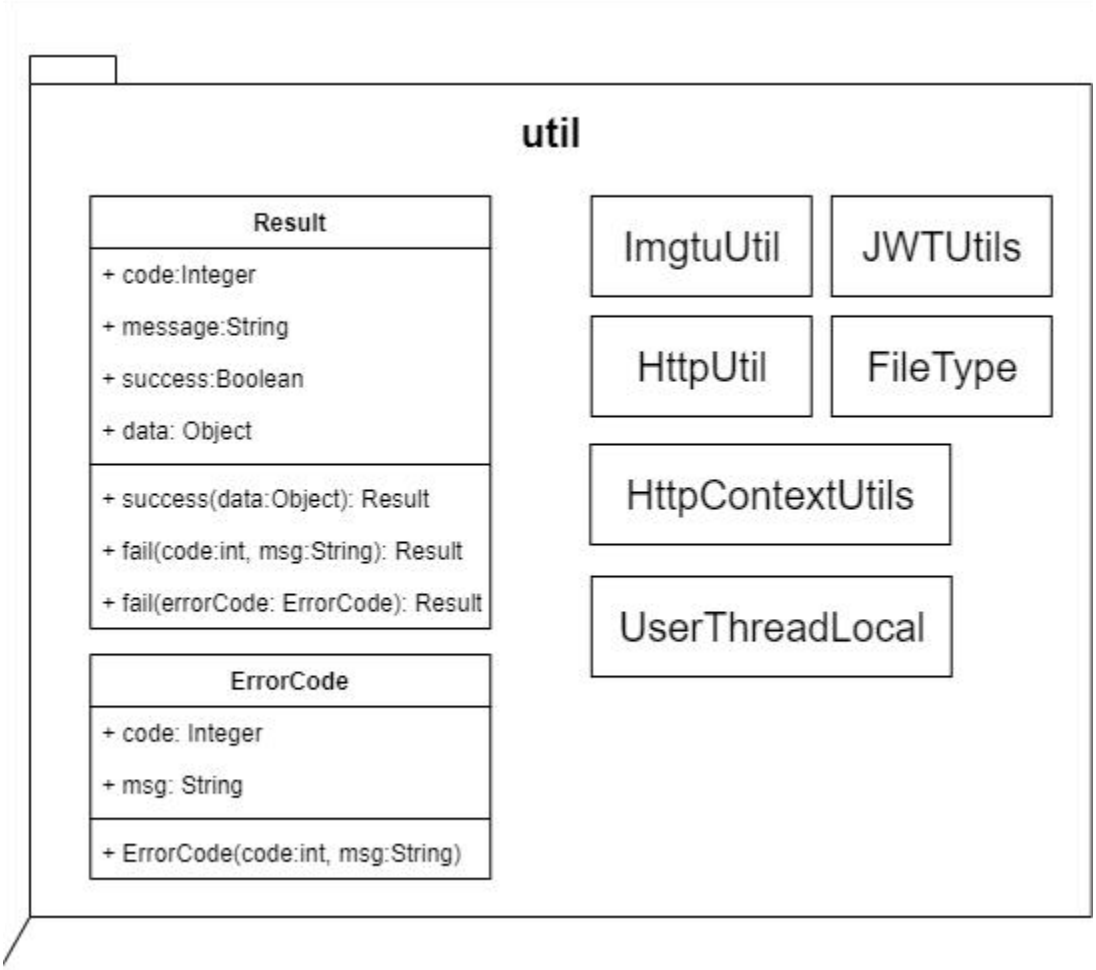

service
FriendService
+ friendDao: FriendDao
+ friendReqDAO: FriendReqDao
+ userDao: UserDao
+ statusDao: StatusDao
+ statusService: StatusService
+ chatService: ChatService
+ userService: UserService
+ showFriendList(page: Page,userId: String):List<FriendVo>
+ searchFriend(name:String,page:Page):List<UserVo>
+ delFriend(fid:String,uid:String):boolean
+ changeFriendState(fid:String,uid:String, state:int):boolean
+ showReqList(page:Page,rid:rString):List<RequestVo>
+ sendFriendReq(rid:String,sid:String, msg:String):boolean
+ respFriendReq(reqId:String, agree:Boolean):boolean
+ searchNew(name:String, page:PageVo):List<UserVo>
+ findAllFriends(uid:String):List<Friend>
UserService
+ userDao: userDao
+ dataCounterService: DataCounterService
+ friendDao: FriendDao
+ userGroupDao: UserGroupDao
+ checkUserToken(token:String):String
+ findUser(uname: String, pwd: String): User
+ findUserByName(uname: String): User
+ logout(userId: String):boolean
+ register(email:String, uname:String, pwd:String):User
+ showUserInfo(userId: String):User
+ showUnameAvatar(userId: String):User
+ changeUserInfo(userInfo:UserVo, id: String):boolean
+ searchNew(name:String, page:PageVo):IPage<User>
+ upDatetime(uid: String): boolean
StatusService
+ statusDao:StatusDao
+ commentService :CommentService
+ dataCounterService :DataCounterService
+ userDao :UserDao
+ friendDao : FriendDao
+ showStatusList(sid:String, page:Page):List<Status>
+ showAllStatusList(uid:String):List<Status>
+ postStatus(status:Status):boolean
+ delStatus(statusId:String):boolean
+ likeStatus(id:String, like:Boolean):boolean
GroupService
+ groupDao:GroupDao
+ groupInviteDao: GroupInviteDao
+ userGroupDao:UsersGroupDao
+ userDao: UserDao
+ groupChatDao: GroupChatDao
+ friendDao: FriendDao
+ showGroupMebList(gid: String):List<UserVo>
+ changeGroupInfo(group: Group):boolean
+ showGroupList(page: Page):List<GroupVo>
+ leaveGroup(gid:String,uid:String):boolean
+ changeGroupState(gid: String,uid: String, state: int):boolean
+ showInviteList(page: Page):List<InviteVo>
+ sendInvite(groupInvite:GroupInvite):boolean
+ respGroupInvite(InviteId:String,agree:Boolean):boolean
+ createGroup(groupName: String,uid: String):boolean
+ joinGroup(groupId:String) : boolean
+ findAllMembers(gid: String):Set<String>
LoadDataInDataBase
+ userDao: UserDao
+ friendReqDao : FriendReqDao
+ statusDao : StatusDao
+ groupDao: GroupDao
+ commentDao: CommentDao
+ groupInviteDao: GroupInviteDao
+ LoadListUser(userList: List<User>):boolean
+ LoadListFriendReq(friendReqList: List<FriendReq>):boolean
+ LoadListStatus(statusList: List<Status>):boolean
+ LoadListGroup(groupList: List<Group>):boolean
+ LoadListComment(commentList: List<Comment>):boolean
+ LoadListGroupInvite(groupInviteList: List<GroupInvite>):boolean


The methods in the service layer are specific operations related to the database. The classes in the service layer are divided into interface and implementation classes. The interface sets the envisaged method and implements the logic of the implementation method in the class. The service layer is usually called by the controller layer to accept the parameters passed in by the controller and return the data in the corresponding database.

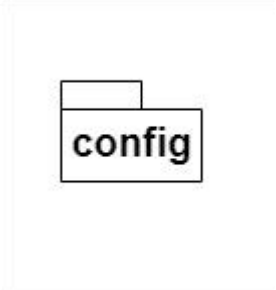
For example, CommentService has two methods in this service class, showCommentList and postComment. ShowCommentList takes the status Id as a parameter, finds the record with the same status Id as the parameter in the comment table in the database through commentDao, and returns all qualified records as a list. The postComment method accepts three parameters: status Id, comment information and sender. This method creates a new comment and inserts it into the database. This method will return a Boolean value based on whether the insertion is successful.

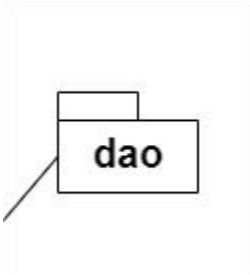

common
«annoation»
LogAnnoation
LogAspect
«annoation»
Cache
CacheAspect


handler
AllExceptionHandler
LoginInterceptor




The classes in the util package are tool classes to make other parts of the code more concise. The ErrorCode class contains the codes that need to be returned when all programs fail. There are different codes according to different failure types. The Result class is the data format returned from the front end in the controller. Result has two methods: success and failure. When the controller calls the method in the service and gets a successful return, the controller will call the success method of Result and put the information that needs to be passed into the front end as a parameter into the Result. When the controller fails to call the method in service, it will call the fail method of Result and use the code in ErrorCode according to the category of the method.






We use annotation development in dao layer. For example:

```
@Mapper
public interface FriendChatDao extends BaseMapper<FriendChat> {
}
```

Through annotations, we don't need to write complex mysql language. We can call records in the database through the preset interface in the library.

```
LambdaQueryWrapper<Comment> lqw = new LambdaQueryWrapper<Comment>();
lqw.eq(Comment::getStatusId, id1);
List<Comment> comments = commentDao.selectList(lqw);
```

The above example is a process for finding comments. The status id in the comment should be the same as the input parameter.

Id1 is the status id entered.

lqw.eq(Comment::getStatusId, id1); This code means that the filter condition is that the status id of comment is equal to id1.

## 5.6 Front-end

### 5.6.1 Prototype


Figure 5.6.1.1: Overall Client Side Prototype C&C Project

The figure 5.6.1.1 above is the user side overall prototype for the C&C project, and it can be check by the link User Front Page Structure in Appendix 2. Source Documents, this figure shows all twenty pages that will be shown to the user while using the C&C system. There is a main page that has a big difference of color with the other pages is because the team is still considering about the color set of this system when it is shown to the users.

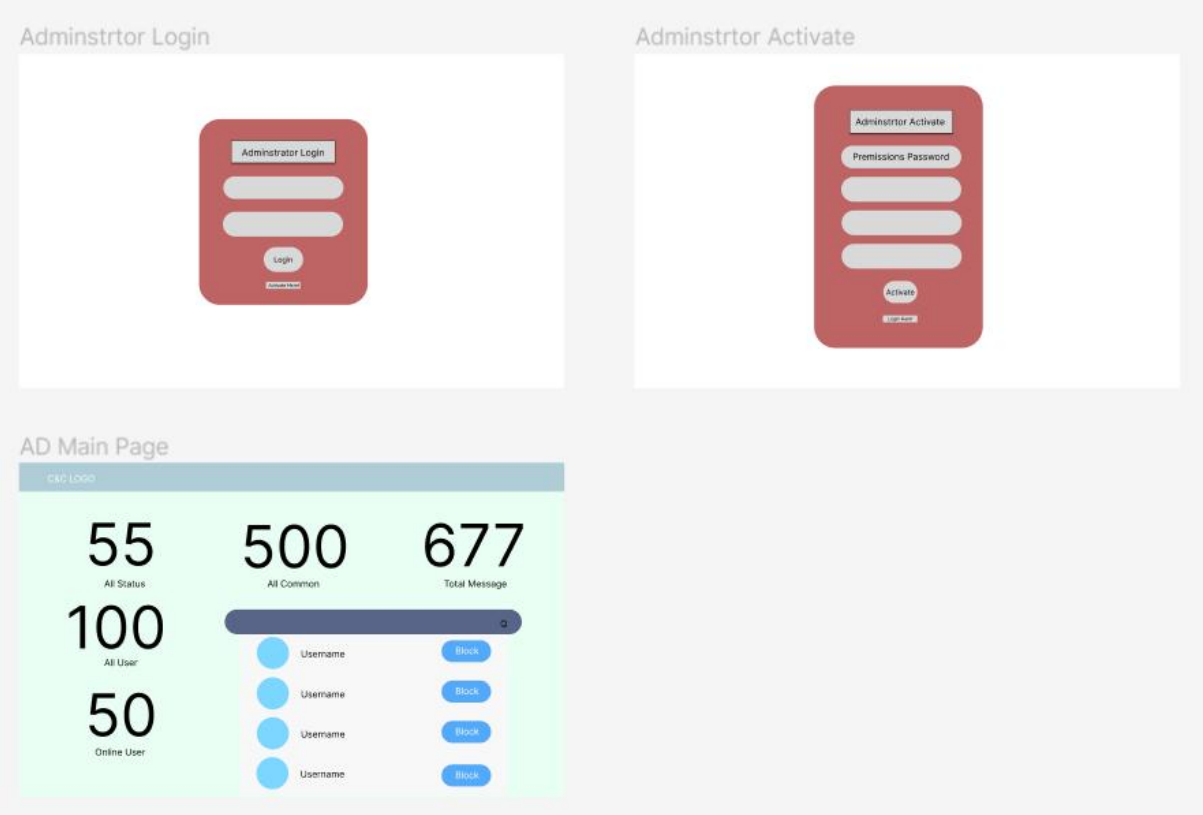



Figure 5.6.1.2: Overall Administrator Side Prototype C&C Project

The figure 5.6.1.2 above is the administrator side overall prototype for the C&C project, this figure shows all three pages that will be shown to the administrator while using the C&C system. The team is deciding about if new administrator is allowed to register or just fix it.

For more information on all the page structures, refer to 5.6.2 Page Structure Document

### 5.6.2 Page Structure Document

All the page designs are made in the website Figma [42], this design website is convenient to use and easy to make changes, so the team decide to use it as design support.

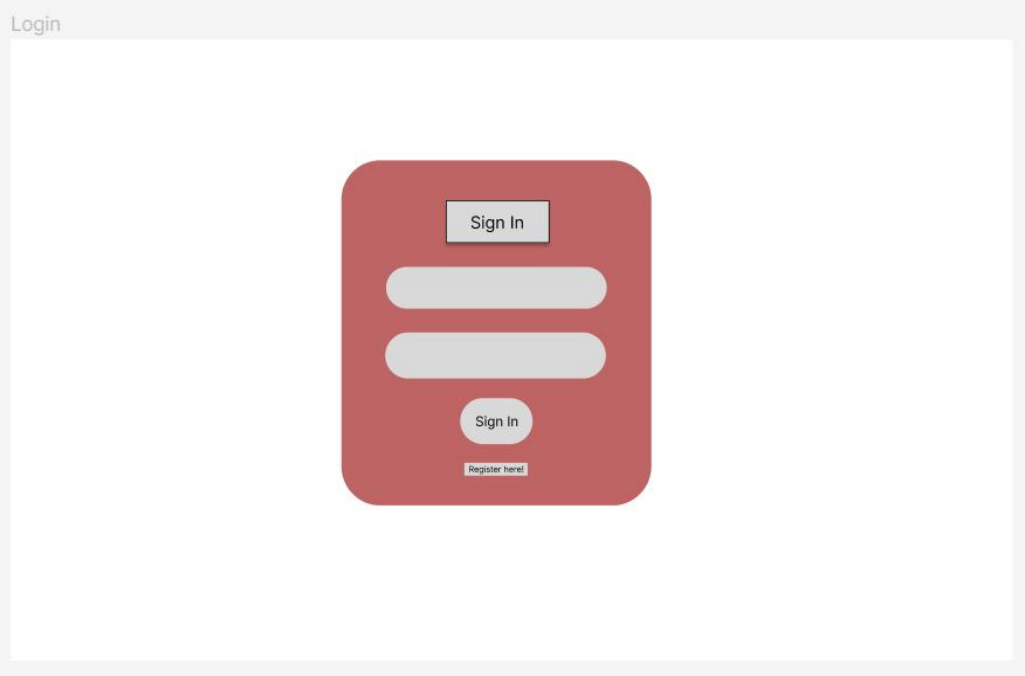



Figure 5.6.2.1 - “Login” page

“Login” page (Figure 5.6.2.1) is used to let users login to the system, user’s name and password are needed. On this page, the user can either choose to login by clicking the “Sign In “ button or if the user is new to the system, this page can jump to the “Register” page by clicking the “Register here!” button.

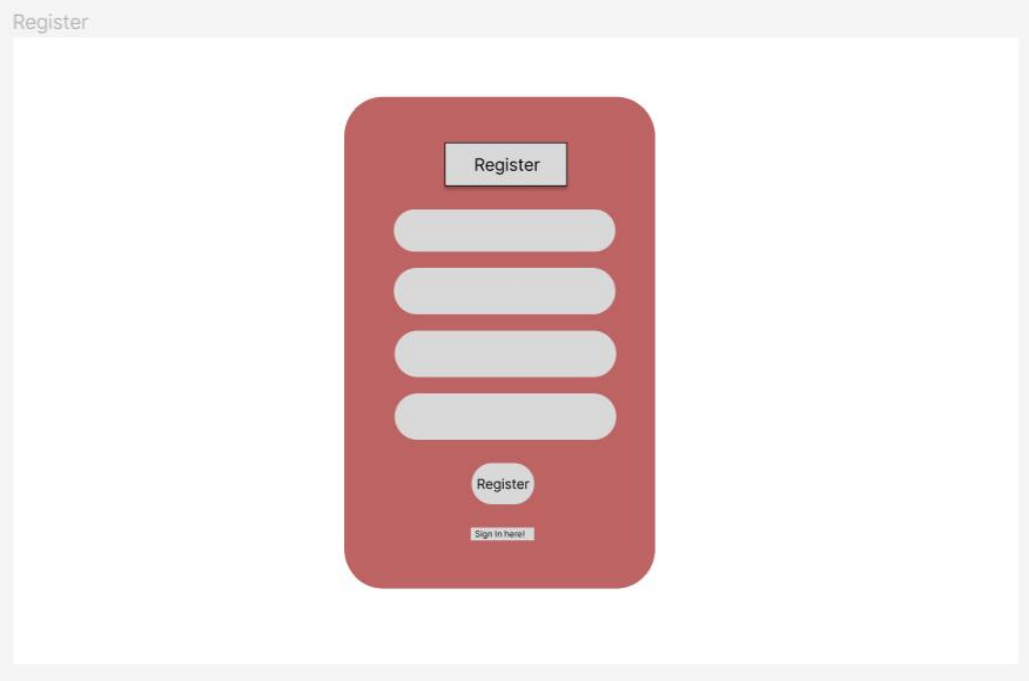


Figure 5.6.2.2 - “Register” page

“Register” page (Figure 5.6.2.2) is used for the registration of users. On this page, users need to create a username, provide their email address, and confirm their password to make sure the two passwords are the same. Users can click the “Register” button to register, or if the user already has an account, they can click “Sign In Here!” to jump to the “login” page.


Figure 5.6.2.3 - “Main” page

“Main” page (Figure 5.6.2.3) is the page that will be shown to users after they login to the system, it will show the C&C logo, the searching bar and the user avatar at the top, then it followed by six main parts of this website: friend list, group list, find something new, friends requests, group invitations and the status.

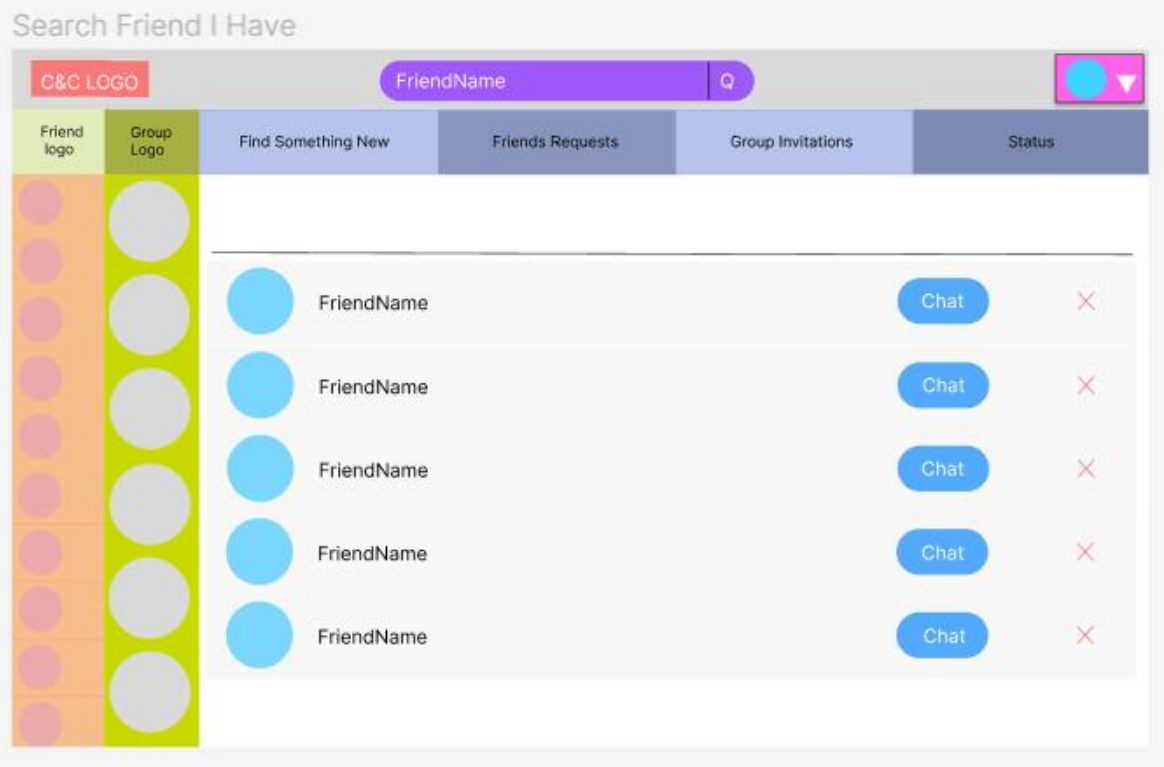



Figure 5.6.2.4 - “Search Friend I Have” page

“Search Friend I Have” page (Figure 5.6.2.4) is shown when the user searches a friend’s name in the searching bar, the page will show the scroll component of all the similar friends’ names, then the user can choose to chat with them.

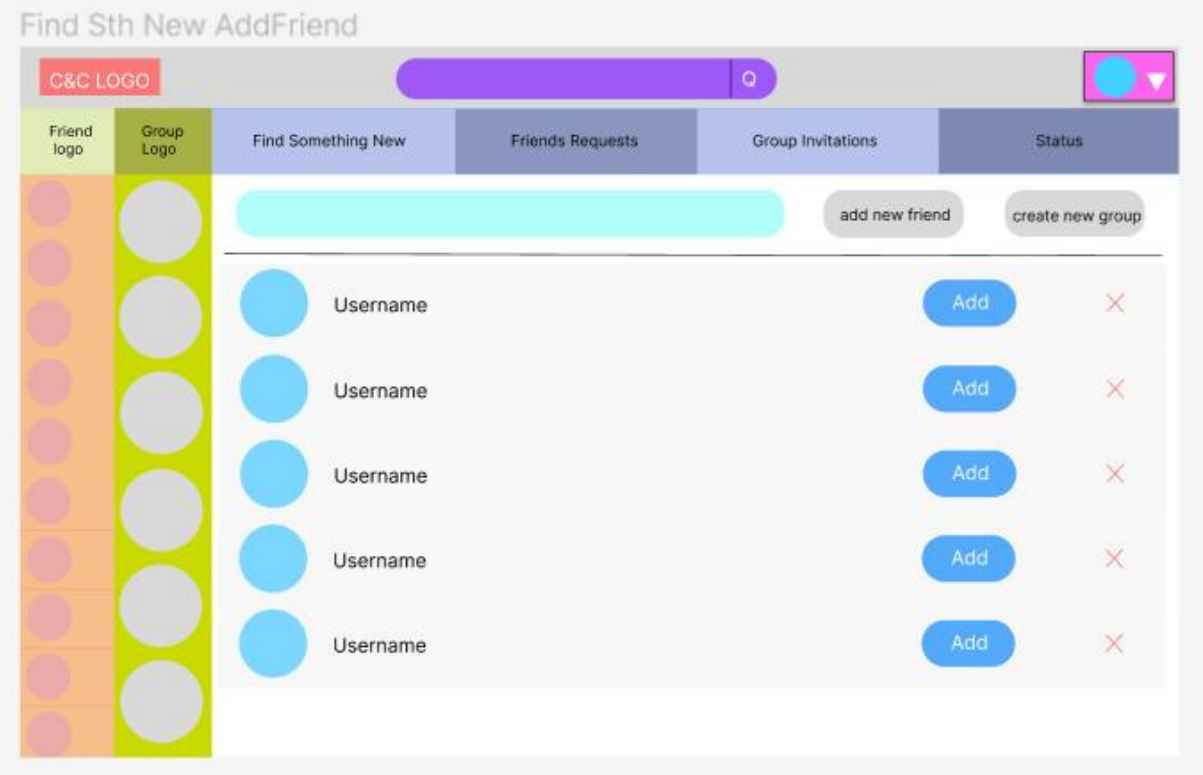



Figure 5.6.2.5 - “Find Something New Add Friend” page

“Find Something New Add Friend” page (Figure 5.6.2.5) is shown when the user wants to find a new friend, then can input the name in the search bar, then the C&C system will find all the similar names to help the user find this person.

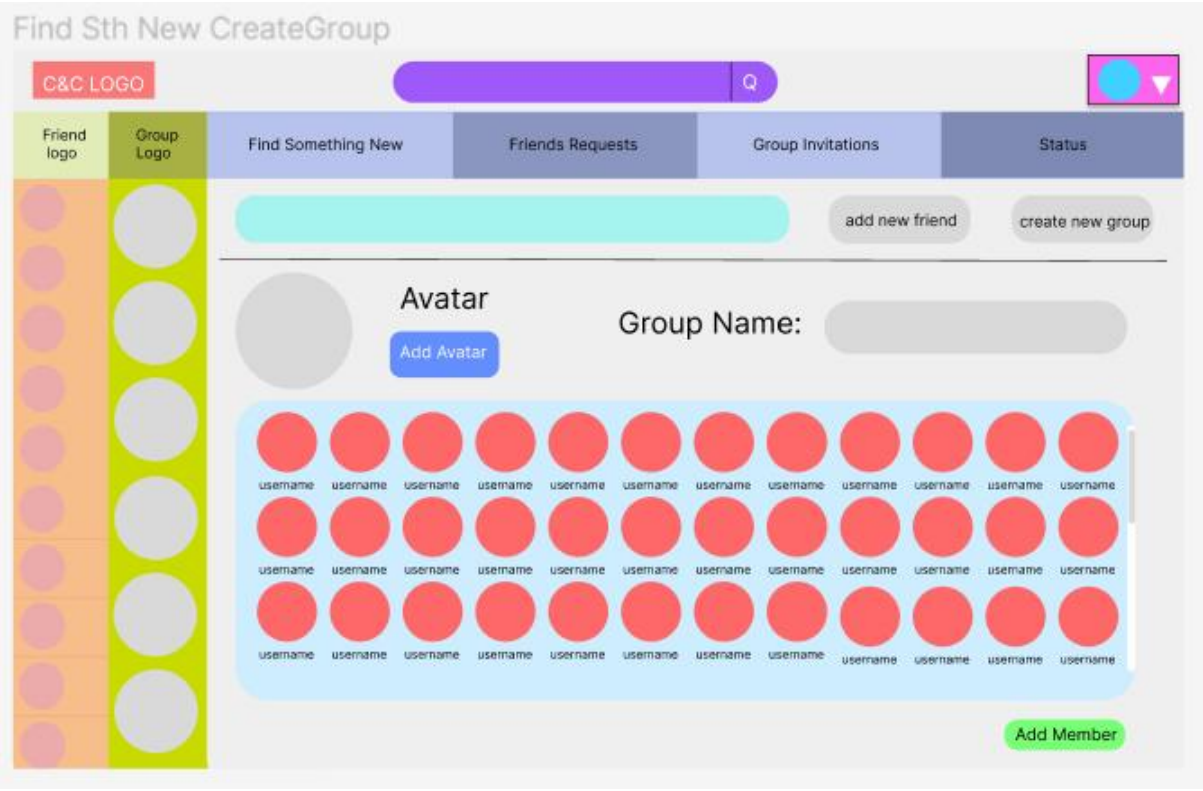


Figure 5.6.2.6 - “Find Something New Create Group” page

“Find Something New Create Group” page (Figure 5.6.2.6) is used when the user wants to create a new group, then the user can choose an avatar for the group and give a name to the group. By clicking the “Add Member” button, the selected friends will be shown in the scroll component, then the selected friends will get a group invitation.

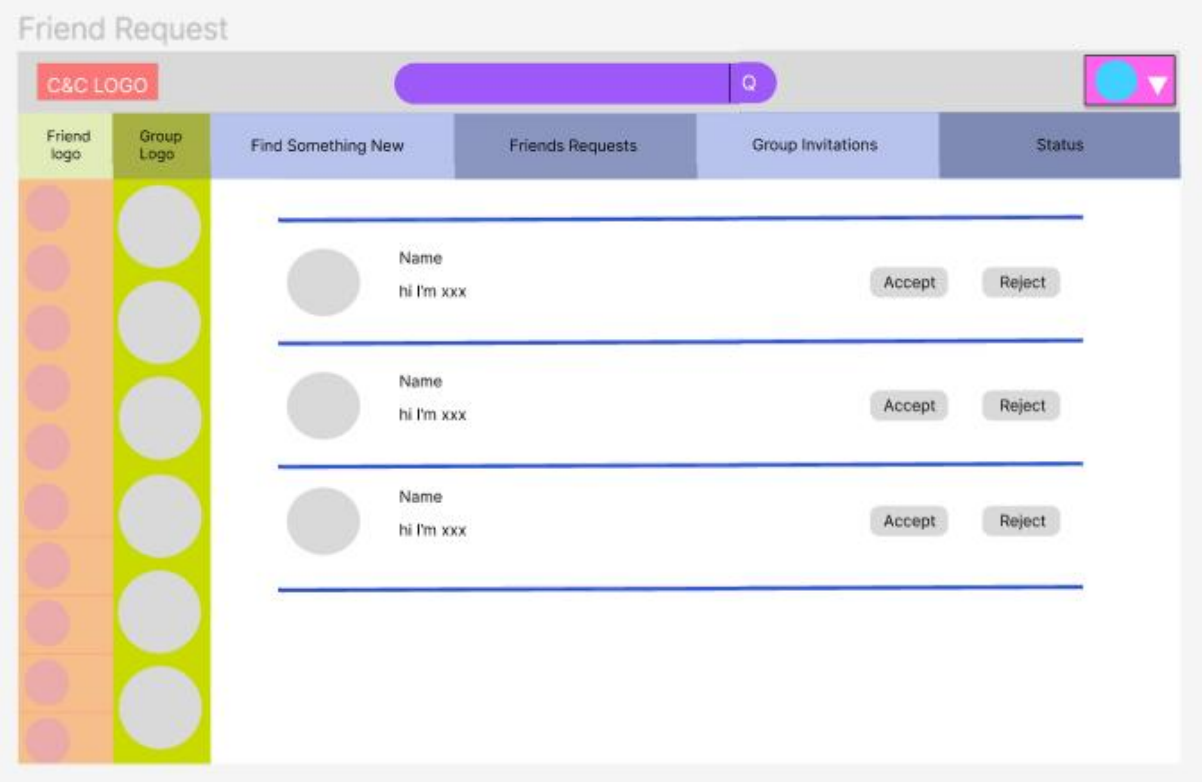



Figure 5.6.2.7 - “Friend Request” page

“Friend Request” page (Figure 5.6.2.7) is used for showing all the pending friends, the user can choose to accept or reject the friend request.

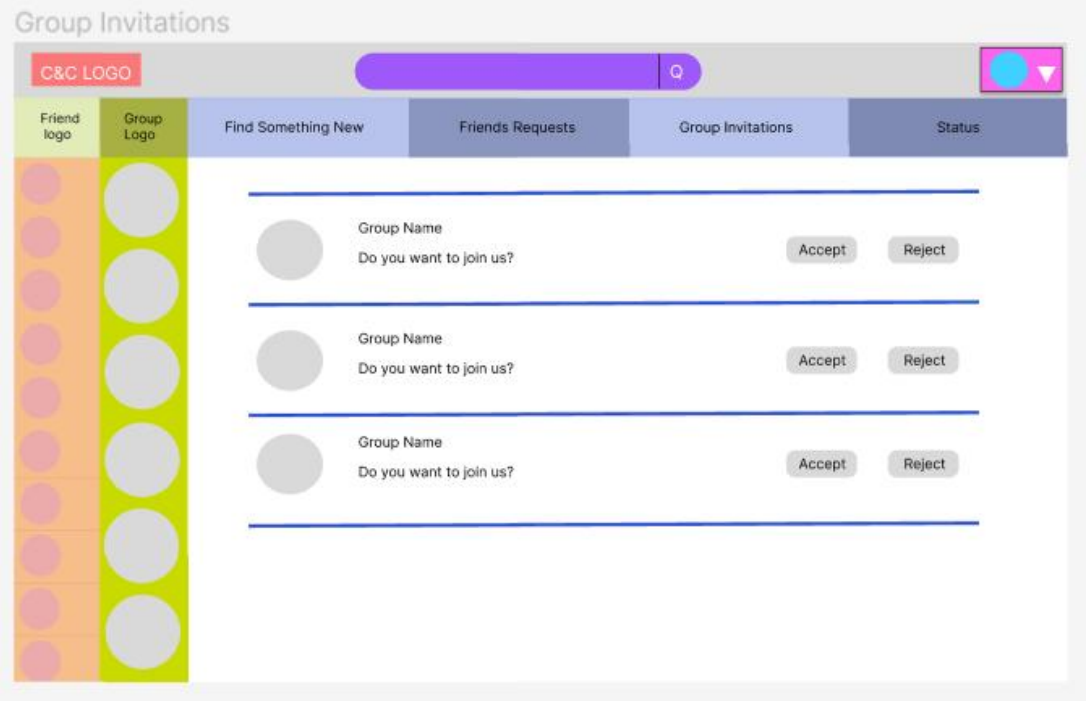



Figure 5.6.2.8 - “Group Invitations” page

“Group Invitations” page (Figure 5.6.2.8) is used for showing all the pending group invitations, the user can choose to accept or reject the group invitations.

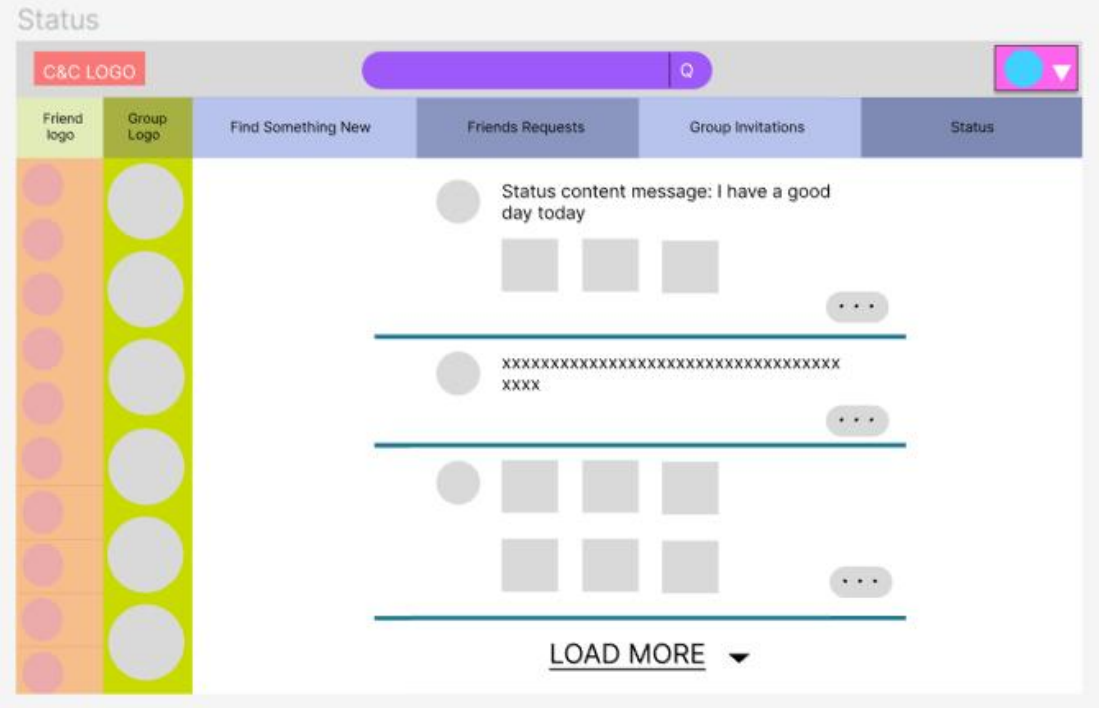



Figure 5.6.2.9 - “Status” page

“Status” page (Figure 5.6.2.9) is used to show all the statuses that the user can see, user can do a “like” action or comment on the status.

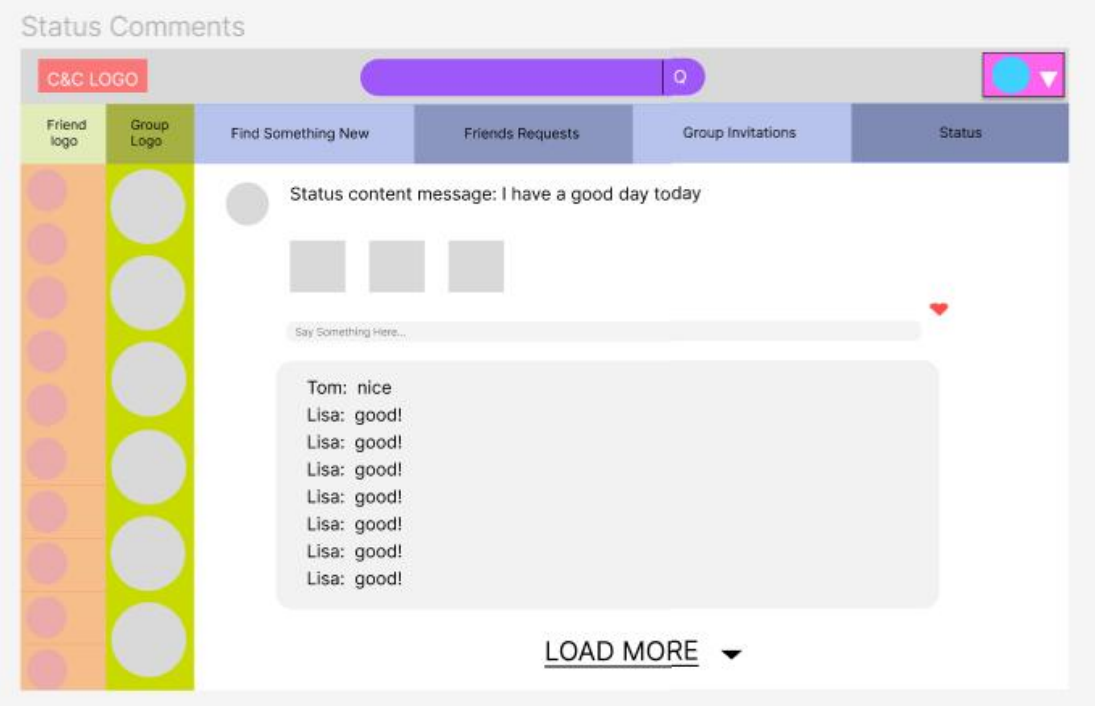



Figure 5.6.2.10 - “Status Comments” page

“Status Comments” page (Figure 5.6.2.10) can show all the comments in detail, the user can see comments from common friends.

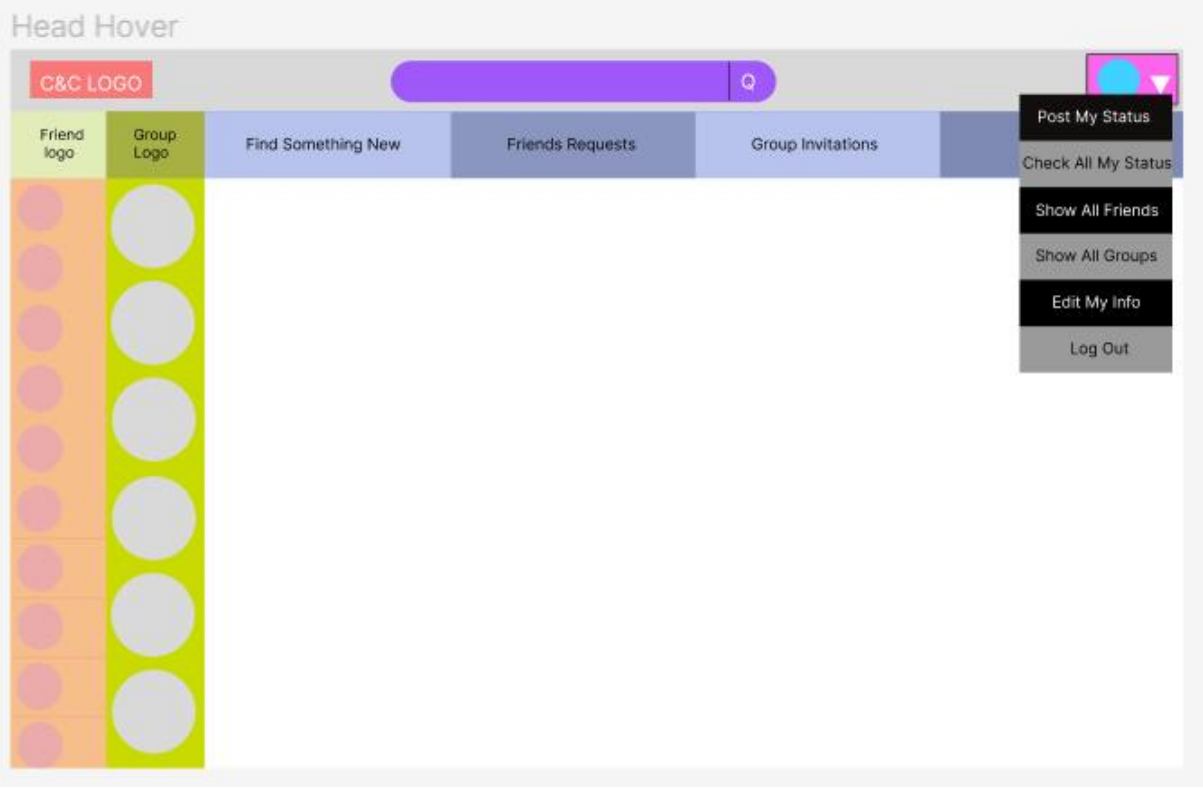



Figure 5.6.2.11 - “Head Hover” page

Head hover is shown in Figure 5.6.2.11, the head hover can post status, check users' own status, edit users' own info and log out. Besides that, the "show all friends" and "show all groups" can refresh users' friend list and group list.

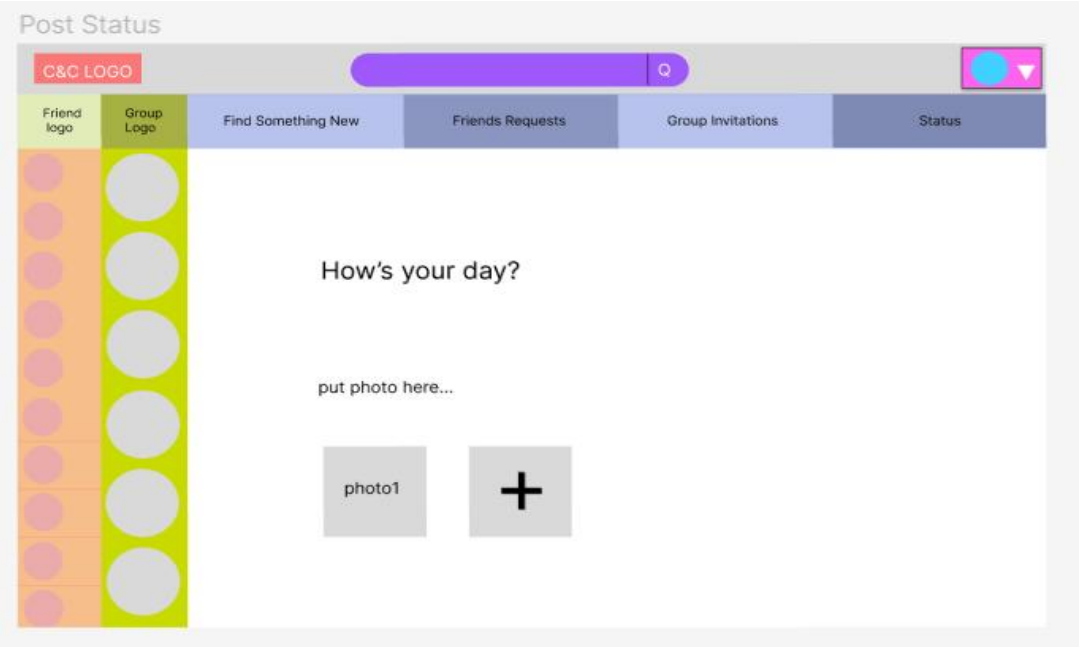



Figure 5.6.2.12 - "Post Status" page

Users can post their status through the "Post Status" page (Figure 5.6.2.12).

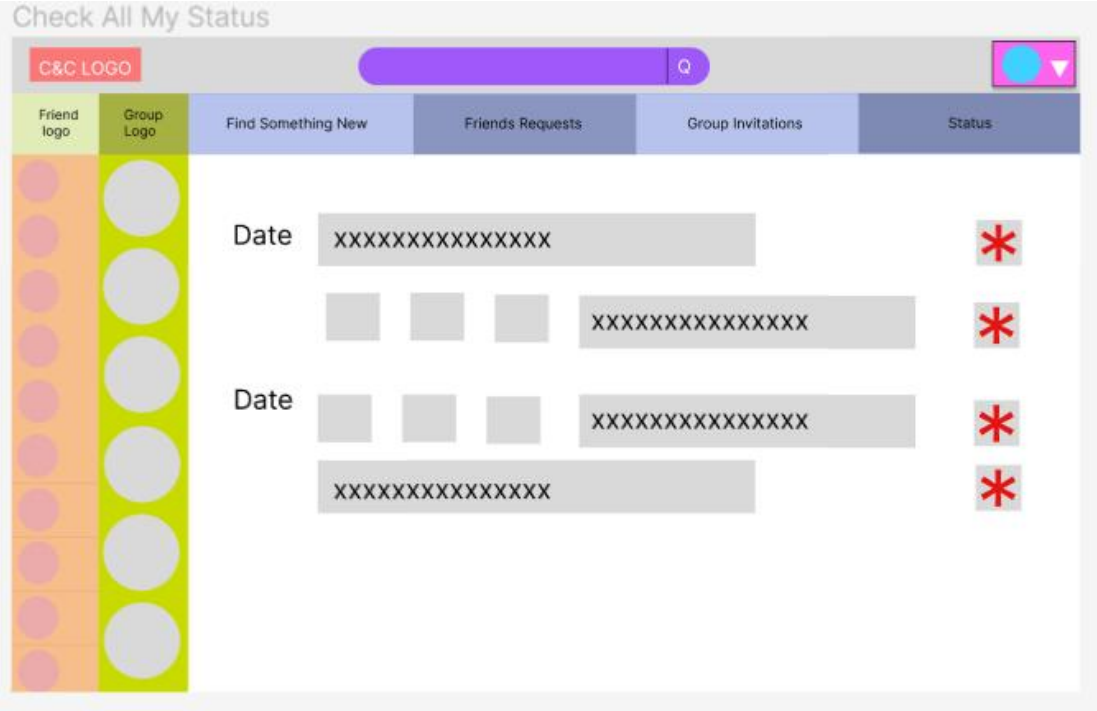



Figure 5.6.2.13 - "Check All My Status" page

Users can check all their past statuses or delete their statuses on "Check All My Status" page (Frigure

5.6.2.13), the status on this page will be shown with the post date.

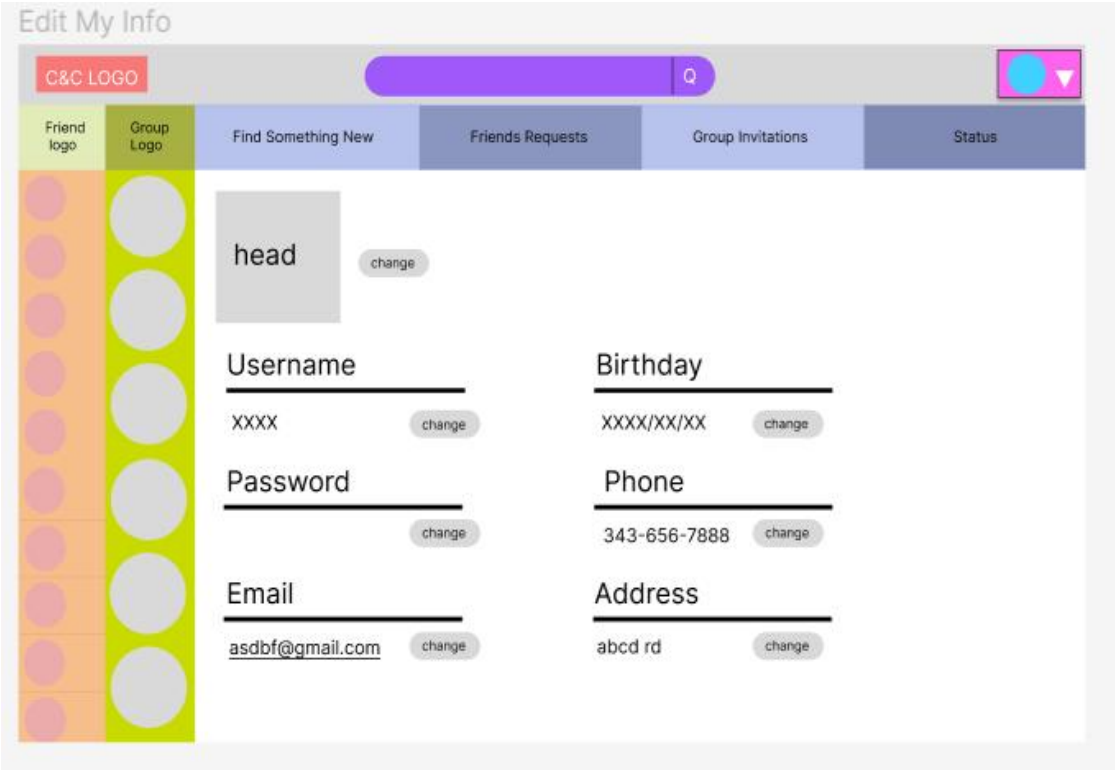



Figure 5.6.2.14 - “Edit My Info” page

Users can edit their personal information on “Edit My Info” page (Figure 5.6.2.14).


Figure 5.6.2.15 - “Friend and Group Hover” page

Friend and group hover are shown in Figure 5.6.2.15. The friend hover shows the friends’ names on top, users

can choose to hide, mute or delete this friend. Users can use the group hover to show the group member list, hide the group, mute the group or leave the group. After the user chooses to mute the friend or the group, the mute button will become unset. After the user chose to hide the friend or the group, they can use "show all friends" and "show all groups" in the head hover to show those friends or the group again.

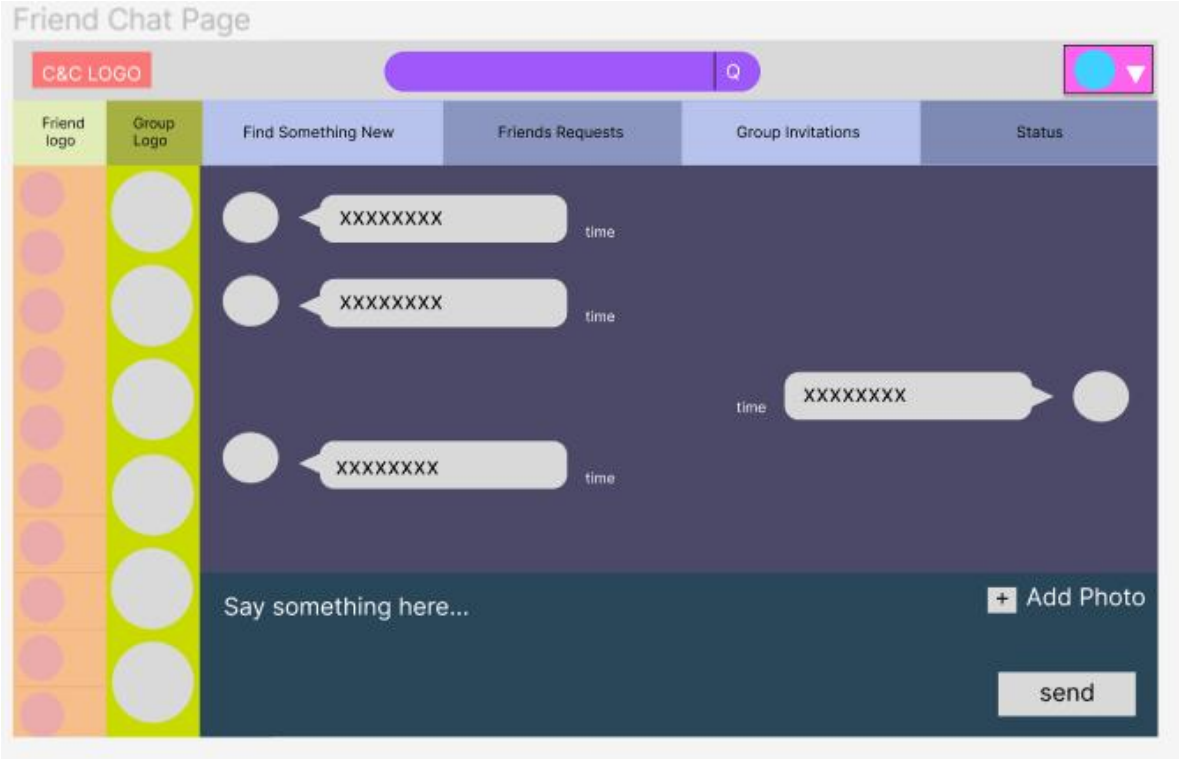



Figure 5.6.2.16 - "Friend Chat" page

"Friend Chat" page (Figure 5.6.2.16) support the users to chat with other people, and upload photoes while chatting. This page will also show the time that the message has been sent and the sender's avatar. By clicking the friend's avatar, user can get into this friend's info page.

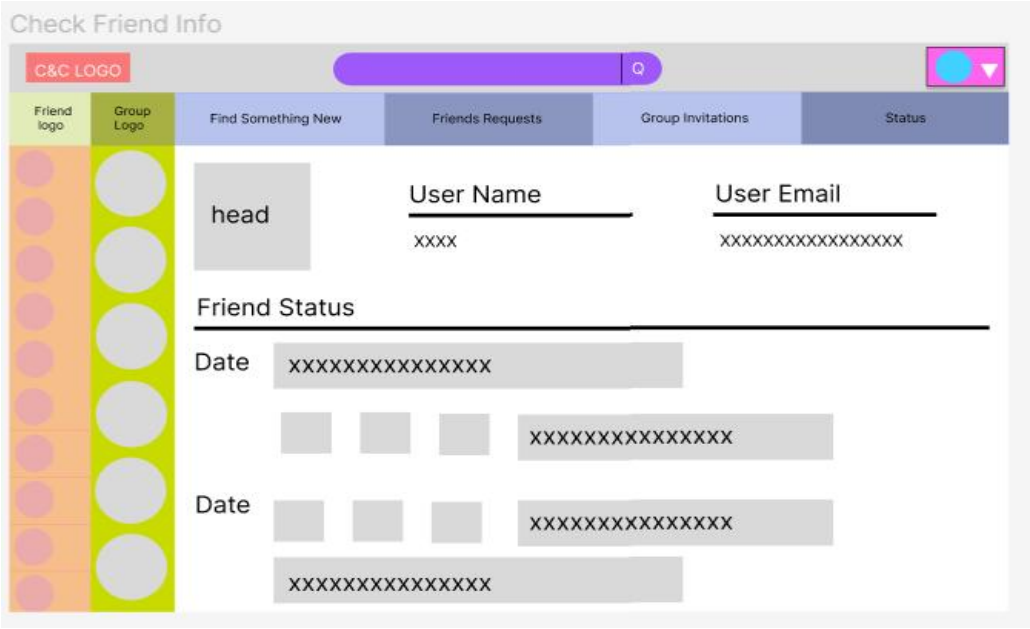



Figure 5.6.2.17 - “Check Friend Info” page

“Check Friend Info” page (Figure 5.6.2.17) show some of the friends’ information and their statuses with post dates.

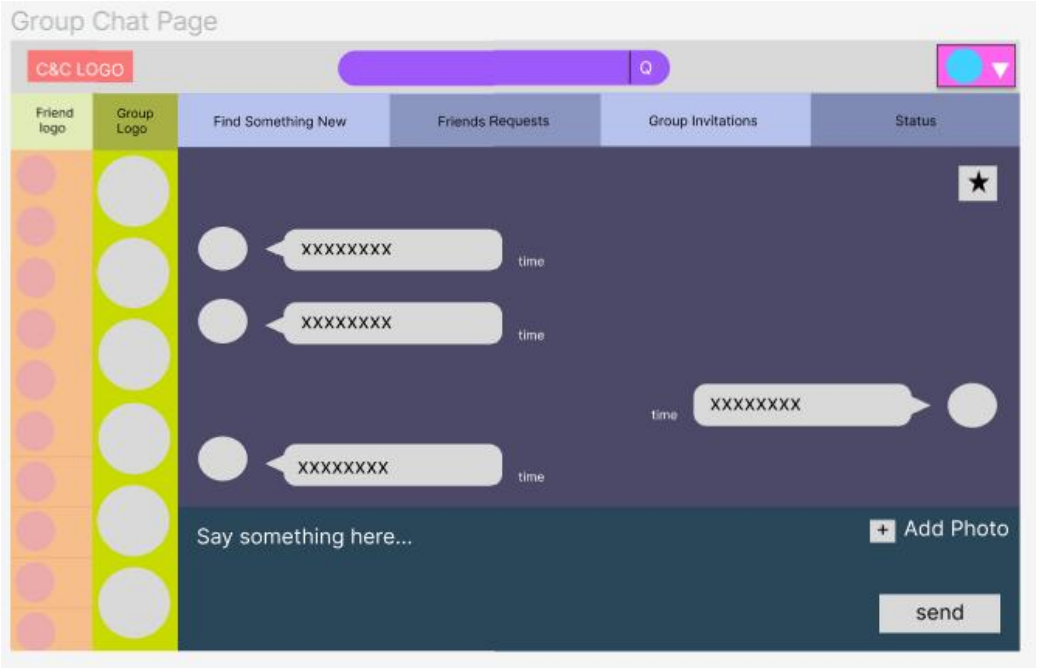



Figure 5.6.2.18 - “Group Chat” page

“Group Chat” page (Figure 5.6.2.18) is similar to the “Friend Chat” page, the difference is, this time the user cannot see personal info by clicking the avatar but they can click the star on the right corner to access to the group setting page.

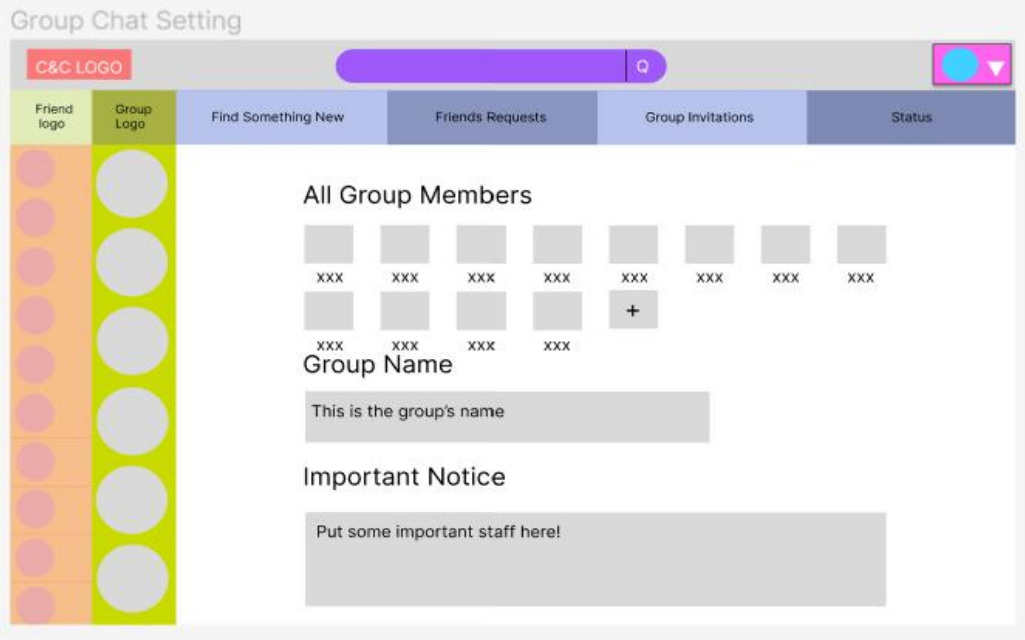



Figure 5.6.2.19 - “Group Chat Setting” page

“Group Chat Setting” page (Figure 5.6.2.19) shows all the group members, group name and important notice for this group, on this page, users can invite their friends to the group, the little add icon will pop an invitation window.

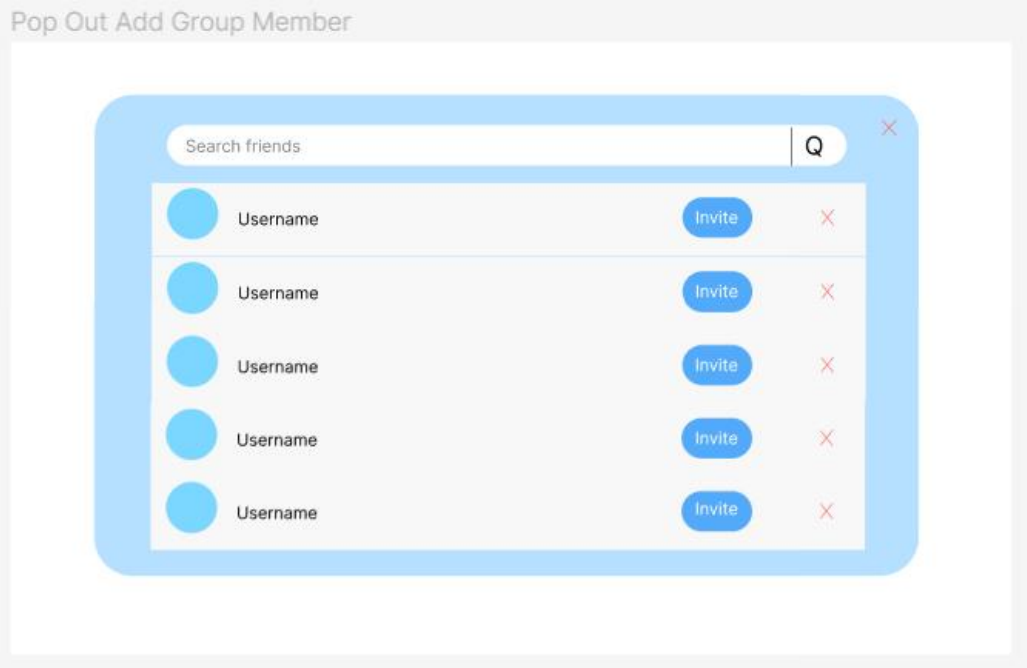



Figure 5.6.2.20 - “Pop Out Add Group Member” page

“Pop Out Add Group Member” page (Figure 5.6.2.20) has a scrolling page to show users’ friends that can be invited, and the users can search for their friends’ name to send the invitation.

## 5.6.3 UML

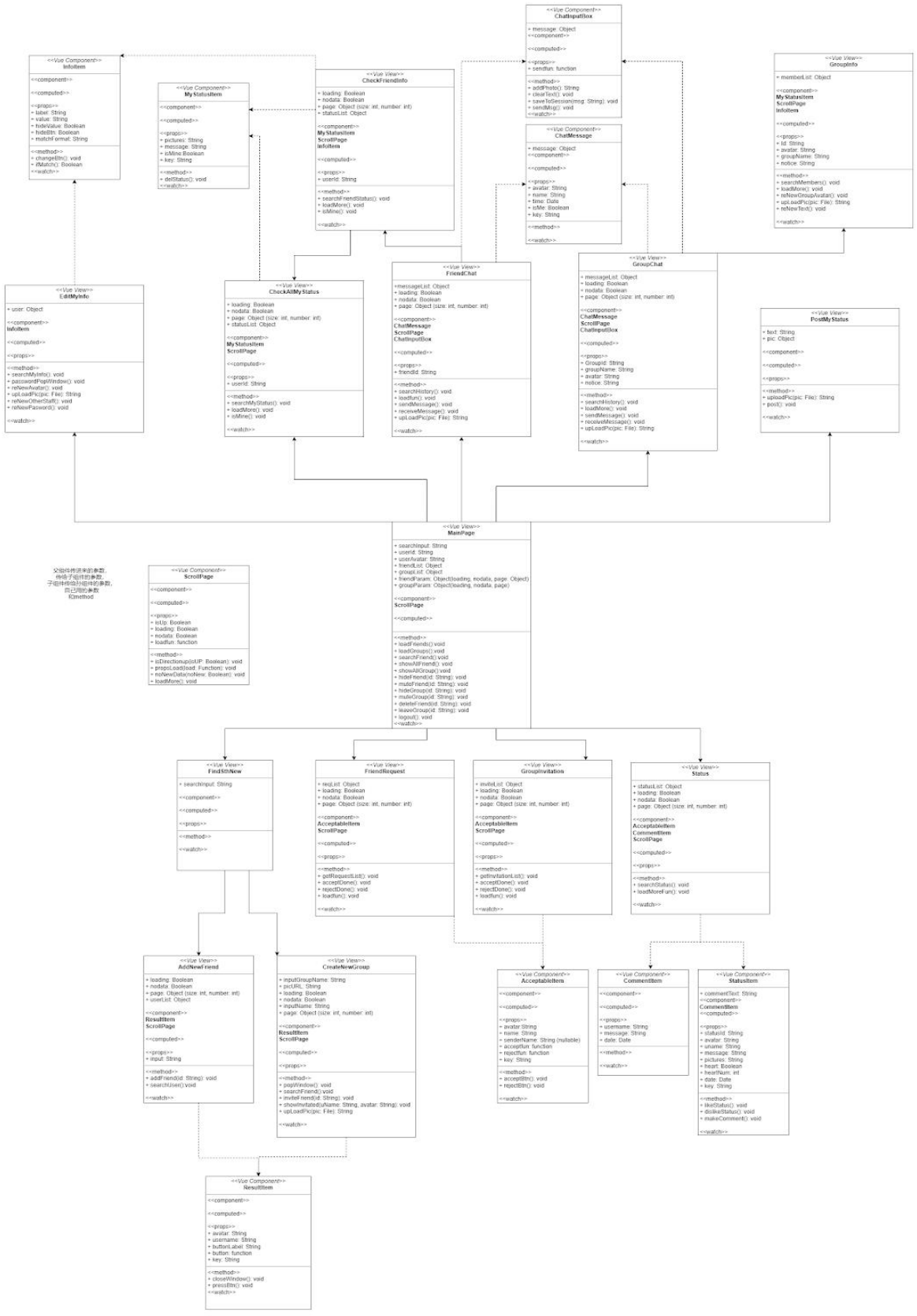



Figure 5.6.3.1 - Front-end UML

Depending on the page jumping rules described in page structure document (5.6.2), a simple front-end UML is designed, which can be found in User Front-End UML in Appendix 2. Source Documents . Arrows pointing directions explains the jumping sequences, for example, jumping from MainPage to EditMyInfo is possible.

Each box represents a class, and the very top few lines are the local variables of the class. The component part shows all reused classes in the current class. The props part shows all parameters given by the class's parent. The computed part lists all parameters that change instantly, and the method part shows all local functions. The watch part list all parameters that are monitored by the class.

Although, most front pages structures are designed well in the front-end UML, small changes will occur during implementation.

## 5.7 API Documents

The team has already designed the APIs that we can think about to build this C&C system, all the APIs are saved in the API document(api_v1.4.md). The explanation of a sample API  is shown in (figure 5.7.1).

This API will be used in the user login page, and it is corresponding to the userController in the login() method. The request type of this sign in API is PUT, it will ask for the username and the password in a string from the backend, and the data that is returned will tell if the request succeeds, sending the error code and the succeed message back and returning the token of the data (figure 5.6.4).

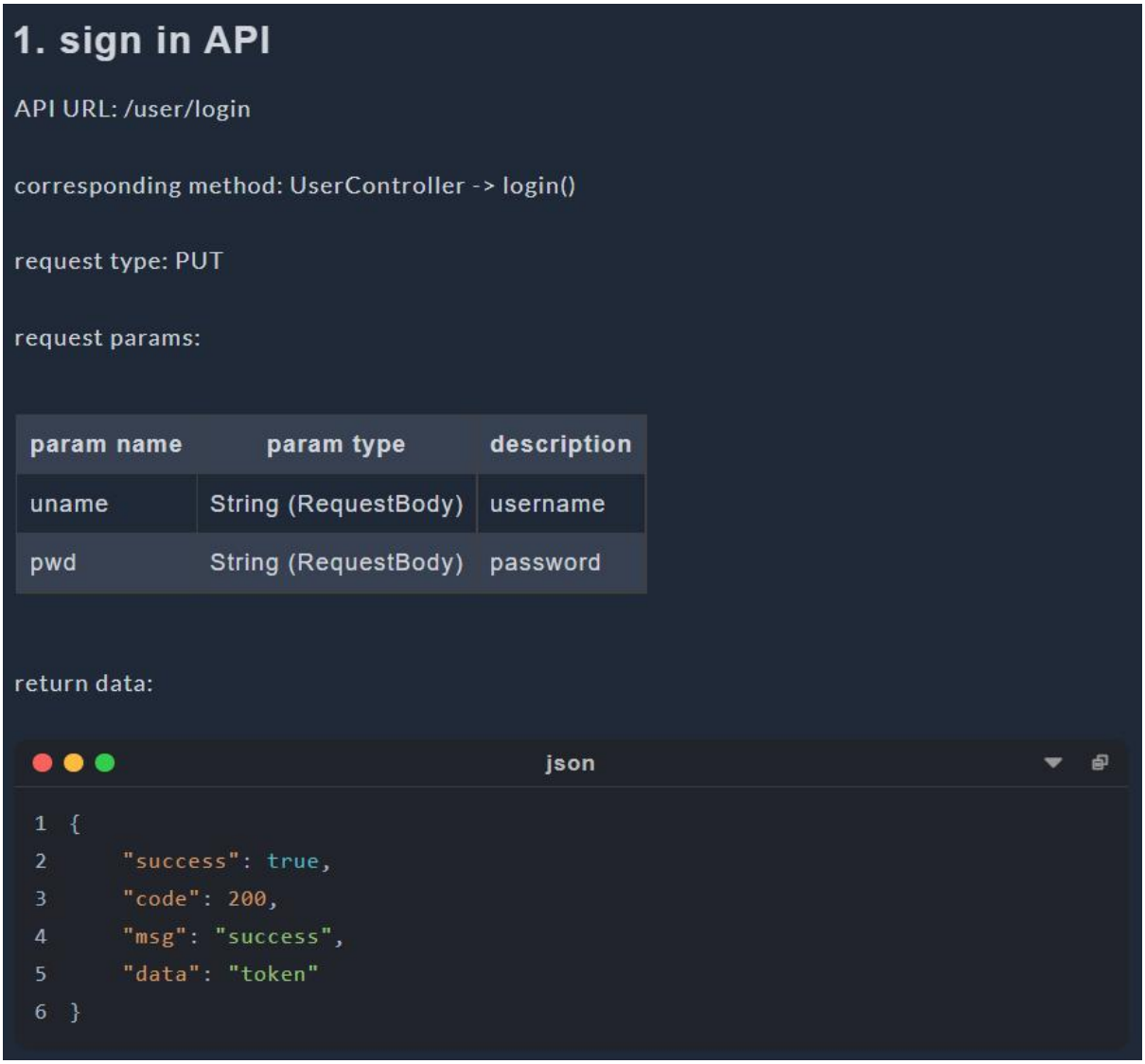



Figure 5.7.1 - “Sign In” API

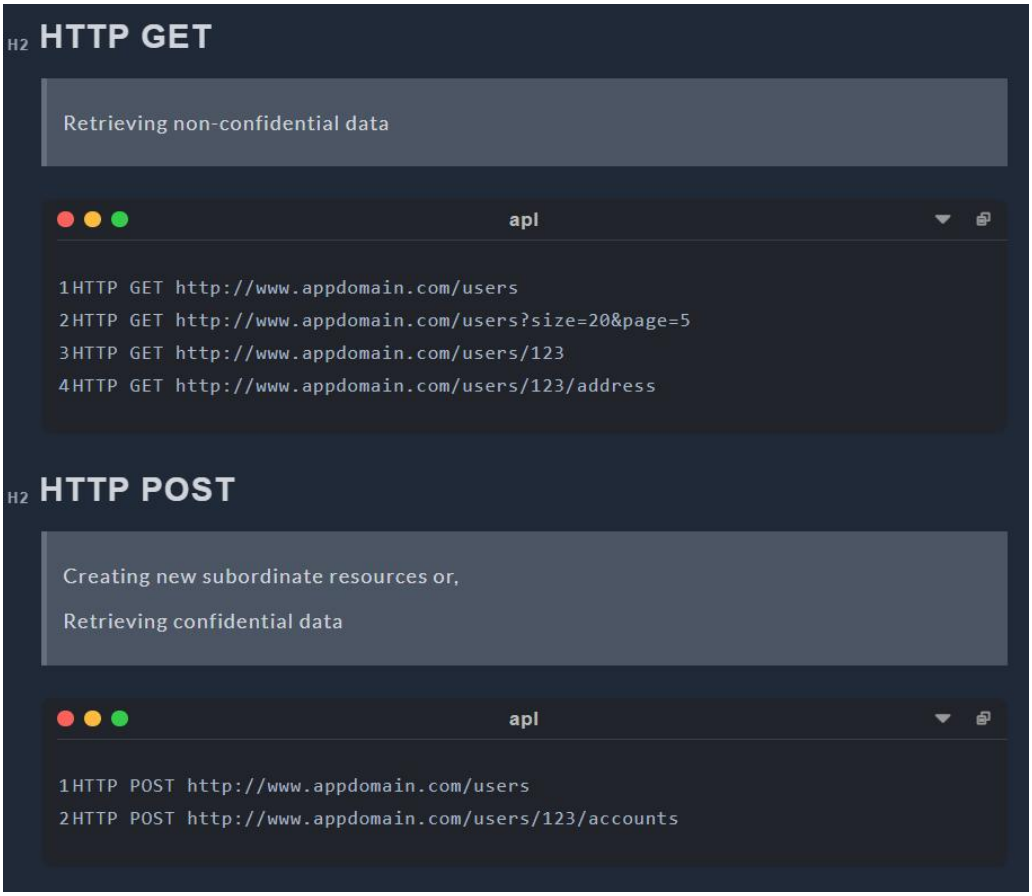



Figure 5.7.2 - HTTP Request Structure 1

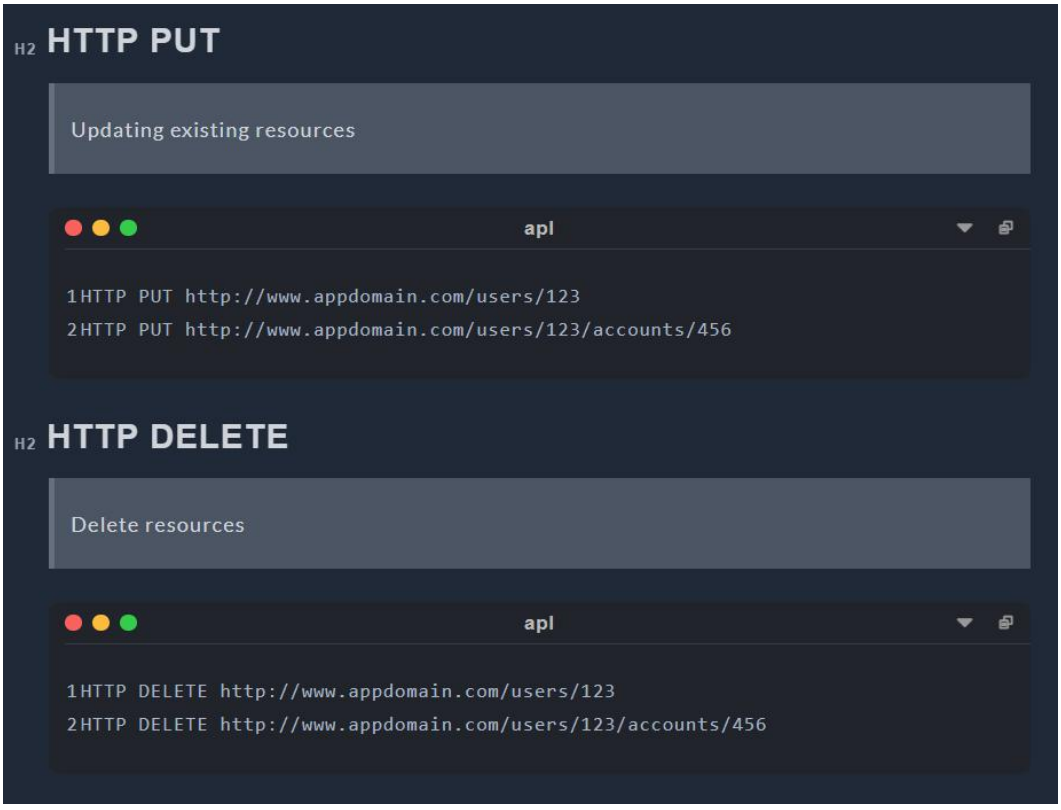



Figure 5.7.3 - HTTP Request Structure 2

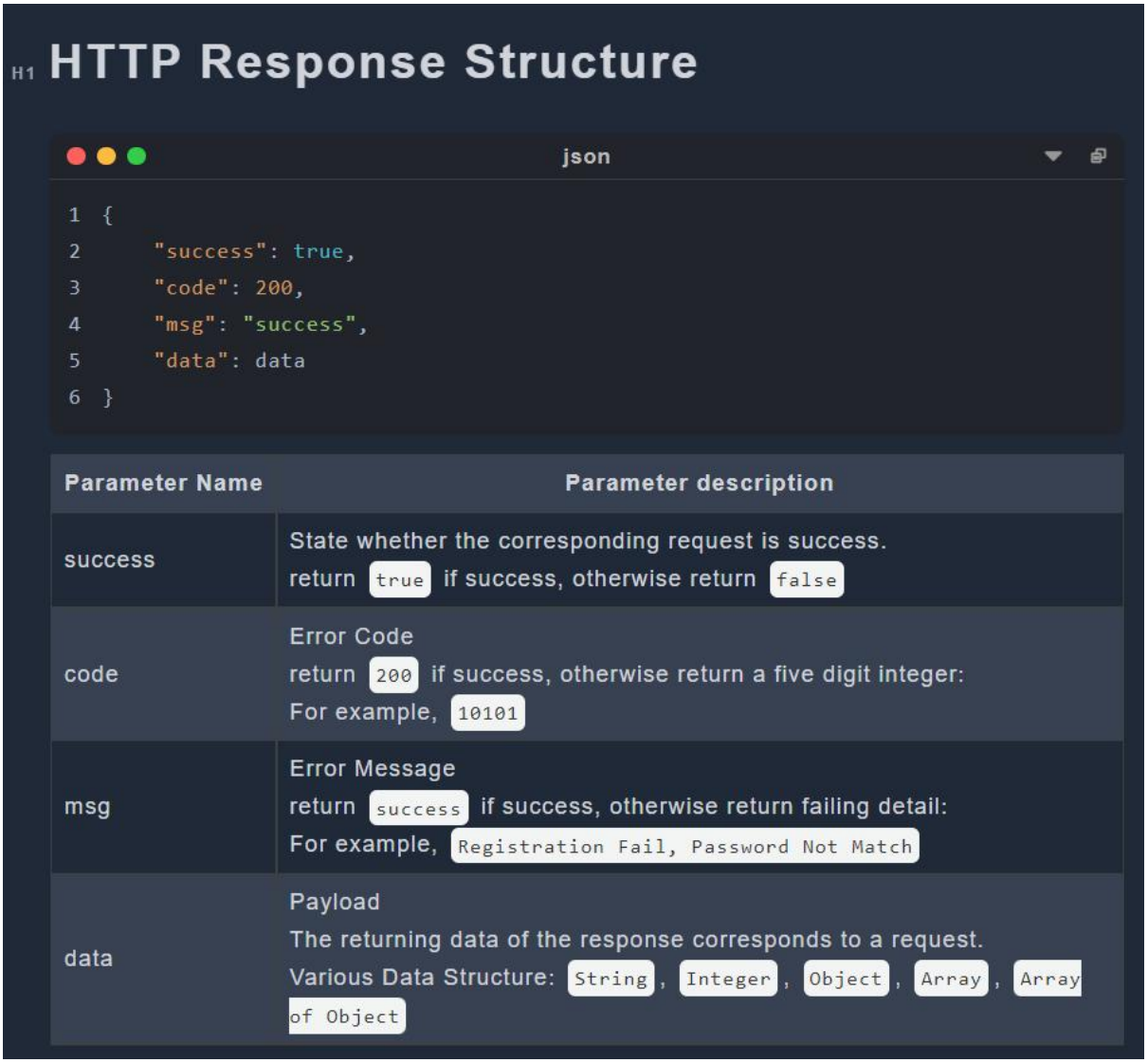



Figure 5.7.4 - Explanation of the returning data

# 6. Tools Selected and Justification

Implementation of C&C has not start yet. We focus on choosing the reasonable techniques which will be used in the implementation.

## 6.1 Java

Specifically, the Java language has the following advantages:

1. Java is a pure object-oriented language. It can directly reflect the objects in real life, such as animals. Therefore, it is easier for developers to understand and write programs.

2. Java language can be used on multiple platforms. Java programs can be compiled on Windows platform, Linux, MacOs and other platforms. The compiled programs can run on other platforms.

3. Java provides a lot of built-in class libraries, which can simplify the programming work of developers and shorten the development time of projects.

4. It provides support for Web application development. For example, Servlets and JSPs can be used to develop Web applications.

5. The C++ language has removed features that are difficult to understand or easily confused, such as header files, pointers, structures, units, operator overloads, virtual basic classes, multiple inheritance, etc., making the program more rigorous and clean. [25]

Using java technology can help us reduce unnecessary repetitive work when making this project, and the database provided by java can help us reduce the working time.

## 6.2 Spring

Spring is an open source framework. One of the main advantages of the framework is its hierarchical architecture, which allows users to choose which component to use. Spring uses basic JavaBeans to do things that previously could only be done by EJBs. However, Spring's use is not limited to server-side development. From the perspective of simplicity, testability and loose coupling, any Java application can benefit from Spring.

1. Declaration of support
In Spring, we can get rid of the transaction management code and flexibly manage transactions in a declarative manner to improve development efficiency and quality.

2. Convenient program testing
In Spring, testing is no longer a complex operation, but a very simple thing. For example, Spring supports Junit4 and can easily test Spring programs through annotations.

3. Convenient integration of various excellent frameworks
Spring does not exclude various excellent open source frameworks. On the contrary, Spring can reduce the difficulty of using various frameworks. Spring provides direct support for various excellent frameworks.

4. Reduce the difficulty of using Java EE APIs
Spring provides an encapsulation layer for many Java EE APIs that are difficult to use. Through the simple encapsulation of Spring, the difficulty of using these Java EE APIs is greatly reduced. [26]
In this project, we use spring because spring is used by many people. Other users have written many spring based libraries and new technologies. These technologies can make it easier for us to deal with projects.

## 6.3 MYSQL

The main advantages of MySQL are as follows:
1. Speed: fast operation.

2. Price: MySQL is free for most individuals.

3. Easy to use: Compared with the setting and management of other large databases, it is less complex and easy to learn.

4. Portability: It can work on many different system platforms, such as Windows, Linux, UNIX, Mac OS, etc.

5. Rich interfaces: APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, Tcl and other languages are provided.

6. Security and connectivity: Host based authentication is allowed. When connecting to the server, all password

transmissions are encrypted to ensure password security. Because MySQL is networked, it can be accessed anywhere on the Internet to improve the efficiency of data sharing. [27]

In this project, we use MySQL because it is a very popular and free technology. We can use MySQL to reduce our spending on projects and find rich learning resources.

## 6.4 Redis

A user is unlikely to send only one request after logging into C&C. Therefore, a lot of time is wasted frequently retrieving data from MYSQL. To speed up the generation of responses to requests, it is helpful for the C&C system to keep useful data for an extended period of time instead of retrieving it from MYSQL every time it needs it.

NoSQL Database solves this problem perfectly; they store temporary data in a RAM or cache and provide data when software needs them. Retrieving data from RAM (NoSQL Database) is a million times faster than the same operation from SSD/HDD (Relational Database). Therefore, to speed up the response, C&C stores frequently used data in the NoSQL database, for example, user tokens. There are two famous NoSQL databases: Redis and MongoDB. Redis is faster and uses fewer resources, but most of its APIs are atomic, which means it could take time to write customized APIs for C&C to use it. Redis stores data in the RAM only in the Key-Value pair and does not support other query languages. MongoDB is slower and uses more resources; while it owns APIs with various functions, it is undoubtedly more straightforward to use than Redis. MongoDB store data in Key-Value pair but allow multiple keys and support many query languages, even JSON [28].

Redis states that the MongoDB database needs Redis when the following requirements are required [29]:

- Query optimization (caching)
- Session management
- Real-time analytics
- High-speed data ingestion
- Message brokering and queues

The comparison between Redis and MongoDB is very similar to that between C and Python; both Redis and C have higher performance, while MongoDB and Python are easier for developers. C&C will not store complex data in RAM; the speed is the most relevant. Therefore, Redis is chosen. However, if C&C is a huge, sophisticated software and stores various types of data in RAM or cache, MongoDB is the better choice.

## 6.5 Vue3

Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that helps you efficiently develop user interfaces, be they simple or complex [30].

Vue is designed to be flexible and incrementally adoptable, depending on the use case, Vue can be used in different ways:

- Enhancing static HTML without a build step
- Embedding as Web Components on any page
- Single-Page Application (SPA)
- Fullstack / Server-Side Rendering (SSR)
- Jamstack / Static Site Generation (SSG)
- Targeting desktop, mobile, WebGL, and even the terminal

We choose it to build the C&C web-pages, because compared with the traditional HTML, Vue contains HTML, CSS and JavaScript together, so it decrease the possibility of using the wrong path while building the user interface. Also, Vue has a better structure than HTML, instead of having three files for a single page, Vue page contain all of HTML, CSS and JS codes in the same file, the size of the C&C system can be reduced significantly.

## 6.6 Element-Plus & Tailwind

Element-Plus and Tailwind will be used for designing the webpages, these two components will take the place of the traditional CSS while developing the webpage.

Element-Plus is a Vue 3.0 based component library for developers, it contains many built-in beautiful visual components that can be used to decorate the webpage. Since Vue 3.0 is being used to build the system, Element-Plus is suitable for design. It also provides many fancy webpage building functions that can be used directly [31].

Tailwind is an API for the design system, different from CSS that we need to use big blocks for each class, Tailwind can reduce those classes into one line of code. Each part of the webpage can have its own style, instead of collecting all the style parts and putting them together, the high flexibility of it also allows customized web-pages [32].

Although both Element-Plus and Tailwind simplify the process of decorating front pages dramatically, neither of them can replace CSS for the reason that some customized requirements cannot be proceeded by them. CSS is still an indispensable decorating tool for C&C project.

## 6.7 HTTP and WebSocket

HTTP represents for Hypertext Transfer Protocol which is an application-layer protocol for transmitting hypermedia documents, such as HTML [33] and JSON. Through it, data can be transferred between Front-End and Back-End easily. In C&C, most of the transferred data are in format of JSON [34]. However, only the client can send HTTP requests, not the server, and there must be an HTTP request before an HTTP response.

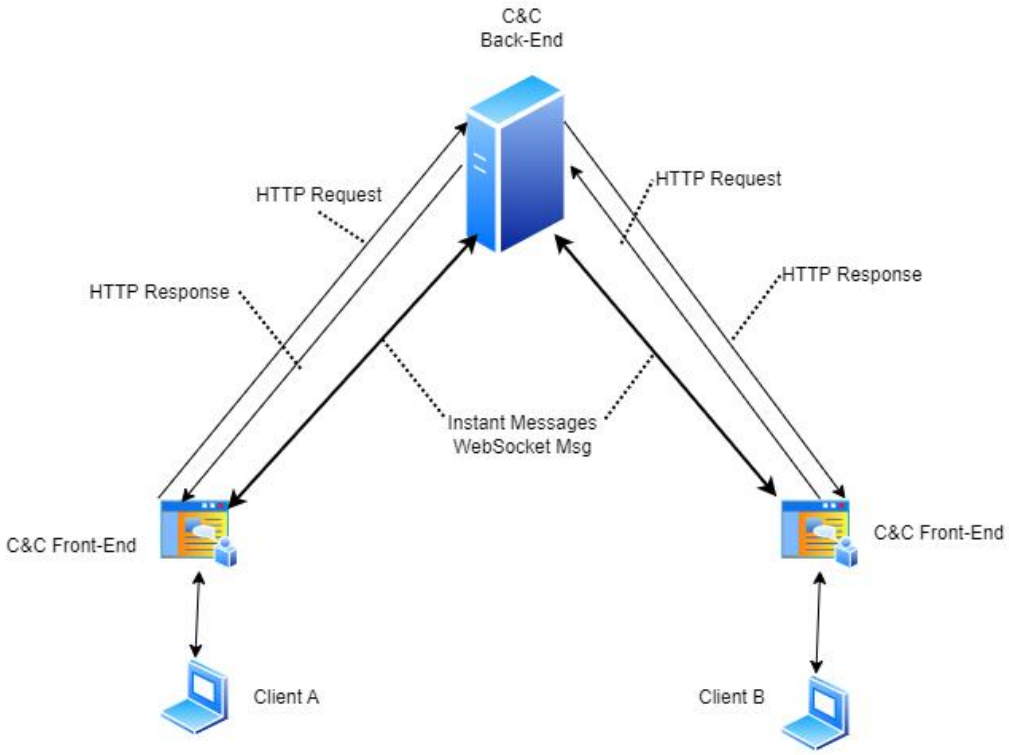



Figure 6.7.1: Data Flow Diagram of C&C

There is no direct connection between Client A and Client B users in the C&C system in Figure 6.7.1. The only way for A to see messages sent by B is for the server to send messages to A without being requested. However, servers cannot send unsolicited HTTP messages to clients, while client A needs to know messages sent by client B immediately. To be dumb, we could solve this problem by generating a set of HTTP requests to the server

once every second. But this would seriously increase the pressure on the server, and a one-second delay is a long time for instant messaging, so such a solution doesn't work.

To solve the problem, there are two possible solutions according to our investigation: WebSocket and WebRTC.

WebSocket is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server [35]. WebRTC represents for Web Real-Time Communication which is a technology that enables Web applications and sites to capture and optionally stream audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary [36]. Both technology allows clients connect with each others.

Briefly, WebSocket need a intermediary to create TCP connection among clients while WebRTC create p2p UDP connection (peer to peer) directly between two clients. UDP connection [37] could lose some data during transmittion while 100% transmission integrity guaranteed for TCP connection [38]. The guarantee of transmission integrity can take a lot of time, so that UDP transmission will be faster than TCP.

In terms of performance, WebRTC is much faster than WebSocket, as it connects directly to its peer using UDP, while WebSocket first connects to the server and then connects to another client from the server using TCP. Conversely, WebSocket has a clear advantage in terms of data integrity.
In C&C, a chat messages encoding requirement is described in sub-section 2.1.8-Optional Functions; C&C may need to control whether or not chat messages are encoded among users. A chat history viewing function is described in term 5 of sub-section 2.1.4-Chat Functions; C&C need to store all sent chat messages for each user. To do message encoding, decoding and recording, we cannot put the three steps in front-end for security reasons, back-end it the only choice, therefore, WebSocket is the better choice of C&C.

In the future, C&C may add video and voice calling capabilities, and WebRTC would be a good choice for implementing these features.

## 6.8 Restful API

To handle HTTP requests, it is essential to have a consistent architecture for APIs. Fortunately, many API architectures have been developed by predecessors. According to out investigation, there are 3 main API architectures: REST, SOAP and RPC.

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer, is a set of guidelines for scalable, lightweight, and easy-to-use APIs [39]. The guidelines of REST are:

- Client-Server Separation: Separate client from server.
- Uniform Interface: Use specific HTTP format.
- Stateless: Each client-server interaction is independent of every other interaction.
- Layered system: Requests and responses must always be formatted the same way.
- Cacheable: Server responses should indicate whether a provided resource can be cached by the client and for how long.

The design of C&C which separates client and server is a guideline for REST. As the most famous API architecture, it perfect for our web-based instant messaging software — C&C.

SOAP stands for Simple Object Access Protocol which relies heavily on XML, and together with schemas, defines a very strongly typed messaging framework. SOAP strictly defines how messages should be sent and what must be included. This makes SOAP APIs more secure than REST APIs, although the rigid guidelines make them more code-heavy and harder to implement [40]. In C&C, various types of data structures are used in API transfer; therefore, despite its safety features, it is not a good choice for the C&C project.

The RPC (Remote Procedural Call) protocol is the most straightforward of the three architectures. Unlike REST and SOAP, which facilitate data transfer, RPC APIs invoke processes. In other words, they execute scripts on a server. RPC APIs are limited in security and capabilities, so it is unlikely to see them as often as REST or SOAP APIs on the web. However, it can be used for internal systems for making basic process requests, especially many at once [41]. As an internal system-specific API architecture, it is not a suitable choice for the API architecture of C&C.

In conclusion, REST API or Restful API is the best choice among the three API architectures.

# 7. Implementation

## 7.1 Backend

### 7.1.1 MYSQL

We use Navicat to develop the database. Connect to the mysql database through the spring boot web project. We set the spring-boot-starter-websocket and mysql-connector-java dependencies in the pom.xml file. Then set the datasource in application.yml. Set the username, password, url, driver-class-name and type to connect to the database in the datasource.

### 7.1.2 Springboot

POJO VO

The classes in pojo are used to directly carry records in the database, using MyBatis-Plus and lombok libraries. Mybatis-Plus can complete the mysql language automatically. Use TableField and TableName annotations in pojo to help development. The TableField annotation is placed above the parameter. It can associate the parameter in the class with the field in the database table. TableName is placed above the class name to associate the class with the database table. The class in pojo also uses Lombok to simplify development. For example, the comment class in pojo

```
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableName;
import lombok.Data;

@Data
@TableName("tb_comment")
public class Comment {
  private String id;
  private String msg;
  @TableField(value = "status_id")
  private String statusId;
  @TableField(value = "date")
  private Long createDate;
  @TableField(value = "sid")
  private String userId;

  @TableField(exist = false)
  private String uname;
}
```

The class in the vo package is not directly associated with the database. Its function is only to transform information. For example, the information obtained by the controller after calling service is of pojo type. The controller needs to call the method of the class in vo to convert the data into vo type and send it to the front end.

Service

There are interfaces and implementation classes in the service layer. The interface defines the method, and the implementation class implements the specific method logic. The following is an example of the interface.

```
public interface CommentService {

  List<Comment> showCommentList(String statusId);

  boolean postComment(String statusId, String msg, String sid);

  boolean delComment(String statusId);
}
```

First, add a service annotation to the implementation class. This annotation can mark the current class as a service class. The advantage of this is that the controller can call the service directly. For example, the following.

```
@Transactional
@Service
public class CommentServiceImpl implements CommentService {

public class CommentController {

  @Autowired
  private CommentService commentService;
```

The Transactional annotation in the above example is used to start a transaction. This annotation allows you to test the system without modifying the database records. The principle is that all operations will be rolled back at the end of the test.

The method in service accepts parameters, and then uses the dao command provided by spring boot to directly operate on the database. For example, delComment accepts the parameter status Id in the following example. Method uses LambdaQueryWrapper to set the condition that the record in the statusId field of the comment table is equal to the parameter. Then complete the search through commentDao. selectList (lqw). After completing the search, a list will be returned. Delete the list by traversing it. Determine whether the operation is successful according to the int returned by the delete operation. Finally, a Boolean value is returned.

```
@Override
public boolean delComment(String statusId) {
  LambdaQueryWrapper<Comment> lqw = new LambdaQueryWrapper<Comment>();
  lqw.eq(Comment::getStatusId, statusId);
  List<Comment> Comments = commentDao.selectList(lqw);
  boolean flag = true;
  for (Comment comment : Comments) {
     int i = commentDao.deleteById(comment.getId());
     if(i <= 0) flag = false;
  }
  return flag;
}
```

### MyBatis-Plus

MyBatis-Plus can save a lot of time. All mysql languages can be automated. MyBatis-Plus is an enhancement tool for MyBatis, which is based on MyBatis. First, add MyBatis-Plus dependency in pom and xml. MyBatis-Plus provides methods for adding, deleting, and modifying the corresponding database. Through LambdaQueryWrapper and corresponding methods, all operations against the database can be completed. For example, the following code.

```
LambdaQueryWrapper<Comment> lqw = new LambdaQueryWrapper<Comment>();
lqw.eq(Comment::getStatusId,statusId);
lqw.orderBy(true,false, Comment::getCreateDate);
List<Comment> Comments = commentDao.selectList(lqw);

int a = commentDao.insert(Comment);

dataCounterDao.updateOnlineUserAmount(1);

commentDao.deleteById(comment.getId());
```

### Controller

The Controller class automatically assembles the service layer through Autowire in spring boot. for example

```
@Autowired
 private CommentService commentService;
```

The following example is the showCommentList method in CommentController. This method takes the status Id as a parameter and calls commentService to find qualified records. This method converts the record to vo format and sends it to the front end in the type of Result@ GetMapping ("list/{statusId}") indicates that the request sent by the front end needs to be a get request. "List/{statusId}" indicates that the format of the front-end request needs to start with a list followed by parameters.

```
@LogA
@GetMapping("list/{statusId}")
public Result showCommentList(@PathVariable String statusId) {
  CommentVo commentVo = new CommentVo();
  List<Comment> Comments = commentService.showCommentList(statusId);

  List<CommentVo> commentVos = new ArrayList<>();
  for (Comment comment : Comments) {
     CommentVo transfer = commentVo.transfer(comment);
     commentVos.add(transfer);
  }

  if(commentVos == null) {
     return Result.fail(ErrorCode.SHOW_COMMENT_LIST_ERROR);
  }
  return Result.success(commentVos);
}
```

Lombok

Lombok project is a java library that can be automatically inserted into editors and build tools to enhance the performance of java. There is no need to write getter, setter, or equals methods. First, add the Lombok dependency in the pom.xml file and then use it. In this project, we mainly use Data annotation, which integrates Getter, Setter, ToString, EqualAndHashCode, and RequiredArgsConstructor annotations.

### 7.1.3 Redis

```
spring:
    redis:
        host: localhost
        port: 6379
```

Figure 7.1.3.1: Redis Configuration

```
1 usage  zirui
@Configuration
public class RedisConfig {
    1 usage
    @Resource
    private RedisConnectionFactory factory;

    1 usage  zirui
    @Bean
    public RedisTemplate<String, Object> redisTemplate() {
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(new FastJsonRedisSerializer<>(Object.class));
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashValueSerializer(new FastJsonRedisSerializer<>(Object.class));
        redisTemplate.setConnectionFactory(factory);

        return redisTemplate;
    }
}
```

Figure 7.1.3.2: Redis Configuration Class

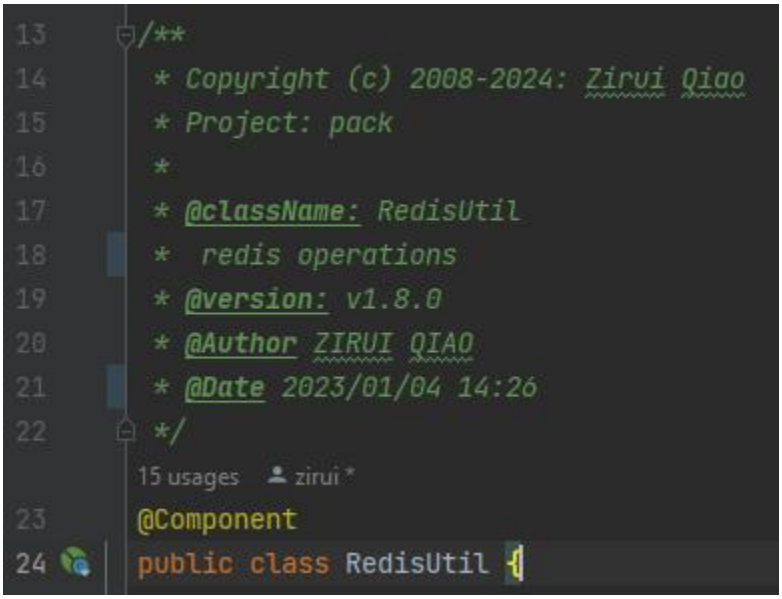


Figure 7.1.3.3: RedisUtil

The port for redis is set up in configuration file: “src/main/resources/application.yml” which is shown in Figure 7.1.3.1. Some addition configuration is set in redis related configuration class: “src/main/java/com/nart/config/RedisConfig.java” which is shown in Figure 7.1.3.2. Operations on redis are mainly carried out via the methods provided by RedisUtil (“src/main/java/com/nart/util/RedisUtil.java”) which is shown in Figure 7.1.3.3. Some general get, set and delete functions are declared, also some long type expiry times.

## 7.2 Frontend

### 7.2.1 HTTP requests and api

```
/*
 * @FileDescription: axios HTTP request general function
 * @Author: Zirui Qiao
 * @Date: 2022/12/25 14:13
 * @LastEditor: Zirui Qiao
 * @LastEditTime: 2023/01/01 21:56
 */
import axios from 'axios'
import { url } from './token'

axios.defaults.withCredentials = true;
const req = axios.create({
    baseURL: 'http://' + url,
    timeout: 10000
});
export default req
```
Figure 7.2.1.1: HTTP Request Handler

```
export function searchFriend(token, input, page) {
    return req({
        headers: {'Authorization': token},
        method: 'post',
        url: `/friend/search/${input}`,
        data: {
            pageSize: page.pageSize,
            pageNum: page.pageNum
        }
    })
}
```

Figure 7.2.1.2: Example API Function

All HTTP requests sent by front-end are handled by a single function in “request/index.js” which is shown in Figure 7.2.1.1. All APIs are encapsulated in corresponding files named after the request type which are stored in the ”api” folder. (For example, “api/user.js”). Each API stated the token, url, request type and data, an example of an API function is shown in Figure 7.2.1.2

## 7.2.2 Router

```
{
  path: '/',
  name: 'main',
  component: MainPage,
  children: [
    {
      path: '/findSthNew',
      name: 'findNew',
      component: FindSthNew,
      children: [
        {
          path: '/findSthNew/createGroup',
          name: 'createGroup',
          component: CreateGroup
        },
        {
          path: '/findSthNew/searchNew',
          name: 'addNewFriend',
          component: AddNewFriend
        }
      ]
    },
    {
      path: '/friendRequests',
      name: 'reqList',
      component: ReqList
    },
```

Figure 7.2.2.1: Router File Example

The page jumping rule is defined in file “router/index.js”, it defines which view will the designated part jump to. For each view or destination, four values are set: url path, name, actual view component and children. Url path

is the path that will display in the address bar, name is similar to a variable name for coding convenience, and component is the actual view file. Children attribute is default null and contains the page jumping rule of the new reached view. An part of the router file is shown in Figure 7.2.2.1.

```
129  router.beforeEach((to, from, next) => {
130    //console.log(localStorage.getItem('token'));
131    if (!localStorage.getItem('token')) {
132      if (to.name == "login" || to.name == "register") {
133        next();
134      } else {
135        router.push('login');
136      }
137    } else {
138      next();
139    }
140  });
```

Figure 7.2.2.2: Router Interceptor

At the end of router, an interceptor is declared. Going to all pages except for “login” and “register” requires a token. If token is not found, router will jump the website to the “login” page. The listener is shown in Figure 7.2.2.2.

## 7.2.3 Multi language

Vue-i18n is the package that we used for the multi language, this is a building backage in vue language, to use it we need to install vuei8n, by doing this package can be used as regular APIs in the vue file.

```
import { createI18n } from 'vue-i18n'
import zh from './zh'
import en from './en'

const message = {
  en,
  zh,
}
const language = (navigator.language || 'en').toLocaleLowerCase()
const i18n = createI18n( options: {
  locale: localStorage.getItem( key: 'lang') || language.split( separator: '-')[0] || 'en',
  legacy: false,
  globalInjection: true,
  messages: message,
})

export default i18n
```

Figure 7.2.3.1: Example Of Importing Multi Language And Combining Language Packages
In this project we made two languages: Chinese and English. All of the messages that will be showing to the users are collected in the src/locals/en.js and src/locals/en.js, then the English package and the Chinese are

combined together in src/locals/index.js. There is unique names for each message in the vue file for each page in C&C, so that we can switch the language by click the language changing button on the website.

```
/*
 * @FileDescription: Chinese language
 * @Author: Shizhong Shang
 * @Date: 2022/12/25 14:16
 * @LastEditor: Shizhong Shang
 * @LastEditTime: 2023/01/07 15:26
*/
export default{
    changeLang: "English",
    main: {
        menu: {
            item1: '发新状态',
            item2: '查看所有状态',
            item3: '查看所有好友',
            item3_hide: '继续隐藏好友',
            item4: '查看所有群组',
            item4_hide: '继续隐藏群组',
            item5: '编辑个人信息',
            item6: '退出'
        },
        bar: {
            item1: '朋友',
            item2: '群组',
            item3: '新鲜事',
            item4: '好友申请',
            item5: '群组邀请',
```

Figure 7.2.3.4: Example Of Chinese Package

```
/*
 * @FileDescription: English language
 * @Author: Shizhong Shang
 * @Date: 2022/12/25 14:15
 * @LastEditor: Shizhong Shang
 * @LastEditTime: 2023/01/07 16:13
*/
export default{
    changeLang: "中文",
    main: {
        menu: {
            item1: 'post my status',
            item2: 'check all my status',
            item3: 'show all friends',
            item3_hide: 'keep hide friends',
            item4: 'show all groups',
            item4_hide: 'keep hide groups',
            item5: 'edit my info',
            item6: 'log out'
        },
        bar: {
            item1: 'Friends',
            item2: 'Groups',
            item3: 'Find Something New',
            item4: 'Friend Requests',
            item5: 'Group Invitations',
```

Figure 7.2.3.4: Example Of English Package

*After we build the language.js file now the vue-i18n can be used a the API method by import it from the vue file, the it can be used though the $t method, the example is showing in Figure 7.2.3.5, on the main page, menu list, item2 can either be "Group" in English or "群组" in Chinese.*

```
import { useI18n } from "vue-i18n";
const { t } = useI18n();
```

Figure 7.2.3.4: Example Of Importing Multi Language In Vue File

```
<el-dropdown-item @click="router.push({ name: 'myStatus' })">
  <!-- myStatus -->
  {{ $t("main.menu.item2") }}</el-dropdown-item
>
```

Figure 7.2.3.5: Example Of Using Multi Language In Vue File

### 7.2.4 Components and Views

By following the proposal and the progress report that we wrote last semester, all the components are build successfully refer to our API document: API Specification Document Version 3, and contained in src/components, those components have their own UI and JS functions in the components.vue. The reason that we choose to do those components is because it will be used in a high frequent and it might be used in several different pages, then we can save time for coding. Figure 7.2.4.1 is the example of the result items, it is used to show the searching result on the webpage by import the ResultItem.vue in the view vue pages as a teg (Figure 7.2.4.2 and Figure 7.2.4.3), so the child component ResultItem is called by the parent vue pages.

```
<template>
  <div>
    <div class="all">
      <div id="left">
        <el-avatar :size="80" :src="props.avatar" alt="'https://cube.elemecdn.com/3/7c/3ea
        <span class="text">{{ props.username }}</span>
      </div>
      <div id="mid">
        <el-button round type="primary" size="large" @click="pressBtn">{{
          props.buttonLabel
        }}</el-button>
      </div>
      <div id="right">
        <el-icon @click="deleteItem" color="red" :size="30" class="icon"
          ><Close
        /></el-icon>
      </div>
    </div>
  </div>
</template>
```

Figure 7.2.4.1: ResultItem.vue Component

```
import ResultItem from "@/components/ResultItem.vue";
```

Figure 7.2.4.2: Example Of Import Component In View

```
<result-item
  :avatar="nf.avatar"
  :id="nf.id"
  :username="nf.uname"
  :button-label="t('newFriendList.add')"
  @delItem="close"
  @btnFunc="addBtn"
></result-item>
```

Figure 7.2.4.2: Example Of Using Component In View

### 7.2.5 Stores

The store.js is divided into different JS files, by doing that we can save time on calling functions from the stores, or if all the functions and variables are saved in one single store, the time cost of searching and using the functions will be increased. Stores are similar as apis but the reason we have a separate store package and not put it in the api package is because store is import from pinia, pinia is a store library for vue and it allows share the state across the pages. In this project, stores are need because the webpage should be changed whenever the user made a change without refresh the page, after using store, mutations is not needed.

```
import { defineStore } from "pinia";
import { getToken, setToken, removeToken } from "@/request/token";
import { signin, signout, register, showUserInfo, showAvatarUname, changeUserInfo} from "@/api/user.js";
import { ElMessage } from 'element-plus'
import i18n from '../locals/index.js'

const useUserStore = defineStore("user", {
  state: () => {
    return {
      token: "",
      name: "",
      avatar: "", // head image
      email: "",
      tel: "",    // user phone number
      address: "",
      birthday: "",
      // group info
      groupId: "",     // the latest access group id
      notice: "",      // the latest access group notice
      groupName: "",   // the latest access group name
      groupAvatar: "", // the latest access group head image
    };
  },
  getters :{
    getGroupId: (state) => {
```

Figure 7.2.5.1: UserStore.js File

```
import { ref, reactive, watch, onBeforeUnmount, onMounted,  } from "vue";
import  useUserStore  from "@/stores/userStore";
import { storeToRefs } from "pinia";
const userStore = useUserStore();
const { token } = storeToRefs(userStore);
```

Figure 7.2.4.2: Example Of Using Store In View

## 7.3 WebSocket

C&C is an IM software and WebSocket is the main technology used to implement IM. Therefore, the implementation part of WebSocket will be covered in detail here.

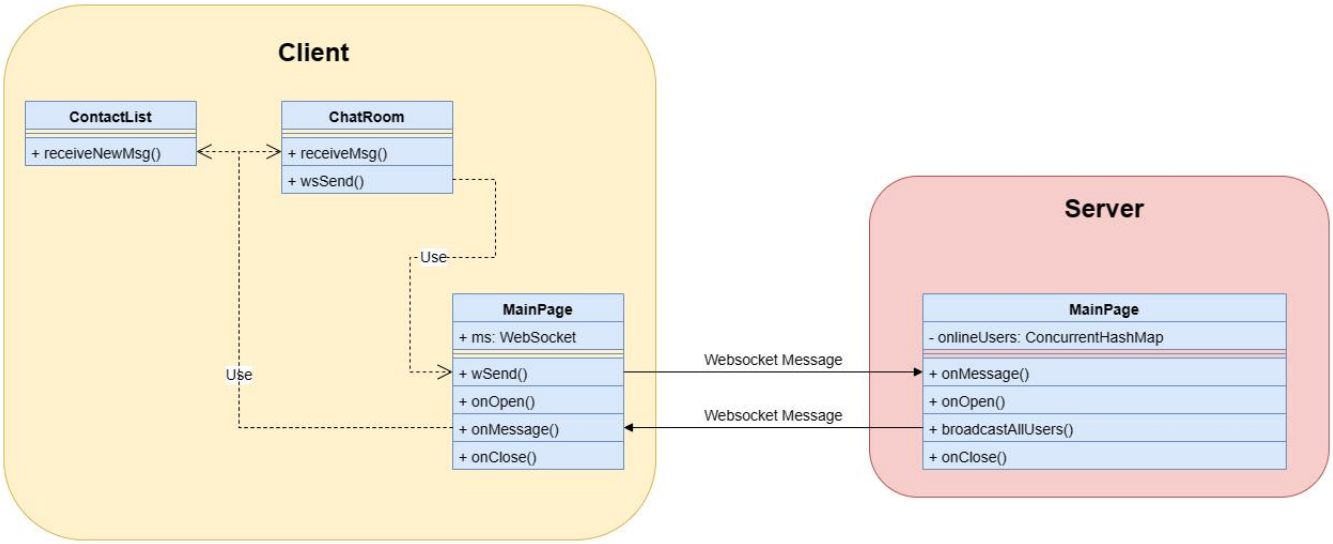



Figure 7.3.0: WebSocket Working Progress

As shown in Figure 7.3.0, in the front end, only ContactList, ChatRoom and MainPage uses WebSocket. ContactList will notice new message if exists, ChatRoom will show the message on the window if the message is sent by the current chatroom. MainPage is responsible for creating, receiving and sending WebSocket messages to the backend. On the server side, all online users are stored in a ConcurrentHashMap — onlineUsers, the onMessage function receive all message sent by clients and broadcastAllUsers function send the received message to all target who are currently online.

In the front end, the WebSocket connection is created during the mounting process of the MainPage, which will be reached directly after user login.

```
301  onMounted(() => {
302    if (ws == "") {
303      ws = new WebSocket("ws://" + url + "/chat");
304  >   ws.onopen = function () {···
306      };
307  >   ws.onclose = function () {···
309      };
310  >   ws.onmessage = function (evt) {···
344      };
345      console.log("ws set up!!");
346      console.log(ws);
347    }
```

Figure 7.3.1: Front End WebSocket Instance

The WebSocket connection is created on line 303 in Figure 7.3.1. The onopen and onclose function shown in Figure 7.3.1 simply print the event in console.

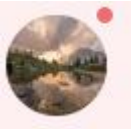


Figure 7.3.2: Contact List New Message Notice

```
310     ws.onmessage = function (evt) {
311       //alert("onMessage");
312       var dataStr = evt.data;
313
314       var res = JSON.parse(dataStr);
315       let rType = res.receiverType;
316       let rid = res.receiver;
317       let sid = res.sender;
318
319       if (rType == "friend") {
320         fnoticeNew.value.noticeNewMsg(true, sid, true);
321       } else {
322         gnoticeNew.value.noticeNewMsg(false, rid, true);
323       }
324
325       if (router.currentRoute.value.name == "chatRoom") {
326         let str = router.currentRoute.value.params.id;
327         let type = str[0];
328         let roomId = str.slice(1);
329
330         if (type == "f") {
331           if (sid == roomId && rType == "friend") {
332             fnoticeNew.value.noticeNewMsg(true, sid, false);
333             //store.setNewMsg(res);
334             order.value.receiveMsg(res);
335           }
336         } else {
337           if (rid == roomId && rType == "group") {
338             gnoticeNew.value.noticeNewMsg(false, rid, false);
339             //store.setNewMsg(res);
340             order.value.receiveMsg(res);
341           }
342         }
343       }
344     };
```

Figure 7.3.3: onmessage Function

In Figure 7.3.3, from line 310 to 317, the function receives the String message from the server and get the message object: res, receiverType(friend or group), receiver id and sender id by deserialization. From line 319 to 323, the function triggers the message notice to the corresponding contact list which is shown in Figure 7.3.2. On line 325, the if statement checks if the user is currently in a chatroom, if the statement is true, the function get room type(friend or group) on line 327 and room id on line 328. If the current room is friend chatroom and the room id matches the sender id of the received message, the value will be shown on the page on line 334 and

the new message notice will be removed from the friends contact list (line 332). If the current room is a group chatroom and the room id matches the sender id of the received message, the function will do the same. Remove the new message notice from group contact list (line 338) and show the message on the page (line 340).

```
292 function wSend(input) {
293   //console.log(input);
294   ws.send(JSON.stringify(input));
295 }
```

Figure 7.3.4: wSend Function

Front end use the wSend function shown in Figure 7.3.4 to send websocket message to the server.

```
23  @Configuration
24  @Slf4j
25  public class WebSocketConfig extends ServerEndpointConfig.Configurator{
26
        3 usages  zirui *
27      @Override
28      public void modifyHandshake(ServerEndpointConfig sec,
29                                  HandshakeRequest request,
30                                  HandshakeResponse response) {
31          HttpSession httpSession = (HttpSession) request.getHttpSession();
32          if (httpSession != null) {
33              sec.getUserProperties().put(HttpSession.class.getName(), httpSession);
34          }
35          super.modifyHandshake(sec, request, response);
36      }
        1 usage  zirui
37      @Bean
38      public ServerEndpointExporter serverEndpointExporter() {
39          //log.info("initial Websocket config!!!!!!!!!!!!!!!!");
40          return new ServerEndpointExporter();
41      }
42  }
```

Figure 7.3.5: WebSocket Configuration Class

Figure 7.3.5 shows the backend configuration for Websocket (“src/main/java/com/nart/config/webSocketConfig.java”). Since a Websocket will be connected every time a user logs into the system, on line 31 the modifyHandshake function stores the user’s HttpSession in the EndpointConfig for easy access later.

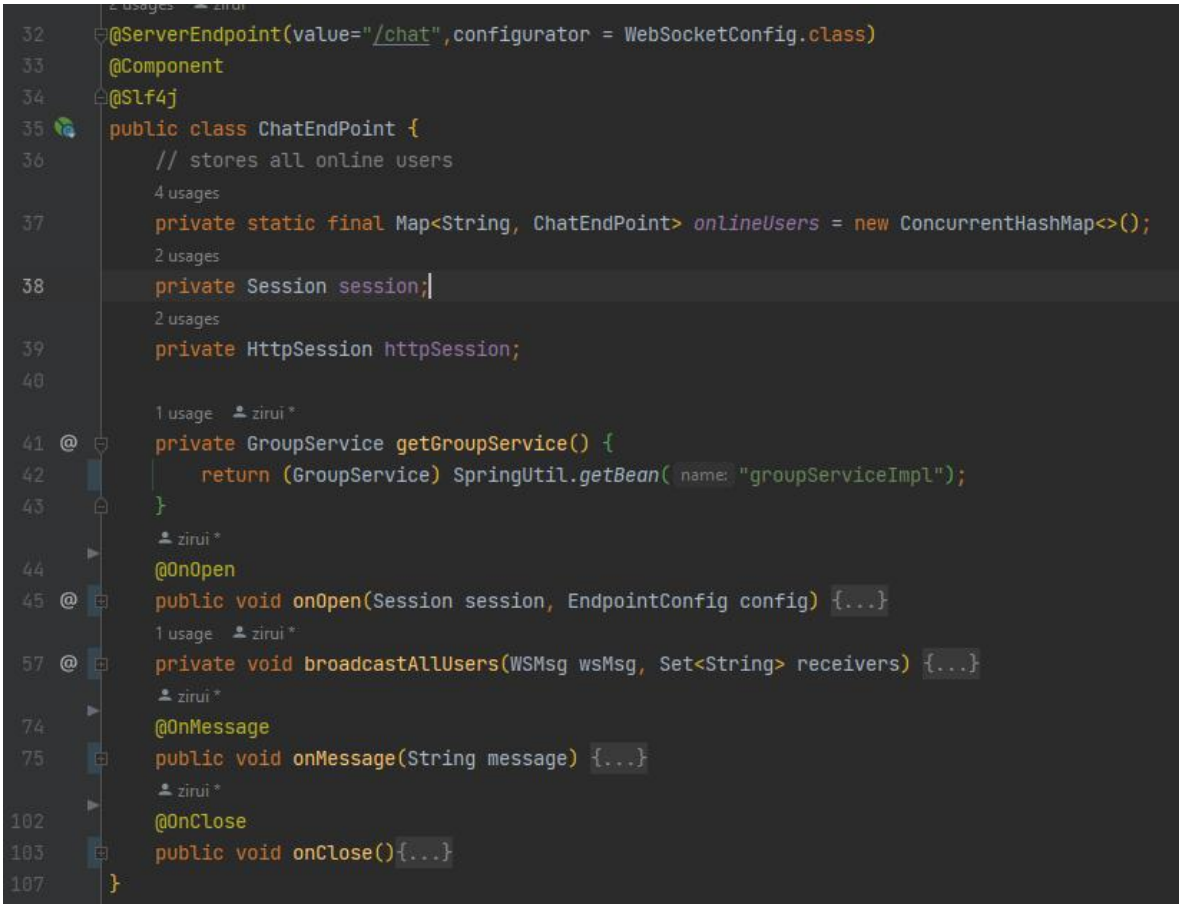


Figure 7.3.6: Websocket Handler

The WebSocket handler shown in Figure 7.3.6 is declared in file "src/main/java/com/nart/common/ChatEndPoint.java".The handler only handles three user actions by corresponding functions: onOpen for create websocket connection, onClose for disconnect websocket and onMessage for receive websocket messages.

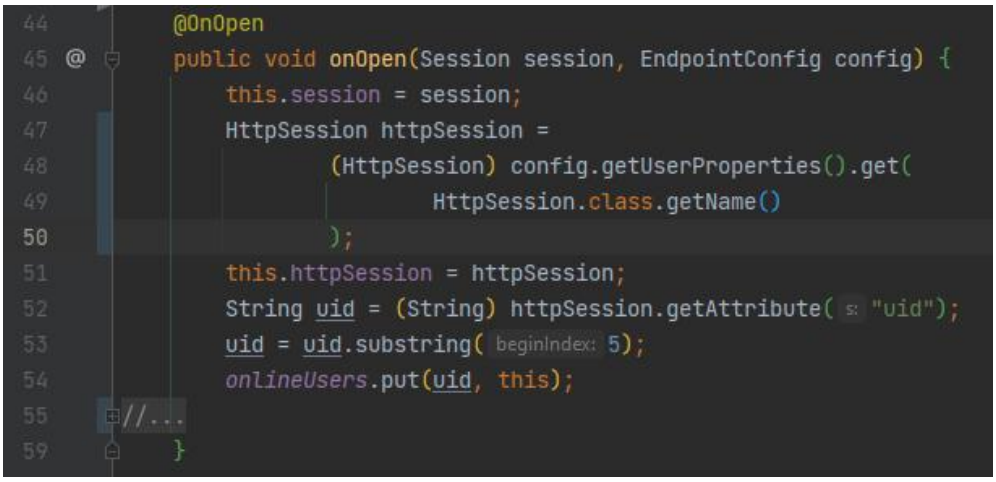


Figure 7.3.7: onOpen Function

The onOpen function shown in Figure 7.3.7 stores the current user's session and previously set HttpSession as private variables of ChatEndPoint Object. The user's id (Set when Login) is retrieved from the HttpSession,

then the (user id, ChatEndPoint) key-value pair is put into onlineUsers (a static Concurrent Hash Map declared in Figure 7.3.6 line 37) for future use.

```
105     @OnClose
106     public void onClose(){
107         String uid = (String) httpSession.getAttribute( name: "uid");
108         onlineUsers.remove(uid);
109     }
```

Figure 7.3.8: onClose Function

The onClose function shown in Figure 7.3.8 remove the current user from the static Concurrent Hash Map.

```
77      @OnMessage
78      public void onMessage(String message) {
79          try {
80              ObjectMapper mapper = new ObjectMapper();
81              WSMsg msg = mapper.readValue(message, WSMsg.class);
82              String sid = msg.getSender();
83              Map<String, Object> stringObjectMap = EncryptUtil.checkToken(sid);
84              if (stringObjectMap != null) {
85                  sid = ((Long) stringObjectMap.get("userId")).toString();
86              }
87              msg.setSender(sid);
88
89              Set<String> receivers = new HashSet<>();
90              String receiverType = msg.getReceiverType();
91              String receiver = msg.getReceiver();
92              // set receivers
93              if(receiverType.equals("friend")) {
94                  receivers.add(receiver);
95              } else {
96                  GroupService gs = getGroupService();
97                  receivers.addAll(gs.findAllMembers(receiver));
98              }
99
100             broadcastAllUsers(msg, receivers);
101         } catch (Exception e) {
102             e.printStackTrace();
103         }
104     }
```

Figure 7.3.9: onMessage Function

The onMessage function shown in Figure 7.3.9 receives a JSON format String message. A JSON deserializer(ObjectMapper) is used to deserialize the message to a WSMsg containing message, sender, and receiver information. The sender attribute in WSMsg is the token of the sender; the EncryptUtil.checkToken function is used to deserialize the token to get the sender's id and reset it to the WSMsg object. Different receivers are added to a set called receivers according to the receiverType, which has only two values: "friend" and "group". Finally, the receivers set and the WSMsg object is passed to broadcastAllUsers function which will send the message to all receivers who are online and in the set.

# 8. Validation and Verification

## 8.1 Unit Testing

With Java unit tests, writing a generic unit test for variable test data isn't easy. This is particularly the case with service packages, where once the data in the database has changed, there is a high risk that the tests that initially passed will no longer pass. To solve this problem, and in conjunction with what was taught in SYSC4101, we decided to use stubbing and mocking for our tests so that we test the function we are currently testing separately from the function it calls. This way, the test case still passes even if the database changes or the method it calls changes.

Junit5 and Mockito are used for stubbing and mocking. SInce the project structure follows a maven project structure, the tests are in package “src/test” which is shown in Figure 8.1.1

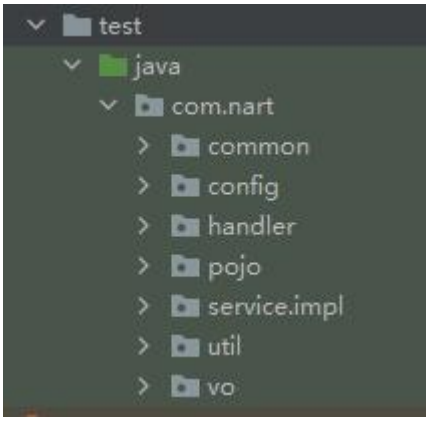


Figure 8.1.1: Test Package

## 8.2 API Testing

The purpose of API testing is to verify whether the front-end can get the correct data from the back-end. We use a postman for testing. Postman is an interface testing tool. When doing interface testing, Postman is equivalent to a client. It can simulate various HTTP requests initiated by users, send the request data to the server, and obtain the corresponding response results, so as to verify whether the result data in the response matches the expected value. According to the settings in the back-end controller, http requests are divided into four types: get, put, post and delete. In the test, different request types are used according to the content of the request. For example, if the client needs to obtain data, it uses the get type. If the client needs to create new data, it uses the post type. If the client needs to modify existing data, it uses the put type. If the client needs to delete data, it uses the delete type.

The API test design document is located in NART/doc/design.
API Specification Document Version 3

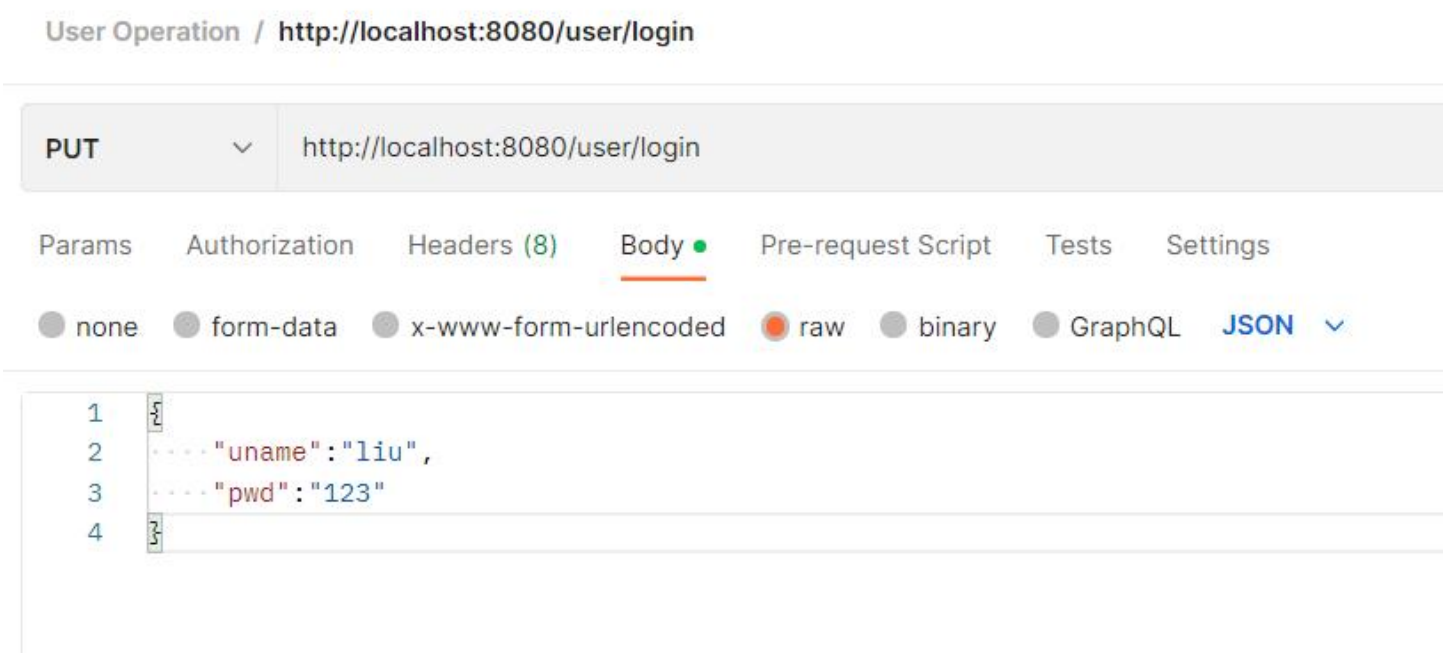



Figure 8.2.1: Login  API Test Example client (Postman)

```
 * @Date 2023/01/12 13:25
 */
@RestController
@RequestMapping("/user")
public class UserController {
```

Figure 8.2.2: Login API Test Example server

```
@LogA
@PutMapping("login")
public Result login(@RequestBody UserVo uInfo, HttpSession session) {
    System.out.println(uInfo);
    return loginService.login(uInfo.getUname(), uInfo.getPwd(), session);
}
```

Figure 8.2.3: Login API Test Example server

Figure 8.2.1 shows the test of client login operation. The web address it uses is divided into four parts. First, localhost indicates that the server uses the local machine. 8080 represents the port used by the server. User corresponds to the mapping of userController in controller in Figure 8.2.2. When user is used in the web address, its request will be converted to the userController layer by the server. The annotation @ PutMapping ("login")

above the login method in Figure 8.2.3 shows two messages. PutMapping indicates that the put type is required when the method is requested by the client, just like the put in the upper left corner of Figure 8.2.1. Login indicates that the last part of the URL in the request needs to be login The parameter part of the login method in Figure 8.2.3 has an annotation RequestBody. This annotation indicates that the type of parameter in the client request is body. UserVo means that the incoming information will be stored in the format of UserVo in the Vo package. In Figure 8.2.1, the parameter type is body. According to the requirements in the login method, the user name and password are added to the body.

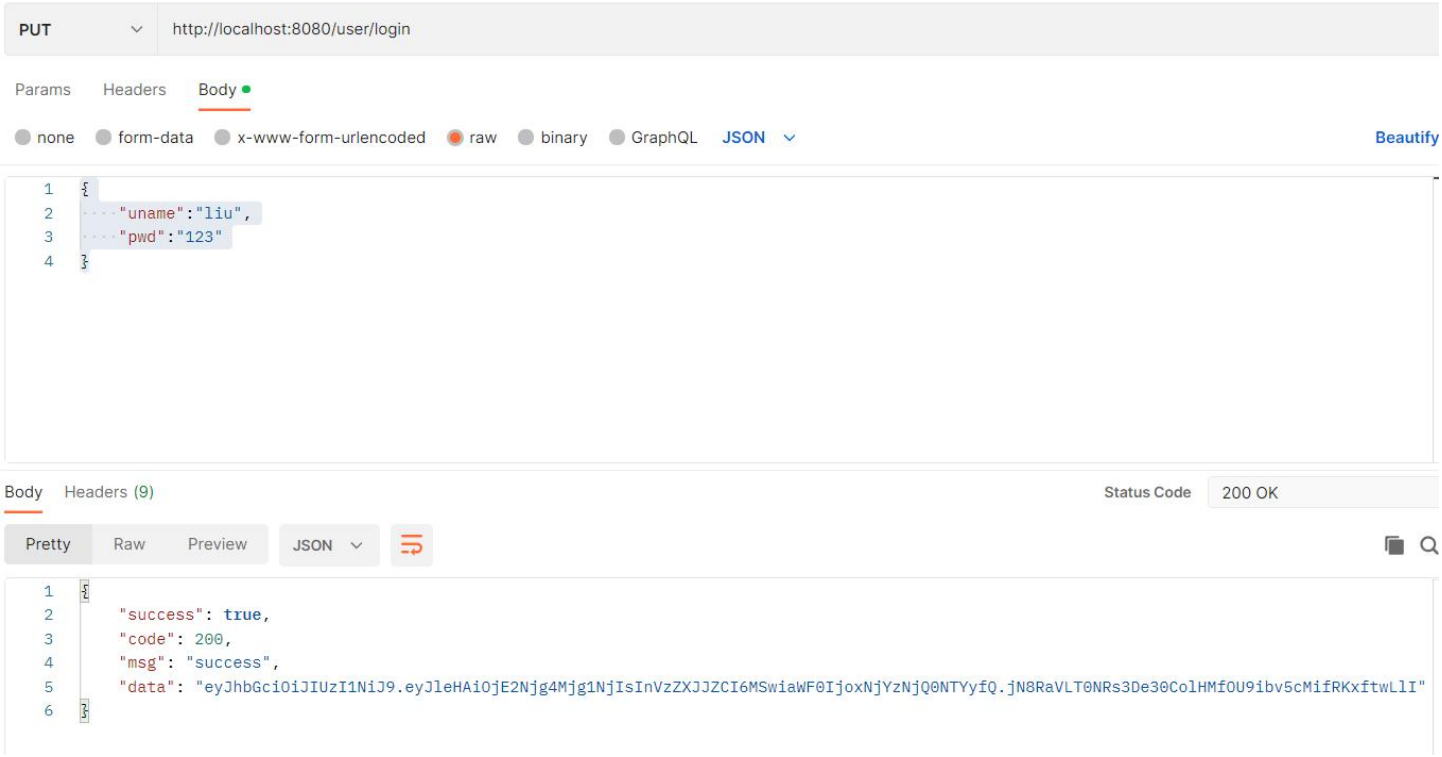



Figure 8.2.4: Login API Test return Example client (Postman)

The lower part of Figure 8.2.4 shows that the postman's request has received a successful response from the server. The format of the response is the Result format, which is defined by the Result class in the util package in the back-end file. The response contains four messages. The first message “success” indicates whether the operation was successful. The second message “code” indicates the status of the operation. If it succeeds, it will return 200. If it fails, it will return different codes according to the type of failure. The third message “msg” represents the client's description of this operation. The fourth information “data” contains the token of the associated user, which will be used in other operations.

## 8.3 Integration Testing

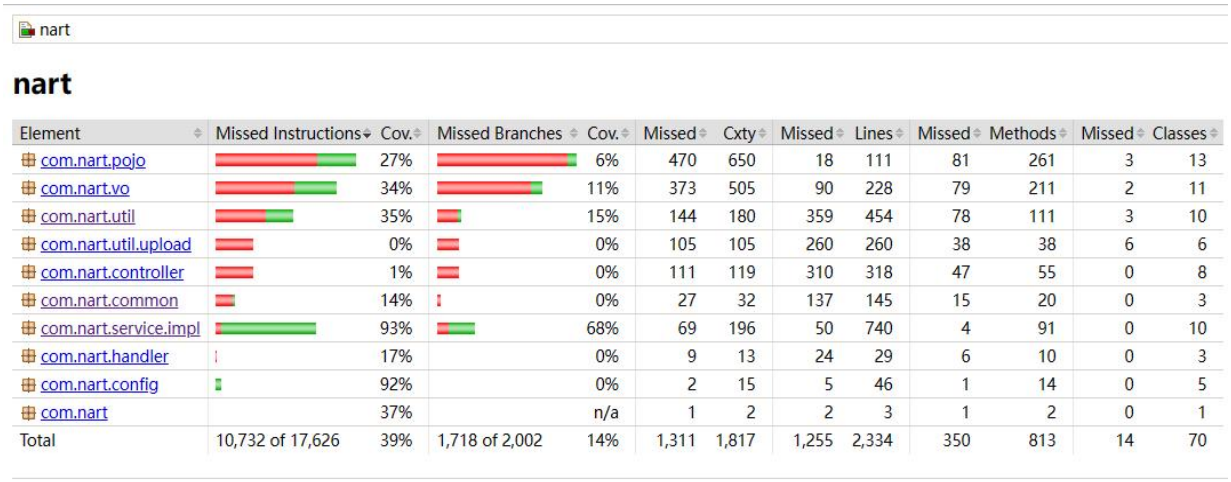
Not Yet

# 9. Deployment and Performance Testing

Not Yet

# 10. Documentation

The code coverage of C&C unit tests is counted using jacoco and its reports can be accessed on through the following link: jacoco report. A preview of Jacoco coverage report is shown in Figure 10.1.



nart

**nart**

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| com.nart.pojo | | 27% | | 6% | 470 | 650 | 18 | 111 | 81 | 261 | 3 | 13 |
| com.nart.vo | | 34% | | 11% | 373 | 505 | 90 | 228 | 79 | 211 | 2 | 11 |
| com.nart.util | | 35% | | 15% | 144 | 180 | 359 | 454 | 78 | 111 | 3 | 10 |
| com.nart.util.upload | | 0% | | 0% | 105 | 105 | 260 | 260 | 38 | 38 | 6 | 6 |
| com.nart.controller | | 1% | | 0% | 111 | 119 | 310 | 318 | 47 | 55 | 0 | 8 |
| com.nart.common | | 14% | | 0% | 27 | 32 | 137 | 145 | 15 | 20 | 0 | 3 |
| com.nart.service.impl | | 93% | | 68% | 69 | 196 | 50 | 740 | 4 | 91 | 0 | 10 |
| com.nart.handler | | 17% | | 0% | 9 | 13 | 24 | 29 | 6 | 10 | 0 | 3 |
| com.nart.config | | 92% | | 0% | 2 | 15 | 5 | 46 | 1 | 14 | 0 | 5 |
| com.nart | | 37% | | n/a | 1 | 2 | 2 | 3 | 1 | 2 | 0 | 1 |
| Total | 10,732 of 17,626 | 39% | 1,718 of 2,002 | 14% | 1,311 | 1,817 | 1,255 | 2,334 | 350 | 813 | 14 | 70 |

Figure 10.1: Jacoco Report

The Java Api Document is automatically generated by javadoc plugin, and the document can be accessed through the following link: javadoc. A preview of the Java Doc is shown in Figure 10.2.

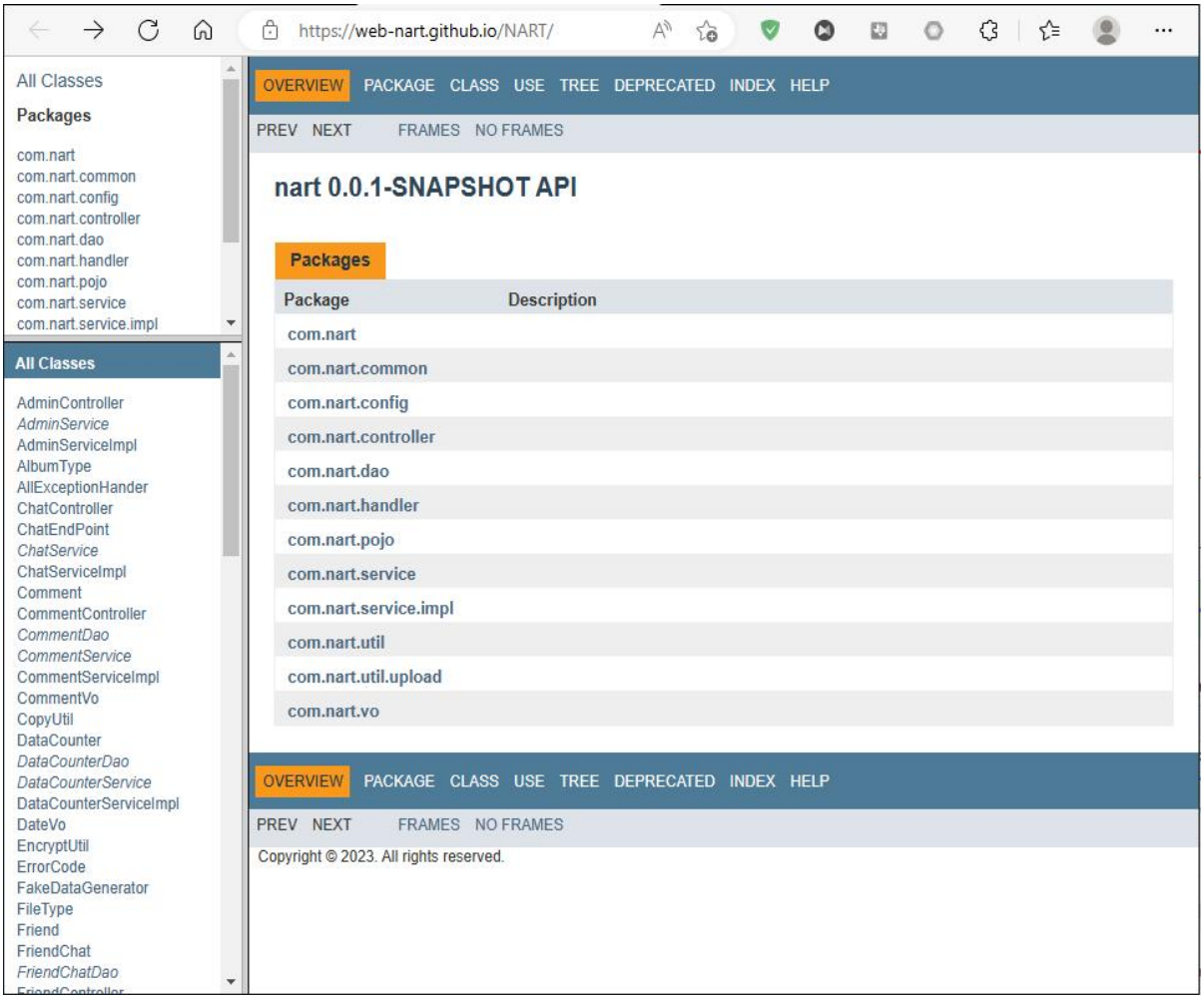


Figure 10.2: Java Doc Preview

The designing graphs including front end designing and UMLs are also stored and can be accessed through the following link: docs.

# 11. Discussion and analysis

Although we tried our best in the investigation, design and implementation, many parts can still be improved.

Due to a lack of investigation, we choose Mybatis-Plus to do the ORM instead of Spring Data. Although its usability and performance are fine, it still causes many problems during testing since Mockito and Mybatis-Plus do not work well together.

Another problem is that the performance of C&C is far away from what we expected. The main reason is that we only use Redis for users' tokens, IDs and names. We retrieve the rest data from MYSQL directly, which wastes a lot of time. If we have more time, data usage statics will be done on C&C to find the most often accessed data; then, we can also store those data in Redis for quick access. Performance is guaranteed to be improved in this case.

The front-end appearance can also be improved. Although we do not have a professional designer in our group, we can make a better-looking page with more time.

The security of C&C is a big problem. Currently, C&C is not a software that provides a service to the public, and the lack of security is acceptable for C&C. However if further development is desired, improving security will be an essential and very laborious part of the process. Firstly, Admin-related services must be separated from client services so clients cannot access Admin API. Then, all chat messages and user information like id, phone number and email should be encrypted. A verification code could be needed when a user is trying to log in for a double identification check. The server's firewall should also be set well to avoid some attacks.

Finally, many planned features are not implemented—for example, the Optional Function: Faction Grouping and the distributed system. Many analyses and statics still need to be done, for example, the previously mentioned data usage statistics. With more time, C&C will become a better software.

# 12. Conclusions

C&C is almost done by following the timeline provided in the proposal, and almost all of the functionalities are achieved. During the time doing this project, our team learnt many new techniques and got more familiar with building a project by learning, using and comparing those techniques. C&C is a huge difficulty when starting to do it, but with time passing by, C&C helped us improve ourselves a lot, and it is a good chance to challenge our team and then know what our weaknesses are and practice how to make a good team. Even though C&C is not perfect, we gain a lot from this, and it will make a good influence on us.

# References


[1]“Discord Developer Portal — API Docs for Bots and Developers,” *Discord Developer Portal*, 2022. https://discord.com/developers/docs/intro (accessed Oct. 11, 2022).

[2]“MySQL :: MySQL 8.0 Reference Manual :: 3 Tutorial,” *Mysql.com*, 2022. https://dev.mysql.com/doc/refman/8.0/en/tutorial.html (accessed Oct. 11, 2022).

[3] “WhatsApp Features,” *WhatsApp.com*, 2022. https://www.whatsapp.com/features (accessed Oct. 11, 2022).

[4]“WhatsApp Business Platform - Documentation - Facebook for Developers,” *Facebook.com*, 2022. https://developers.facebook.com/docs/whatsapp/ (accessed Oct. 11, 2022).

[5]“Authentication - Marketing API - Documentation - Facebook for Developers,” *Facebook.com*, 2022. https://developers.facebook.com/docs/marketing-apis/overview/authentication (accessed Oct. 11, 2022).

[6]“What is a REST API?,” *Redhat.com*, 2022. https://www.redhat.com/en/topics/api/what-is-a-rest-api (accessed Oct. 11, 2022).

[7]“Overview - WhatsApp Business Platform On-Premises API - Documentation - Facebook for Developers,” *Facebook.com*, 2022. https://developers.facebook.com/docs/whatsapp/on-premises/overview (accessed Oct. 11, 2022).

[8]“Client Architecture - WhatsApp Business Platform On-Premises API - Documentation - Facebook for Developers,” *Facebook.com*, 2022. https://developers.facebook.com/docs/whatsapp/on-premises/overview/client-architecture (accessed Oct. 11, 2022).

[9]“Security - WhatsApp Business Platform On-Premises API - Documentation - Facebook for Developers,” *Facebook.com*, 2022. https://developers.facebook.com/docs/whatsapp/on-premises/overview/security (accessed Oct. 11, 2022).

[10]“Reference - WhatsApp Business Platform On-Premises API - Documentation - Facebook for Developers,” *Facebook.com*, 2022. https://developers.facebook.com/docs/whatsapp/on-premises/reference (accessed Oct. 11, 2022).

[11]WEB-NART, “NART/whatsapp.md at main · WEB-NART/NART,” *GitHub*, Oct. 09, 2022. https://github.com/WEB-NART/NART/blob/main/doc/whatsapp.md (accessed Oct. 11, 2022).

[12]“Messenger – Apps on Google Play,” *Google*. [Online]. Available: https://play.google.com/store/apps/details?id=com.facebook.orca&hl=en_CA&gl=US. [Accessed: 11-Oct-2022].

[13]E. Moreau, “Facebook Messenger: Everything you need to know,” *Lifewire*, 30-Dec-2021. [Online]. Available: https://www.lifewire.com/facebook-messenger-4103719. [Accessed: 11-Oct-2022].

[14]“Messenger,” *Facebook*. [Online]. Available: https://www.messenger.com/features?locale=en_GB. [Accessed: 11-Oct-2022].

[15]“HTML introduction,” *Introduction to HTML*. [Online]. Available: https://www.w3schools.com/html/html_intro.asp. [Accessed: 19-Oct-2022].

[16]“What is CSS? - learn web development: MDN,” *Learn web development | MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS. [Accessed: 19-Oct-2022].

[17]“What is javascript? - learn web development: MDN,” *Learn web development | MDN*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Accessed: 19-Oct-2022].

[18]“Introduction | Vue.js,” *Vuejs.org*, 2022. https://vuejs.org/guide/introduction.html (accessed Oct. 18, 2022).

[19]*Java.com*, 2022. https://www.java.com/en/download/help/whatis_java.html (accessed Oct. 18, 2022).

[20]“MySQL,” *Mysql.com*, 2022. https://www.mysql.com/ (accessed Oct. 18, 2022).

[21]Redis, “Redis,” *Redis*, 2022. https://redis.io/ (accessed Oct. 18, 2022).

[22] “Where the world builds software,” *GitHub*. [Online]. Available: https://github.com/. [Accessed: 18-Oct-2022].

[23]Web-Nart, “Web-Nart/NART,” *GitHub*. [Online]. Available: https://github.com/WEB-NART/NART. [Accessed: 19-Oct-2022].

[24]Oloruntoba, S. (2021, November 30). Solid: The first 5 principles of object oriented design. DigitalOcean. Retrieved November 26, 2022, from https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

[25]Learn java. Dev.java: The Destination for Java Developers. (n.d.). Retrieved November 24, 2022, from https://dev.java/learn/

[26]Learn about spring. Spring. (n.d.). Retrieved November 24, 2022, from https://spring.io/learn

[27]Quickly develop cloud applications with mysql heatwave. Oracle Canada. (n.d.). Retrieved November 24, 2022, from https://www.oracle.com/ca-en/mysql/

[28]"MongoDB Vs. Redis Comparison: Pros And Cons," MongoDB, 2022. https://www.mongodb.com/compare/mongodb-vs-redis (accessed Nov. 19, 2022).

[29]"Redis vs MongoDB: Which is Better? MongoDB & Redis 101 | Redis," Redis, Sep. 19, 2022. https://redis.com/docs/why-your-mongodb-needs-redis/ (accessed Nov. 19, 2022).

[30]"Vue.js - The Progressive JavaScript Framework | Vue.js," *Vuejs.org*, 2014. https://vuejs.org/ (accessed Nov. 26, 2022).

[31]"A Vue 3 UI Framework | Element Plus," *Element-plus.org*, 2022. https://element-plus.org/en-US/#hybrid-layout (accessed Nov. 26, 2022).

[32]"Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.," *Tailwindcss.com*, Nov. 15, 2020. https://tailwindcss.com/ (accessed Nov. 26, 2022).

[33]"HTTP | MDN," *Mozilla.org*, Sep. 13, 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP (accessed Nov. 19, 2022).

[34]"JSON," *Json.org*, 2022. https://www.json.org/json-en.html (accessed Nov. 19, 2022).

[35]"The WebSocket API (WebSockets) - Web APIs | MDN," *Mozilla.org*, Sep. 09, 2022. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (accessed Nov. 19, 2022).

[36]“WebRTC API - Web APIs | MDN,” *Mozilla.org*, Oct. 30, 2022. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API (accessed Nov. 19, 2022).

[37]L. Rosencrance, G. Lawton, and C. Moozakis, “User Datagram Protocol (UDP),” *SearchNetworking*, 2021. https://www.techtarget.com/searchnetworking/definition/UDP-User-Datagram-Protocol#:~:text=User%20Datagram%20Protocol%20(UDP)%20is,provided%20by%20the%20receiving%20party. (accessed Nov. 19, 2022).

[38]B. Lutkevich, “Transmission Control Protocol (TCP),” *SearchNetworking*, 2021. https://www.techtarget.com/searchnetworking/definition/TCP#:~:text=Transmission%20Control%20Protocol%20(TCP)%20is,of%20data%20to%20each%20other. (accessed Nov. 19, 2022).

[39]J. Juviler, “REST APIs: How They Work and What You Need to Know,” *Hubspot.com*, Aug. 30, 2022. https://blog.hubspot.com/website/what-is-rest-api (accessed Nov. 19, 2022).

[40]“SOAP vs REST APIs: Which Is Right For You? | SoapUI,” *Soapui.org*, 2014. https://www.soapui.org/learn/api/soap-vs-rest-api/ (accessed Nov. 19, 2022).

[41]J. Juviler, “4 Types of APIs All Marketers Should Know,” *Hubspot.com*, Aug. 26, 2021. https://blog.hubspot.com/website/types-of-apis#:~:text=There%20are%20four%20widely%20agreed,internal%20APIs%2C%20and%20composite%20APIs. (accessed Nov. 19, 2022).

[42]“Figma,” *Figma*, 2022. https://www.figma.com/file/0CA3CWW1COcWTyDnwkJmKk/Untitled?node-id=0%3A1&t=IRd0f2at4CaO30X1-0 (accessed Dec. 07, 2022).

# Appendix 1. Definition, Acronyms, and Abbreviations

C&C --- The Come And Chat Instant Messaging Software System.

Message --- Information in text, image, audio, video, link, or hybrid.

Friend --- A relationship between users that gives both parties a quick link to each other.

Status --- A message posted by a user that the user's friends can only see.

Chat --- A connection tube between two users.

Group Chat --- A connection tube among three or more users.

The system --- represents the C&C Instant Messaging Software system.

IM --- represents Instant Messaging.

API --- application programming interface, which is a software intermediary that allows two applications to talk to each other.

HTTP --- The Hypertext Transfer Protocol is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems.

DDoS --- A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

Guild --- A server that can accommodate multiple users to communicate at the same time

Instance --- A format used by the back end to contain the data transmitted by the front end. Usually, instances can also be added, deleted, modified, and searched for database records.

ORM — Object Relational Mapping

SYSC 2004 --- Object-Oriented Software Development (BlueJ Java)

SYSC 3020 --- Intro. to Software Engineering

SYSC 3110 --- Software Development Project

SYSC 3303 --- Real-Time Concurrent Systems

SYSC 3310 --- Introduction to Real-time Systems

SYSC 2100 --- Algorithm and Data Structures

COMP 3005 --- Database Management Systems

SYSC 3120 --- Software Requirements Engineering

SYSC 4106 --- The Software Economy and Project Management

SYSC 4101 --- Software Validation

SYSC 4801 --- Intro. to Network and Software Security

ECOR 2050 --- Design & Analysis of Engineering Experiments

ECOR 4995 --- Professional Practice

CCDP 2100 --- Communication Skills for Engineering Students

SYSC 4806 --- Software Engineering Lab

SYSC 4120 --- Software Architecture & Design

# Appendix 2. Source Documents

Use Case Diagram Version 1

Use Case Diagram Version 2

Use Case Diagram Version 3

Use Case Diagram Version 4

Use Case Description Version 1

Use Case Description Version 2

User FlowDiagram Version 1

User FlowDiagram Version 2

User FlowDiagram Version 3

Admin FlowDiagram Version 1

Module UML

ERDiagram Version 1

ERDiagram Version 2

ERDiagram Version 3

ERDiagram Version 4

User Front Page Prototype

User Front Page Prototype Detail

User Front Page Structure

User Front-End UML

User Back-End UML Version 1

User Back-End UML Version 2

Admin Front Page Prototype

Admin Front-End UML

API Specification Document Version 1

API Specification Document Version 2

API Specification Document Version 3

Overall C&C Structure

C&C Source Code

# Appendix 3. Proposal

Team6_Proposal.pdf

# Appendix 4. Proposal

Progress Report