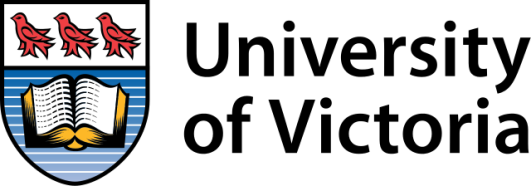
****

**University of Victoria**

**Faculty of Engineering**

**Summer 2017 SENG299 Milestone4**

**Technical Report:**

**Final Report**

Team members:

Zelan Xiang, V00825496, zelan

Haotian Shen, V00817044, hshen

Pengxiang Jia, V00835664, jiapx

# **Table of Contents**

[Table of Contents 1](#_Toc489293024)

[Purpose 2](#_Toc489293025)

[Overview of Design 2](#_Toc489293026)

[Description of Implementation 2](#_Toc489293027)

[Indication of Requirements 16](#_Toc489293036)

[Design Process and Timeline 18](#_Toc489293037)

[Problem Encountered 20](#_Toc489293038)

[Contributions and Contributors 21](#_Toc489293039)

[Acknowledgements 25](#_Toc489293040)

Purpose

This technical report aims to state the work conducted throughout the project， clarify the features of the chat system (Chatus), alone with the changes and difficulties we met during the developing process. This report is based on the milestone 1, milestone 2, milestone 3 from another group, and the implementation of the project.

It contains a high-level overview of the system design, a description of implementation, an indication of requirements table, a timeline of project, the problems that we have met, and the contributions of each group member.

Overview of Design

The overall system design has not been changed since milestone1. There are three classes included in the system: client, server, and Chatroom. Client will collect what user typing, then, check the actions (e.g. regular sending message, /create, /delete, /join, /block, /unblock, or /set\_alias), and merge the action-message to a string. Finally, send it to server. Server listens for new connection and check for incoming data. Once upon data received, the server parses the data and behaves as the client requested. Chatroom is a class to create an object called Chatroom and store all chatroom information.

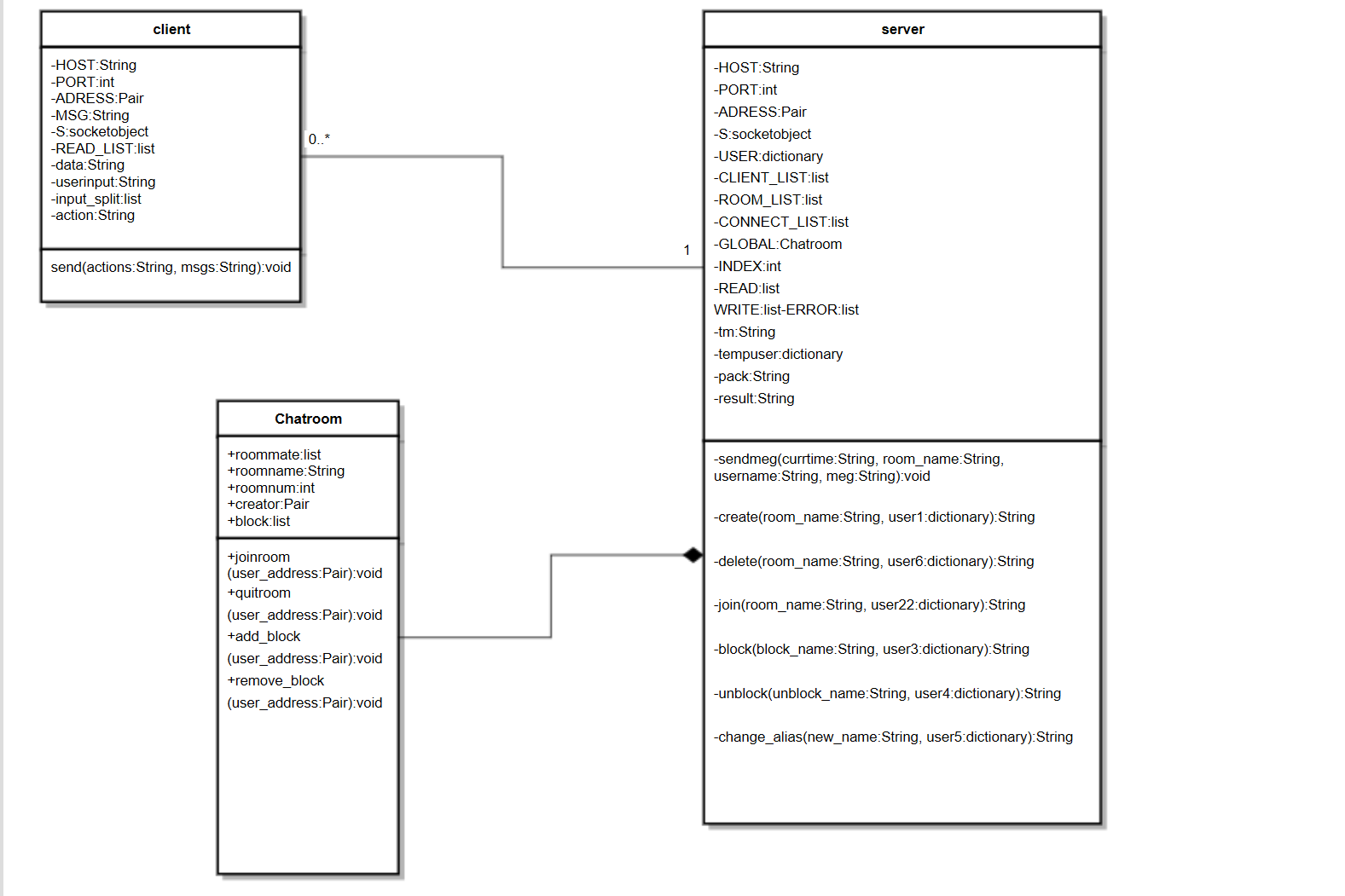
After taking the feedback from milestone 3, we change the parameters data type, advanced user instruction format and server behavior slightly (now kicks blocked user from chatroom).

Description of Implementation

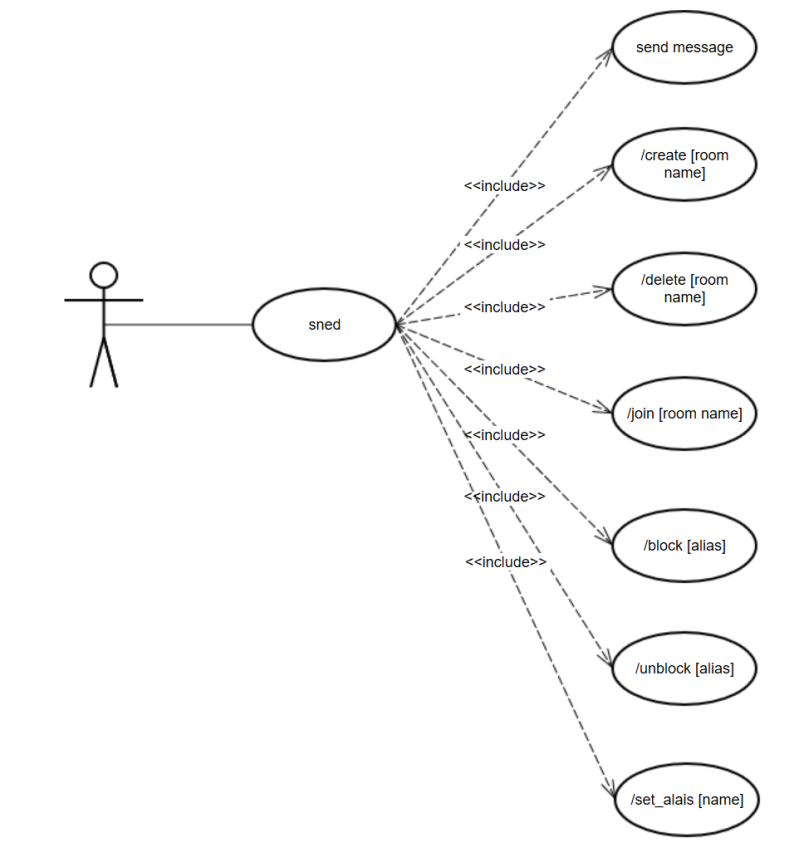
The implementation of our system is based on the design from milestone 2 and the improvement after milestone 3. The implementation satisfies almost all requirements in milestone 1 and 2. GRASP

The design made use of indirection (GRASP), singleton pattern and command design pattern (GoF). The system has Chatroom class which is a indirection design pattern; the system has only one instantiation of server class so it is classified Singleton; the system also has fully employed the Command design pattern.

Client is operated on command interface. It produces commands to invoke action but it doesn’t know the implementations of such action. Commands have the delegated method name to be invoked and store related parameters for that method, although the invoker even doesn’t know how the information composed in command corresponding to receiver’s method and parameters. The server is an encapsulation of all information needed to perform such organizational actions such as redirecting text message, rearranging chatrooms and members within, which subcategory as including or excluding a user to a group chat, and deleting or creating a chatroom.

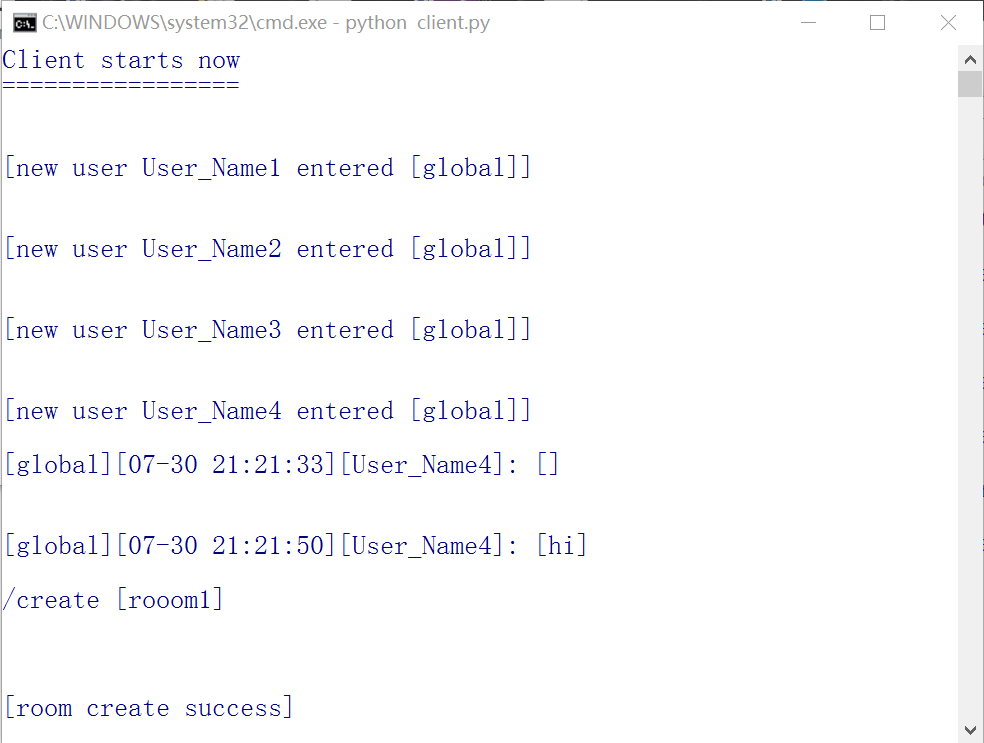
Class Diagram

*Figure 1: class diagram*

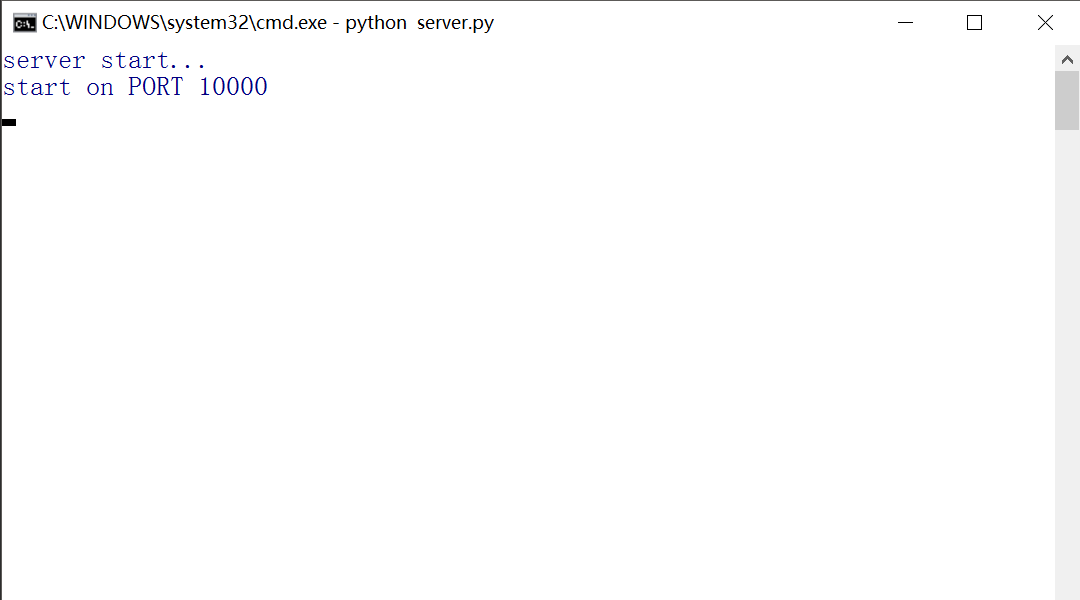
Use Cases Diagram

*Figure 2: use case diagram*

Screenshot

The system does not contain any GUI, so here are the screenshots of terminal. The client window (figure 3) will display new message sent from other clients. The server window (figure 4) will display the system broadcast message, such as “*server start…*”.

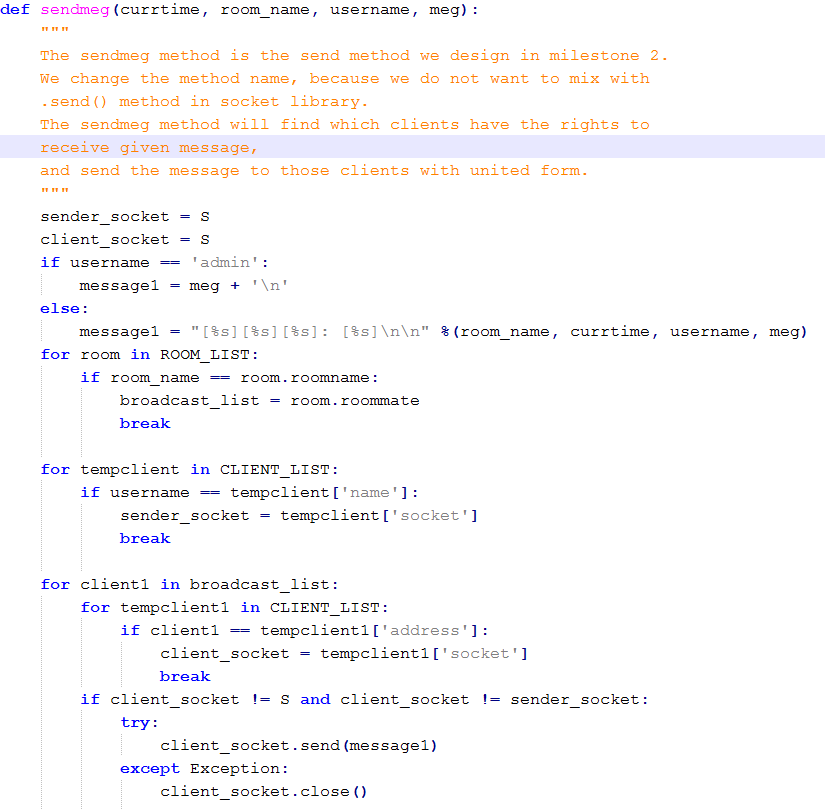
*Figure 3: client*



*Figure 4: server*

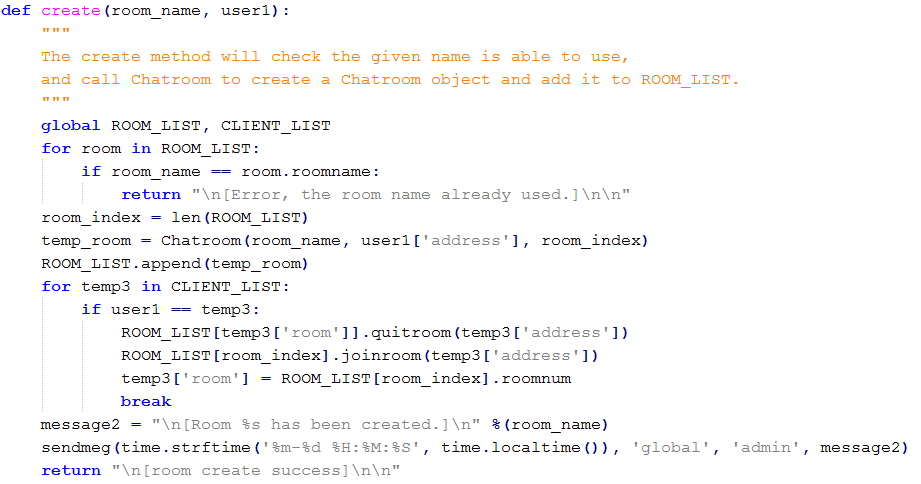
Code Snippets

The code snippet (figure 5 in next page) shows the implementation of basic send message functionality in the server class. This method will check if the given message is system broadcast first. Then, it finds the room members and the sender’s socket. Finally, it sends the message to the room members.



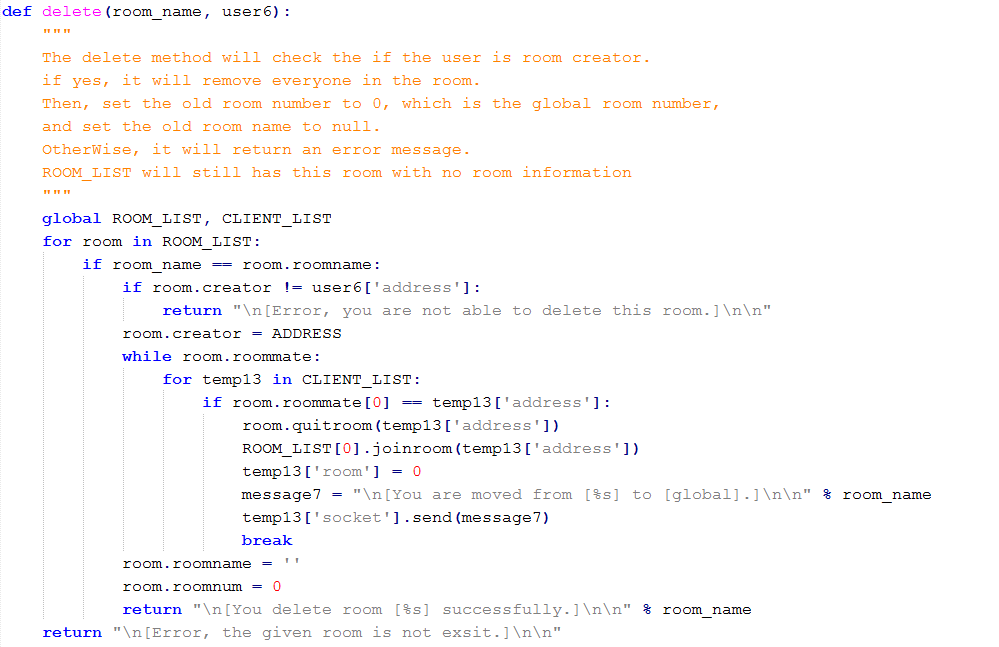
*Figure 5*

The code snippet (figure 6 in next page) shows the implementation of creating room functionality in the server class. This server will then check the room name availability. Then, it initializes the chat room and registers the creator with the chatroom.



*Figure 6*

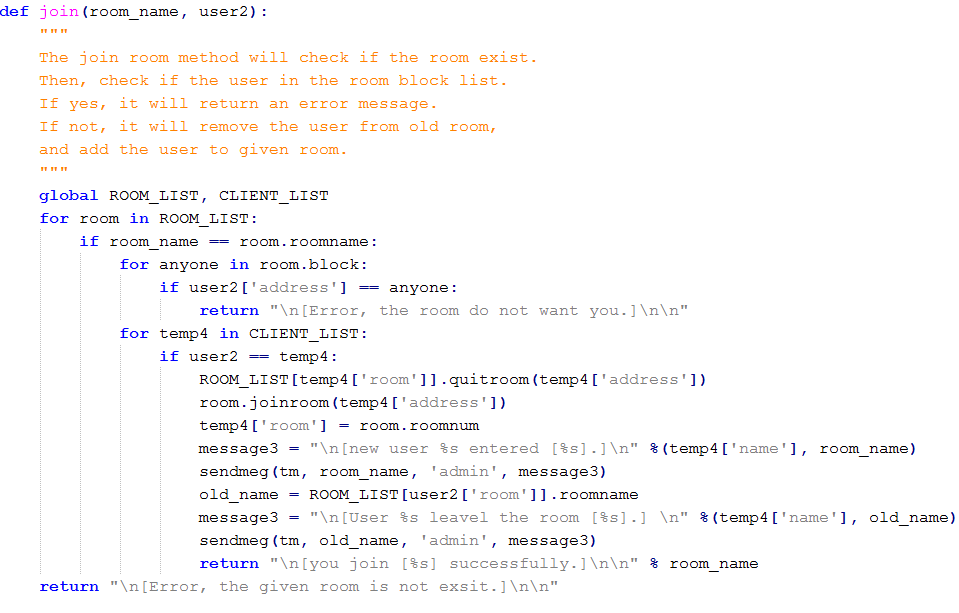
The code snippet (figure 7) shows the implementation of chatroom deleting functionality in the server class. This method will check if the requestor is the room creator. It will remove everyone in the room and add them to global if they are. Finally, the chatroom is cleaned up.

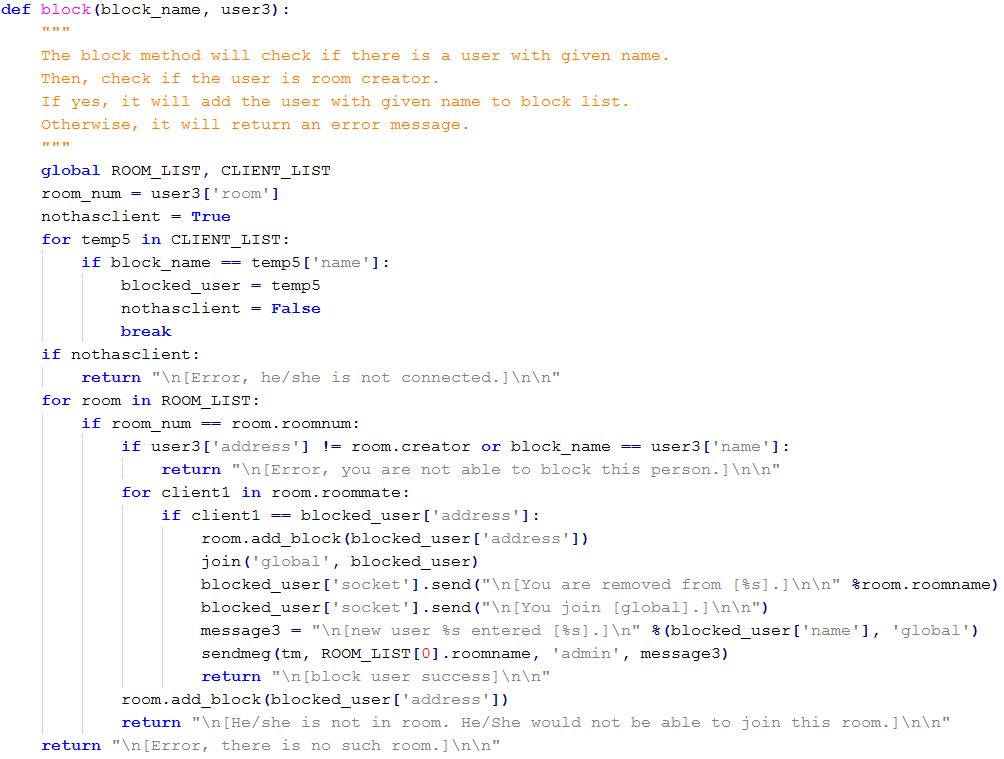


*Figure 7*

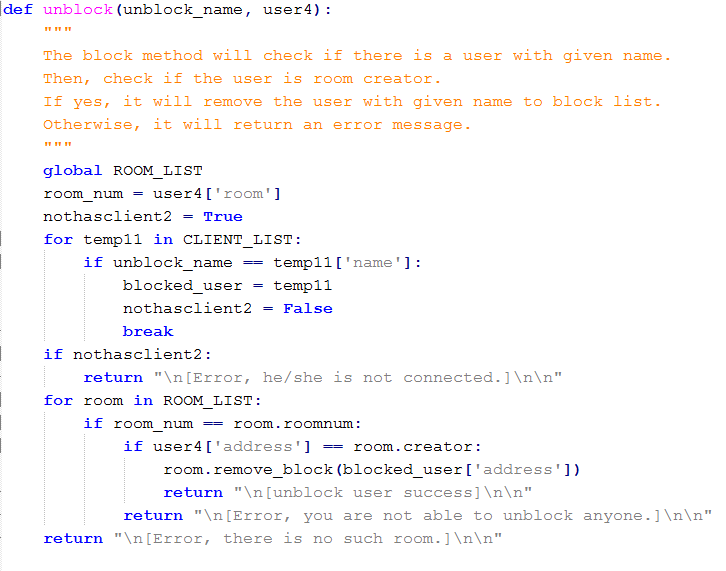
The code snippet (figure 8) shows the implementation of join room functionality in the server class. This method will first check if the given room exists. If yes, it will check if the user is in the room block list. If the user is not in the list, add the user to this room, and remove the user from its original chatroom’s member list.

*Figure 8*

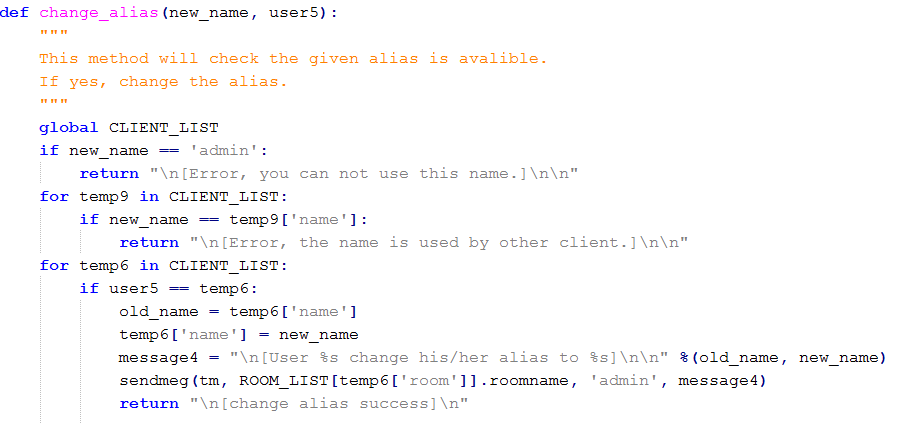


The code snippet (figure 9) shows the code that implement blocking user functionality in server class. This method will check if the blocked user exists. If yes, it will check if the requestor is the creator of the room. If the requestor is creator, remove the blocked user, and add the user to global.

*Figure 9*

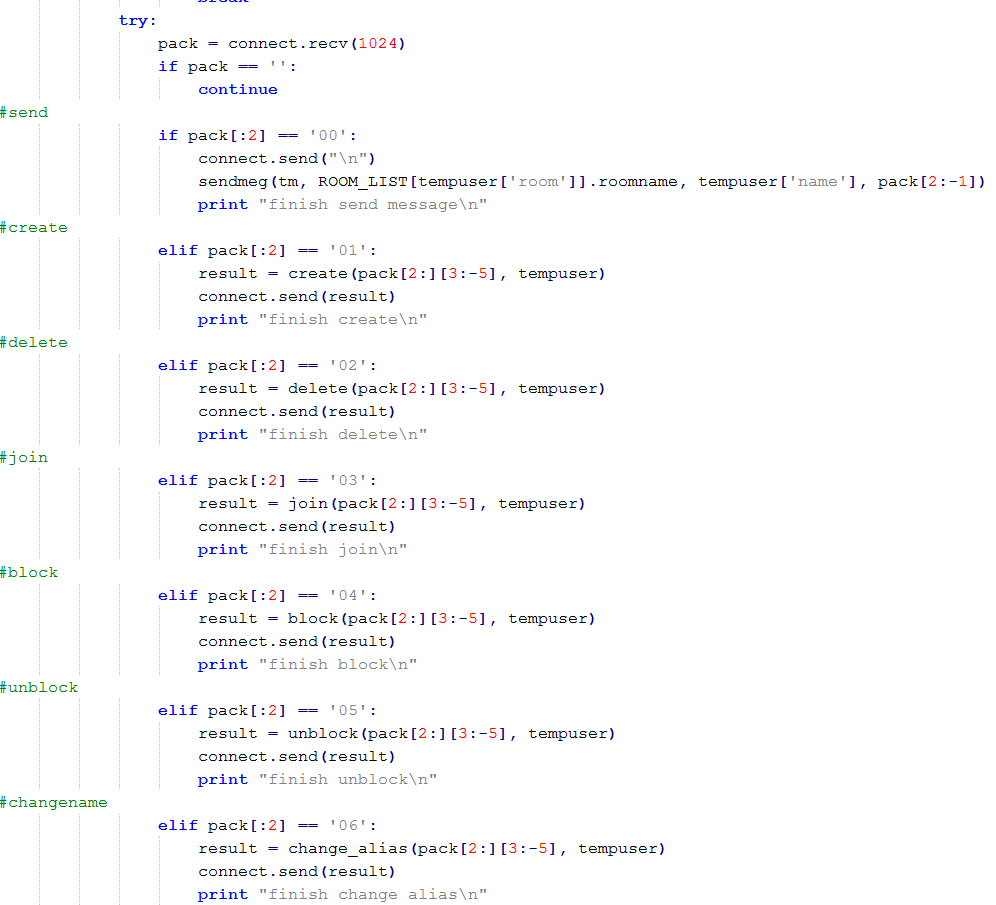
The code snippet (figure 10) shows the code that implement unblocking user functionality in server class. This method first checks the existence of the name in the room’s blocked list. If yes, it will check if the requestor is the room creator. If the requestor is the creator, remove the unblocked user from block list.

*Figure 10*

 The code snippet (figure 11) shows the code that implement changing alias functionality in server class. This method will check if the given name availability. If it is, it sets the user’s new alias in the chatroom.

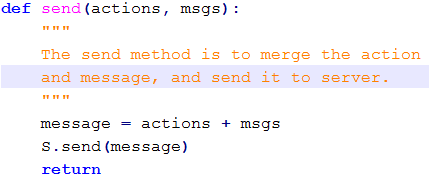
*Figure 11*

The code snippet (figure 12) shows the code that process the received data and call the right method.



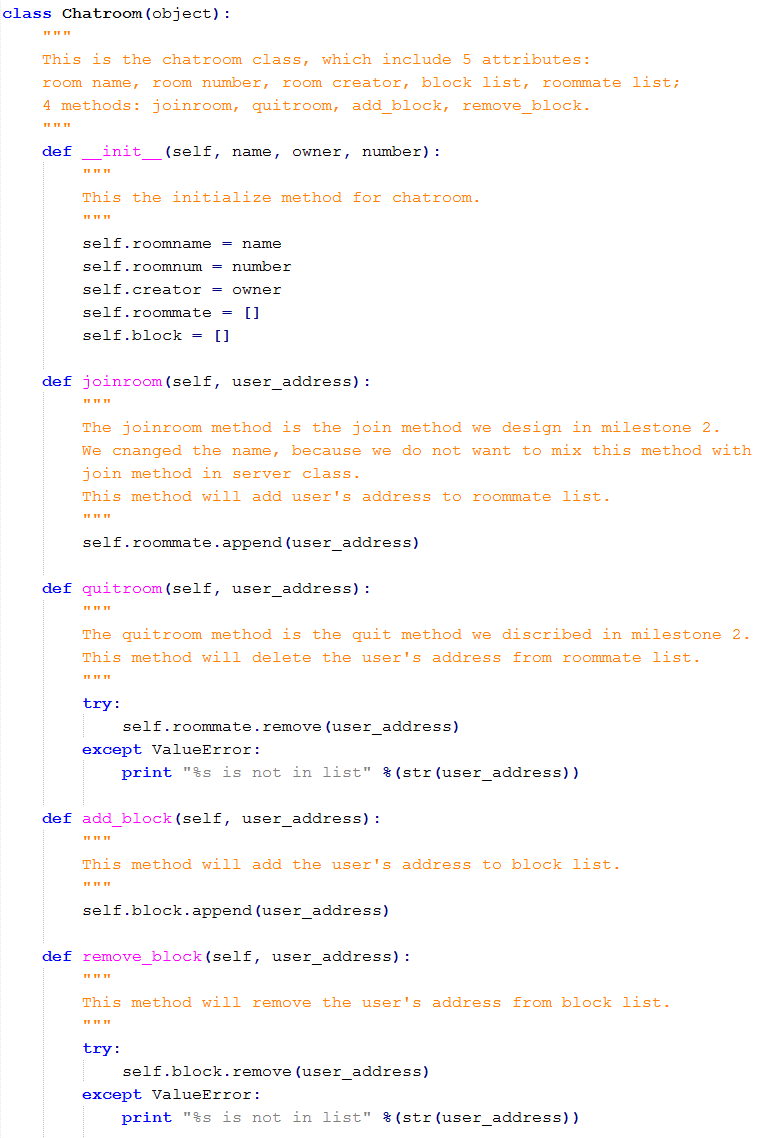
*Figure 12*

The code snippet (figure 13 in next page) shows the code that process the user’s input and split it to action and message in the client class. Then, the method send (figure 14 in next page) will merge them and send to server with united form.



*Figure 14*

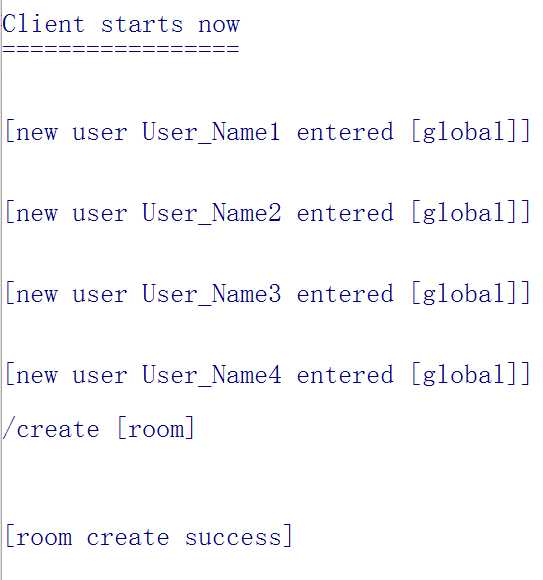
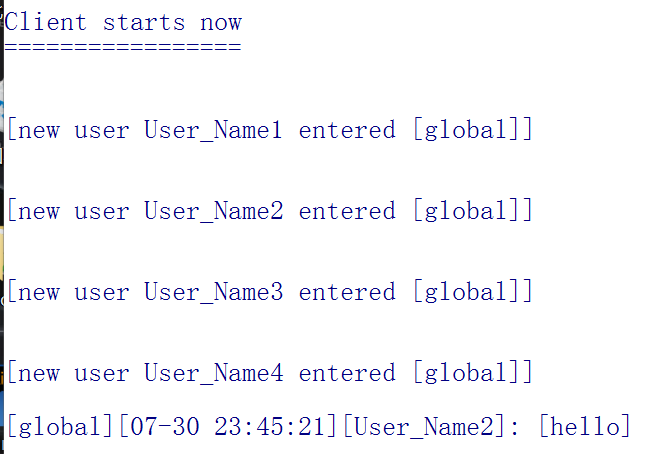
*Figure 13*

This code snippet (figure 15) shows the code of the Chatroom class. This class creates an object with 5 attributes and 4 methods.

*Figure 15*

Use Cases

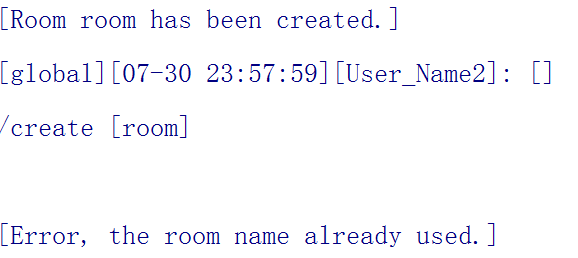
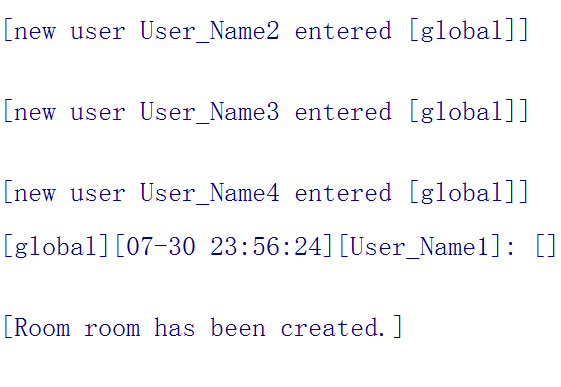
1. Send Message:

User type message, and when they tap enter key, the message shows on all clients in the form of [room name][time][alias]: [message] (figure 16).

*Figure 17*

*Figure 16*

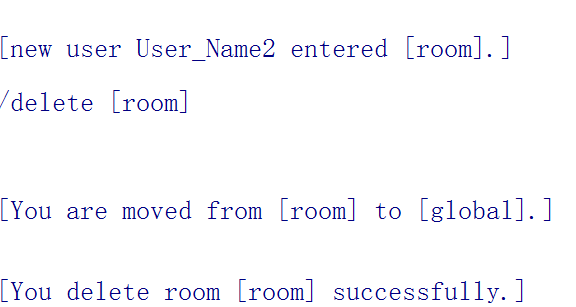
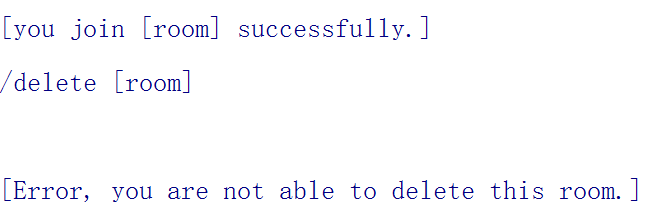
1. Create Chatroom:

User type “/create [xxx]”, and when he/she tap enter key, the chatroom named “xxx” will be enabled for everyone to join (figure 17 and 18). If the room name is already used, user will get an error message (figure 19).

*Figure 19*

*Figure 18*

1. Delete Chatroom:

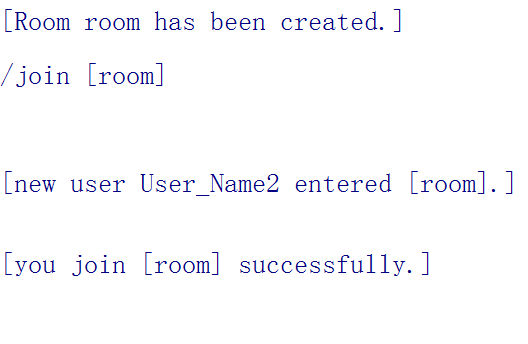
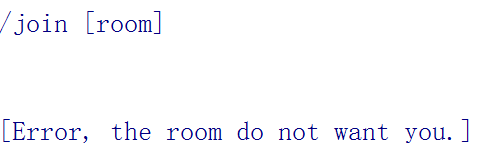
User type “/delete [xxx]”, and when he/she tap enter key, if he/she is the owner, the chatroom named “xxx” will be deleted (figure 20); if he/she is not the owner, he/she will get an error message (figure 21).

*Figure 21*

*Figure 20*

1. Join Chatroom:

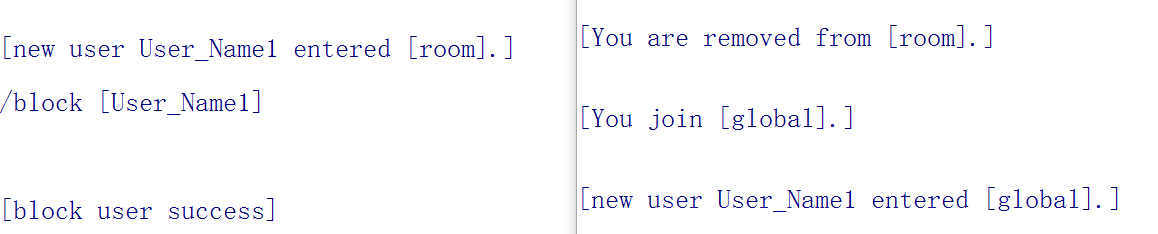
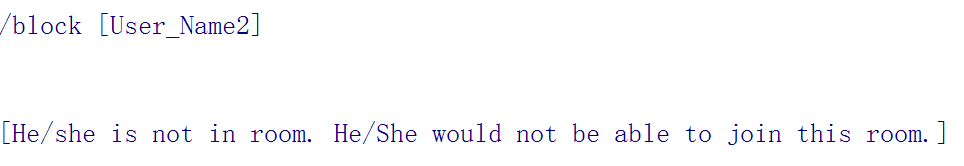
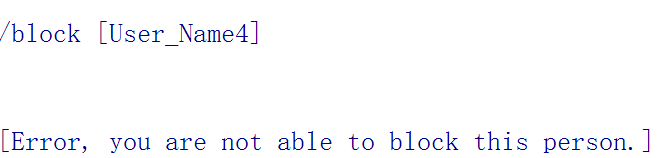
User types “/join [xxx]”, and when he/she tap enter key, he/she leaves their current room and join the chatroom named “xxx”, if chatroom “xxx” is created (figure 22). If the user is blocked by the target chatroom, he will receive an error message (figure 23).



*Figure 23*

*Figure 22*

1. Block User:

User types “/block [xxx]”. If a room creator types: “/block [xxx]”, the corresponding user xxx is removed from chatroom (figure 24) and/or denied to join this chatroom (figure 25). If other user (not the room creator) types this, that user gets error message (figure 26).

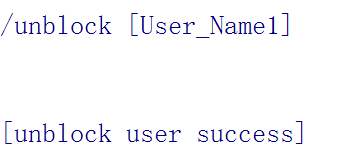
*Figure 24*

*Figure 25*

*Figure 26*

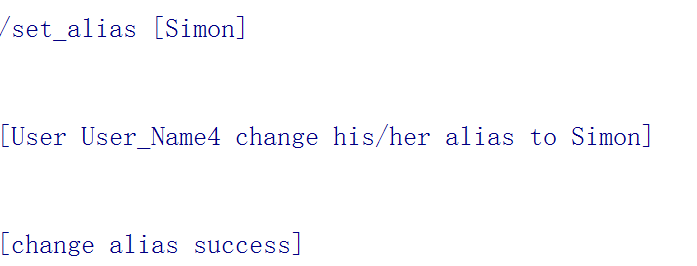
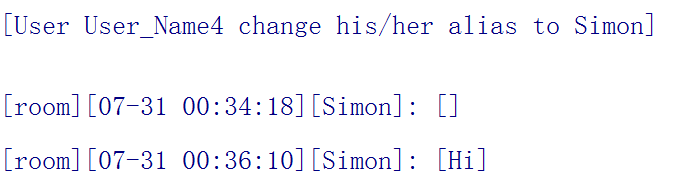
1. Unblock User:

Room creator types “/unblock [xxx]” (figure 27). When xxx typed “/join [chatroom]”, xxx would be able to join the chatroom (figure 22).



*Figure 27*

1. Change Alias:

A user wants to change their user alias which will be displayed in the chatroom. So, he typed: “/set\_alias [new alias]” (figure 28, 29).

*Figure 29*

*Figure 28*

Indication of Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| Client/Server | Index | Requirement | Requirement Met? |
| Client | 1.1.1 | System should have a space for the user to enter message | Met |
| 1.1.2 | System should be ready to take user input at anytime | Met |
| 1.1.3 | System should show timestamp of all messages | Met |
| 1.1.4 | Client shall be able to request to join a chatroom after client is connected with server | Met |
| 1.1.5 | The chat interface should display corresponding information including:  1.1.5.1 username(s)  1.1.5.2 current time  1.1.5.3 chatroom id | Met |
| 1.1.6 | Clients should connect with server and keep connecting after join chat room | Met |
| 2.1.1 | Clients shall be able to send 500 characters | Met |
| 2.1.2 | A client can only be able to join to one chat room at a time | Met |
| 2.1.4 | Client shall include the below information for a client to send to server including:  2.1.4.1 /join [chatroom\_name]  2.1.4.2 /create [chatroom\_name]  2.1.4.3 /set\_alias [alias]  2.1.4.4 /block [user\_alias]  2.1.4.5 /unblock [user\_alias] | Not met  It is created in the design phase but found not necessary to have all instruction included in a message.  In implementation, we found the client only need to send the right instruction to make the server know what to do.  We found this requirement is not satisfied when we finished the implementation and checking for the data format sent by clients. |
| Server | 1.2.1 | Server should be alive all the time even if there is no activity from client | Met |
| 1.2.4 | Server shall be able to create or delete a new chat room | Met |
| 2.2.1 | Messages shall be delivered within 5 secs | Met |
| 2.2.2 | The chat system shall support at least 20 clients at the same time | Met |
| 2.2.4 | Server shall only create one chat room | Met |
| 2.2.8 | Server should be able to accept a new chat request from client and close the corresponding connection after the chat terminates | Met |
| 2.2.9 | Server should check message to see if it going to be blocked by a user before forwarding it to the User | Not met  It is created in the design phase but found not necessary to check because the user is already banned from the room, he/she cannot receive any message from the chatroom.  In implementation, we found that the blocked user is not in the chatroom member list, so we don’t need to consider whether sending message to the blocked client.  We found this requirement is not met while we implementing the /block instruction at the server side. |

Design Process and Timeline

The design process of our project consists of four major phases: requirement, design, implementation and test. And we did the implementation and test phase interchangeably.

|  |  |  |
| --- | --- | --- |
| Time | Work | |
| May 9 | Start write milestone 1 | Draft edition for requirements specification |
|
| Purpose |
| May 10 | Continue on milestone 1 | Editing requirements specification |
| Relevant background required for the reader |
| May 16 | Continue on milestone 1 | Editing requirements specification |
| Draft of timeline |
| May 23 | Finish milestone 1 | Finish timeline |
| Finish requirements specification |
| Hand in milestone 1 |
| Jun 6 | Start milestone 2 | Discuss and design our chat system in terms of architecture |
| Jun 15 | Continue on milestone 2 | Design UML for our system |
| Jun 12 - 16 | Continue on milestone 2 | Write technical report on of our design |
| Jun 16 | Finish milestone 2 | Finish system design technical report |
| Hand in milestone 2 |
| July 1 | Start milestone 3 | Start review other group’s design |
| Take notes for that design |
| July 1 - 3 | Continue on milestone 3 | Discuss the design |
| Draft edition for technical review report |
| July 4 | Continue on milestone 3 | Editing technical review report |
| July 5 | Finish milestone 3 | Editing technical review report |
| Hand in milestone 3 |
| July 11 | Start milestone 4 | Read over the review report |
| Discuss the review report |
| Start implementing the system |
| July 18 | Continue on milestone 4 | Basic PPT framework |
| July 18-28 | Continue on milestone 4 | Implementing the advanced requirements of the chat system and test its functionality |
| July 27-28 | Continue on milestone 4 | Continue editing the PPT for demo |
| Video recording |
| July 28 | Continue on milestone 4 | In class demo |
| Hand in milestone 4 |
| July28-31 | Continue on milestone 4 | Write technique report for the project |
| Aug 5 | Finish milestone 4 | Hand in milestone 4 |

Our project timeline does not quite match the timeline we created in the first milestone since we lost one of our team member after the first milestone. Consequently, all of our implementation plan are delayed a little bit. Additionally, due to the huge difference in networking knowledge scope, one of our teammates have difficulty understanding the basic networking concept, so the workload is distributed between the two of us. Additionally, we met some technical difficulties, ranged from variable scope to data format simplification during the python programming stage (July 18-26). More specifically, the start date of milestone 2 has been delayed for the 13 days (Seng 299 midterm review). The start date of milestone 3 has been delayed for 11 days (final exam review session for other courses).

Problem Encountered

1. **Problem**: list element deleted in method shows up again in the main while loop of server.

**Solution**: define variable as global in method.

1. **Problem**: /Delete [chatroom\_name] delete chatrooms in a chatroom list causing individual user’s chatroom number not working properly.

**Solution**: instead of delete element from chatroom list, search through the whole current room member list and change their room\_number attributes to 0, which represents the general chatroom.

**Potential better solution**: use room name instead of room list index as individual chatroom identification in a user’s property.

1. **Problem**: client receive empty message from server causing text parsing error (index out of bound).

**Solution**: client prints server message directly.

**Potential better solution**: could write different cases in client in regards to the empty server message, and get greater control over the display.

1. **Problem**: cannot open multiple terminals at the same time to test the capability of 1,000 clients.

**Potential** **better** **solution**: used xterm command in mac to open multiple terminals at the same time.

Contributions and Contributors

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Milestone | Contributions | | | contributor(s) |
| milestone 1 | Purpose | | | Rich Chen\* & Haotian Shen |
| Relevant background required for the reader | | | Pengxiang Jia |
| Requirements specification | | | Rich Chen\* & Haotian Shen |
| Timeline | | | Zelan Xiang |
| Editing the report | | | Rich Chen\* & Haotian Shen & Zelan Xiang |
| milestone 2 | Purpose | | | Zelan Xiang |
| Relevant Background Required for the Reader | | | Zelan Xiang |
| Design process and decisions | | | Haotian Shen |
| UML diagram and textual | | Class diagram | Zelan Xiang & Haotian Shen |
| Activity diagram | Zelan Xiang |
| Use Case diagram | Zelan Xiang |
| Use Cases | Zelan Xiang & Haotian Shen |
| Implementation plan | | | Zelan Xiang & Haotian Shen |
| Update Project Plan | | | Zelan Xiang |
| Updates to Requirements | | | Haotian Shen |
| Editing the report | | | Zelan Xiang & Haotian Shen |
| Contribution table | | | Haotian Shen |
| milestone 3 | Table of contents | | | Zelan Xiang |
| Purpose | | | Pengxiang Jia |
| Summary of the designed system | | | Pengxiang Jia & Haotian Shen |
| Critical review | | Requirements | Haotian Shen |
| Class Diagram and Activity diagram | Zelan Xiang |
| Use Case | Pengxiang Jia |
| Summary of review | | | Haotian Shen & Zelan Xiang |
| Recommendations | | | Zelan Xiang & Haotian Shen & Pengxiang Jia |
| Editing the report | | | Haotian Shen & Zelan Xiang |
| Contribution table | | | Zelan Xiang |
| milestone 4 | Implementation and test: part 1 | | Implementation: client class | Haotian Shen |
| Implementation: the connection part and sending message functions of server class | Zelan XIang |
| Test: basic connection between server and clients and sending messages to other clients | Haotian Shen & Zelan Xiang |
| Implementation and test: part 2 | | Implementation: the changing alias function of server class | Zelan Xiang |
| Test: change alias | Zelan Xiang |
| Implementation and test: part 3 | | Implementation: edit the client class | Haotian Shen & Zelan Xiang |
| Implementation: the Chatroom class; the creating room, block user, unblock user, and joining room functions of server class | Zelan Xiang |
| Test: create chatroom, join room, block user, unblock user | Zelan Xiang & Haotian Shen |
| Implementation and test: part 4 | | Implementation: edit the client class | Haotian Shen |
| Implementation: the deleting room function of server class | Zelan Xiang & Haotian Shen |
| Test: delete room | Zelan Xiang & Haotian Shen |
| Checking and simplifying code | | | Zelan Xiang |
| Demo | | Basic PPT framework | Pengxiang Jia |
| Edit PPT | Haotian Shen |
| UML diagrams | Zelan Xiang |
| Demo vedio record | Zelan Xiang |
| Table of contents | | | Zelan Xiang |
| Purpose | | | Haotian Shen |
| Review of the design | | | Zelan Xiang & Haotian Shen |
| Description of implementation | Overall and Design Pattern | | Zelan Xiang & Pengxiang Jia |
| UML Diagrams | | Zelan Xiang |
| Code Snippets | | Zelan Xiang |
| Screenshots | | Zelan Xiang |
| Use Cases | | Zelan Xiang |
| Indication requirements | | | Haotian Shen |
| Design process and timeline | | | Hantian Shen |
| Contribution table | | | Zelan Xiang |
| Editing the report | | | Zelan Xiang & Haotian Shen |
| Problem encountered | | | Haotian Shen |

\*Rich Chen dropped the class

Acknowledgements

This is a report for Summer 2017 Software Architecture & Design course and we would like to thank Caleb Shortt for his guidance during the course and his patience in solving problems that we have encountered.