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

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 send the action-message pair 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.

According to the implementation, the coupling of our system is quite low, and the cohesion is quite high. And our system has a Chatroom class to take care of actions about the chatroom, which is indirection design pattern (GRASP).

Class Diagram

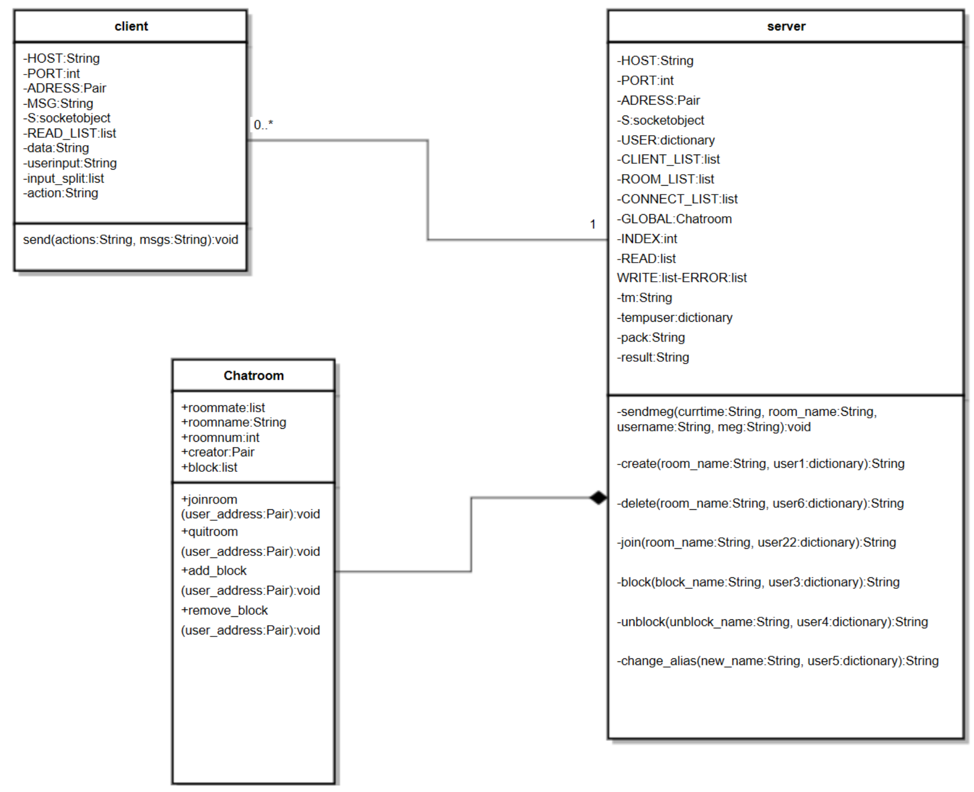

client
-HOST:String
-PORT:int
-ADRESS:Pair
-MSG:String
-S:socketobject
-READ_LIST:list
-data:String
-userinput:String
-input_split:list
-action:String
send(actions:String, msgs:String):void
0..*
1
server
-HOST:String
-PORT:int
-ADRESS:Pair
-S:socketobject
-USER:dictionary
-CLIENT_LIST:list
-ROOM_LIST:list
-CONNECT_LIST:list
-GLOBAL:Chatroom
-INDEX:int
-READ:list
WRITE:list-ERROR:list
-tm:String
-tempuser:dictionary
-pack:String
-result:String
-sendmeg(currtime:String, room_name:String, username:String, meg:String):void
-create(room_name:String, user1:dictionary):String
-delete(room_name:String, user6:dictionary):String
-join(room_name:String, user22:dictionary):String
-block(block_name:String, user3:dictionary):String
-unblock(unblock_name:String, user4:dictionary):String
-change_alias(new_name:String, user5:dictionary):String
Chatroom
+roommate:list
+roomname:String
+roomnum:int
+creator:Pair
+block:list
+joinroom (user_address:Pair):void
+quitroom (user_address:Pair):void
+add_block (user_address:Pair):void
+remove_block (user_address:Pair):void

Figure 1: class diagram

Use Cases Diagram

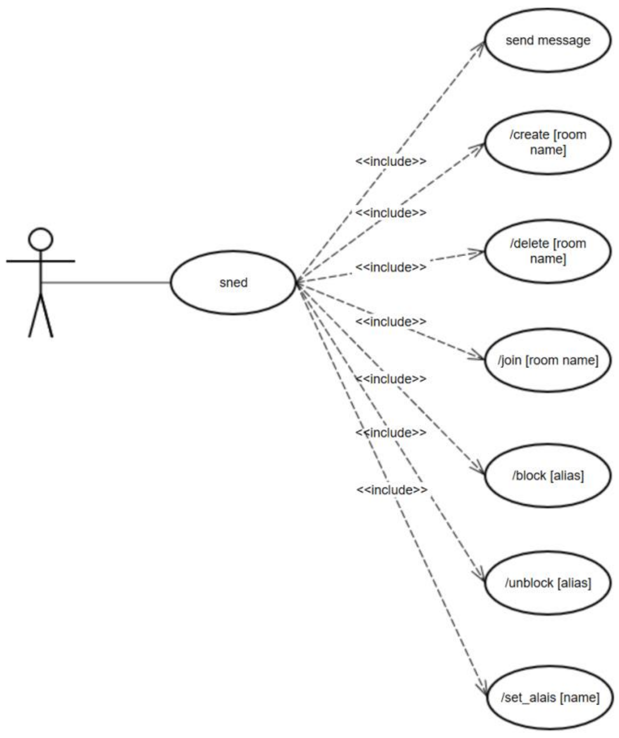
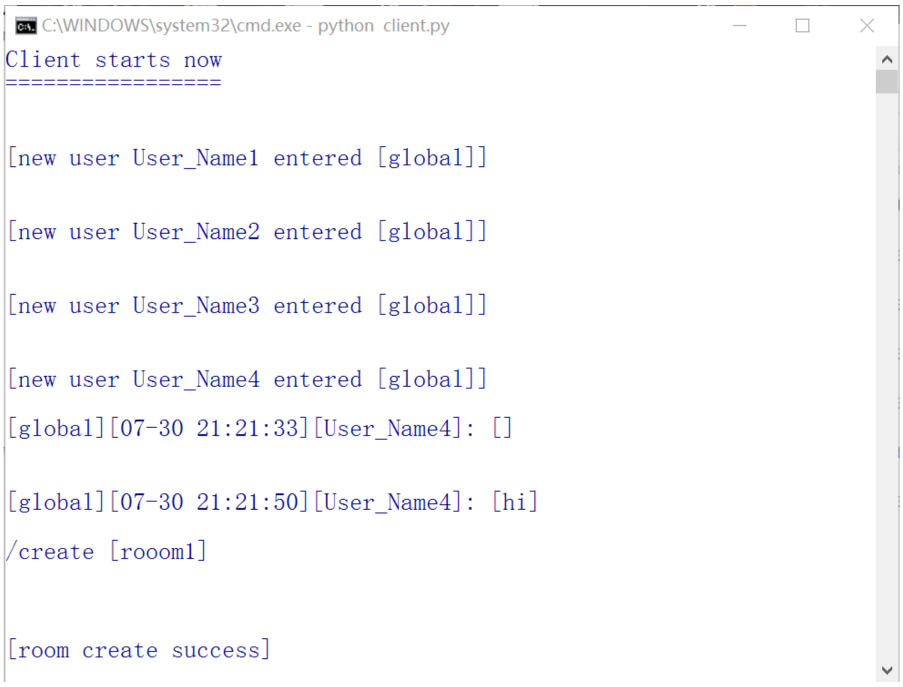


Figure 2: use case diagram

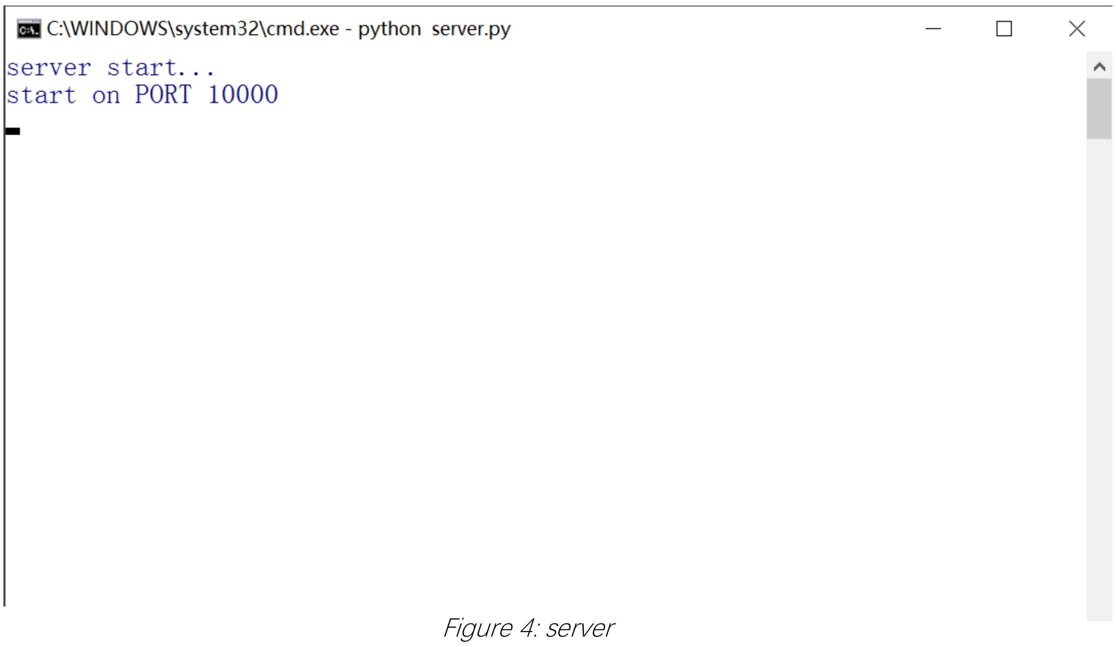
Screenshot

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…*".



```
Client starts now
=================


[new user User_Name1 entered [global]]

[new user User_Name2 entered [global]]

[new user User_Name3 entered [global]]

[new user User_Name4 entered [global]]
[global][07-30 21:21:33][User_Name4]: []

[global][07-30 21:21:50][User_Name4]: [hi]
/create [rooom1]


[room create success]
```

Figure 3: client

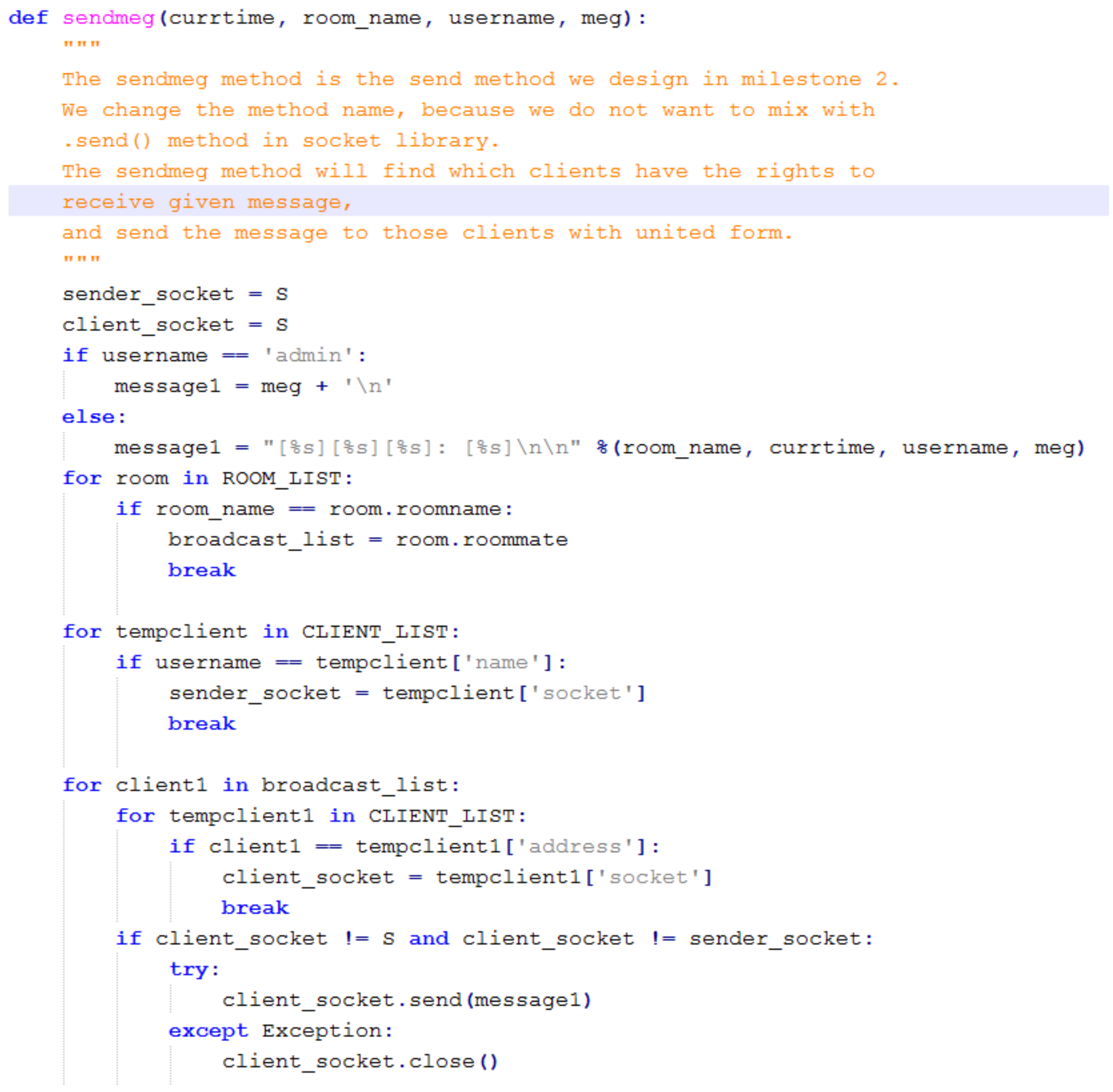


```
server start...
start on PORT 10000
```

Figure 4: server

Code Snippets

The code snippet (figure 5) 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.



```python
def sendmeg(currtime, room_name, username, meg):
    """
    The sendmeg method is the send method we design in milestone 2.
    We change the method name, because we do not want to mix with
    .send() method in socket library.
    The sendmeg method will find which clients have the rights to
    receive given message,
    and send the message to those clients with united form.
    """
    sender_socket = S
    client_socket = S
    if username == 'admin':
        message1 = meg + '\n'
    else:
        message1 = "[%s][%s][%s]: [%s]\n\n" %(room_name, currtime, username, meg)
    for room in ROOM_LIST:
        if room_name == room.roomname:
            broadcast_list = room.roommate
            break

    for tempclient in CLIENT_LIST:
        if username == tempclient['name']:
            sender_socket = tempclient['socket']
            break

    for client1 in broadcast_list:
        for tempclient1 in CLIENT_LIST:
            if client1 == tempclient1['address']:
                client_socket = tempclient1['socket']
                break
        if client_socket != S and client_socket != sender_socket:
            try:
                client_socket.send(message1)
            except Exception:
                client_socket.close()
```

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.

```python
def create(room_name, user1):
    """
    The create method will check the given name is able to use,
    and call Chatroom to create a Chatroom object and add it to ROOM_LIST.
    """
    global ROOM_LIST, CLIENT_LIST
    for room in ROOM_LIST:
        if room_name == room.roomname:
            return "\n[Error, the room name already used.]\n\n"
    room_index = len(ROOM_LIST)
    temp_room = Chatroom(room_name, user1['address'], room_index)
    ROOM_LIST.append(temp_room)
    for temp3 in CLIENT_LIST:
        if user1 == temp3:
            ROOM_LIST[temp3['room']].quitroom(temp3['address'])
            ROOM_LIST[room_index].joinroom(temp3['address'])
            temp3['room'] = ROOM_LIST[room_index].roomnum
            break
    message2 = "\n[Room %s has been created.]\n" %(room_name)
    sendmeg(time.strftime('%m-%d %H:%M:%S', time.localtime()), 'global', 'admin', message2)
    return "\n[room create success]\n\n"
```

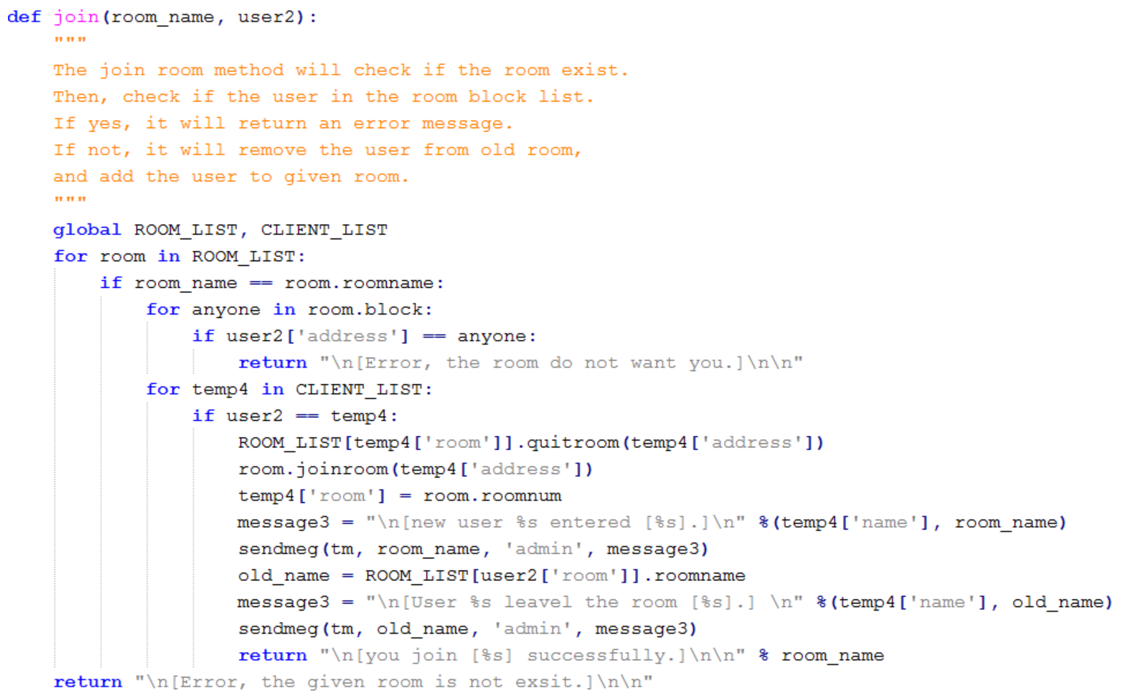
Figure 6

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.

```python
def delete(room_name, user6):
    """
    The delete method will check the if the user is room creator.
    if yes, it will remove everyone in the room.
    Then, set the old room number to 0, which is the global room number,
    and set the old room name to null.
    OtherWise, it will return an error message.
    ROOM_LIST will still has this room with no room information
    """
    global ROOM_LIST, CLIENT_LIST
    for room in ROOM_LIST:
        if room_name == room.roomname:
            if room.creator != user6['address']:
                return "\n[Error, you are not able to delete this room.]\n\n"
            room.creator = ADDRESS
            while room.roommate:
                for temp13 in CLIENT_LIST:
                    if room.roommate[0] == temp13['address']:
                        room.quitroom(temp13['address'])
                        ROOM_LIST[0].joinroom(temp13['address'])
                        temp13['room'] = 0
                        message7 = "\n[You are moved from [%s] to [global].]\n\n" % room_name
                        temp13['socket'].send(message7)
                        break
            room.roomname = ''
            room.roomnum = 0
            return "\n[You delete room [%s] successfully.]\n\n" % room_name
    return "\n[Error, the given room is not exsit.]\n\n"
```

Figure 7

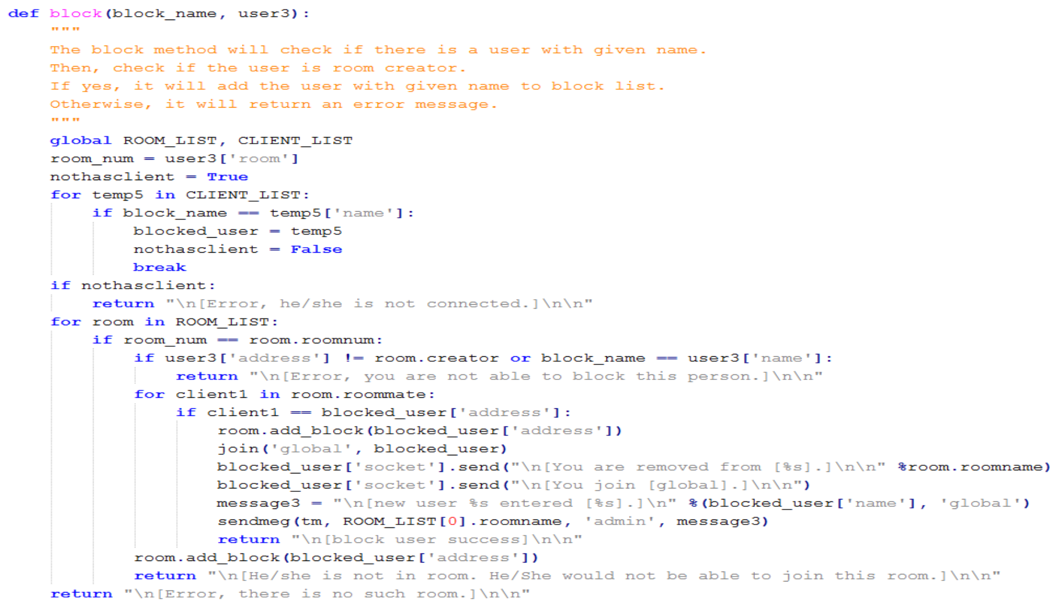
The code snippet (figure 8) shows the implementation of join room functionality in the server class. This method will first check if the given room exists. If yes, it will check if the user is in the room block list. If the user is not in the list, add the user to this room, and remove the user from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



```
def join(room_name, user2):
    """
    The join room method will check if the room exist.
    Then, check if the user in the room block list.
    If yes, it will return an error message.
    If not, it will remove the user from old room,
    and add the user to given room.
    """
    global ROOM_LIST, CLIENT_LIST
    for room in ROOM_LIST:
        if room_name == room.roomname:
            for anyone in room.block:
                if user2['address'] == anyone:
                    return "\n[Error, the room do not want you.]\n\n"
            for temp4 in CLIENT_LIST:
                if user2 == temp4:
                    ROOM_LIST[temp4['room']].quitroom(temp4['address'])
                    room.joinroom(temp4['address'])
                    temp4['room'] = room.roomnum
                    message3 = "\n[new user %s entered [%s].]\n" %(temp4['name'], room_name)
                    sendmeg(tm, room_name, 'admin', message3)
                    old_name = ROOM_LIST[user2['room']].roomname
                    message3 = "\n[User %s leavel the room [%s].] \n" %(temp4['name'], old_name)
                    sendmeg(tm, old_name, 'admin', message3)
                    return "\n[you join [%s] successfully.]\n\n" % room_name
    return "\n[Error, the given room is not exsit.]\n\n"
```

Figure 8

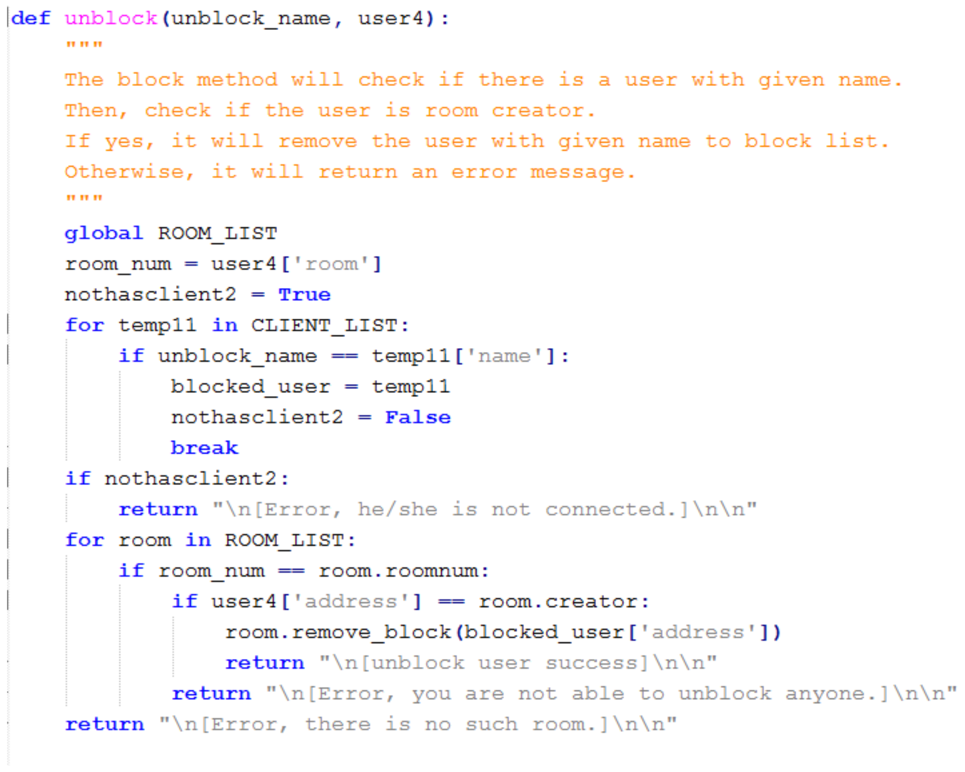
The code snippet (figure 9) shows the code that implement blocking user functionality in server class. This method will check if the blocked user exists. If yes, it will check if the requestor is the creator of the room. If the requestor is creator, remove the blocked user, and add the user to global.



```
def block(block_name, user3):
    """
    The block method will check if there is a user with given name.
    Then, check if the user is room creator.
    If yes, it will add the user with given name to block list.
    Otherwise, it will return an error message.
    """
    global ROOM_LIST, CLIENT_LIST
    room_num = user3['room']
    nothasclient = True
    for temp5 in CLIENT_LIST:
        if block_name == temp5['name']:
            blocked_user = temp5
            nothasclient = False
            break
    if nothasclient:
        return "\n[Error, he/she is not connected.]\n\n"
    for room in ROOM_LIST:
        if room_num == room.roomnum:
            if user3['address'] != room.creator or block_name == user3['name']:
                return "\n[Error, you are not able to block this person.]\n\n"
            for client1 in room.roommate:
                if client1 == blocked_user['address']:
                    room.add_block(blocked_user['address'])
                    join('global', blocked_user)
                    blocked_user['socket'].send("\n[You are removed from [%s].]\n\n" %room.roomname)
                    blocked_user['socket'].send("\n[You join [global].]\n\n")
                    message3 = "\n[new user %s entered [%s].]\n" %(blocked_user['name'], 'global')
                    sendmeg(tm, ROOM_LIST[0].roomname, 'admin', message3)
                    return "\n[block user success]\n\n"
            room.add_block(blocked_user['address'])
            return "\n[He/she is not in room. He/She would not be able to join this room.]\n\n"
    return "\n[Error, there is no such room.]\n\n"
```

Figure 9

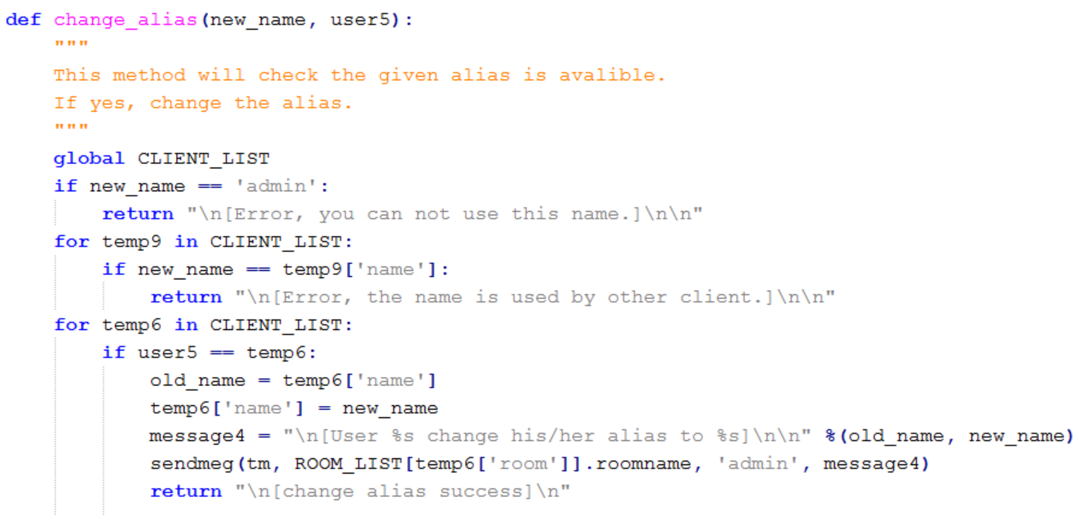
The code snippet (figure 10) shows the code that implement unblocking user functionality in server class. This method first checks the existence of the name in the room's blocked list. If yes, it will check if the requestor is the room creator. If the requestor is the creator, remove the unblocked user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



```
def unblock(unblock_name, user4):
    """
    The block method will check if there is a user with given name.
    Then, check if the user is room creator.
    If yes, it will remove the user with given name to block list.
    Otherwise, it will return an error message.
    """
    global ROOM_LIST
    room_num = user4['room']
    nothasclient2 = True
    for temp11 in CLIENT_LIST:
        if unblock_name == temp11['name']:
            blocked_user = temp11
            nothasclient2 = False
            break
    if nothasclient2:
        return "\n[Error, he/she is not connected.]\n\n"
    for room in ROOM_LIST:
        if room_num == room.roomnum:
            if user4['address'] == room.creator:
                room.remove_block(blocked_user['address'])
                return "\n[unblock user success]\n\n"
            return "\n[Error, you are not able to unblock anyone.]\n\n"
    return "\n[Error, there is no such room.]\n\n"
```

Figure 10

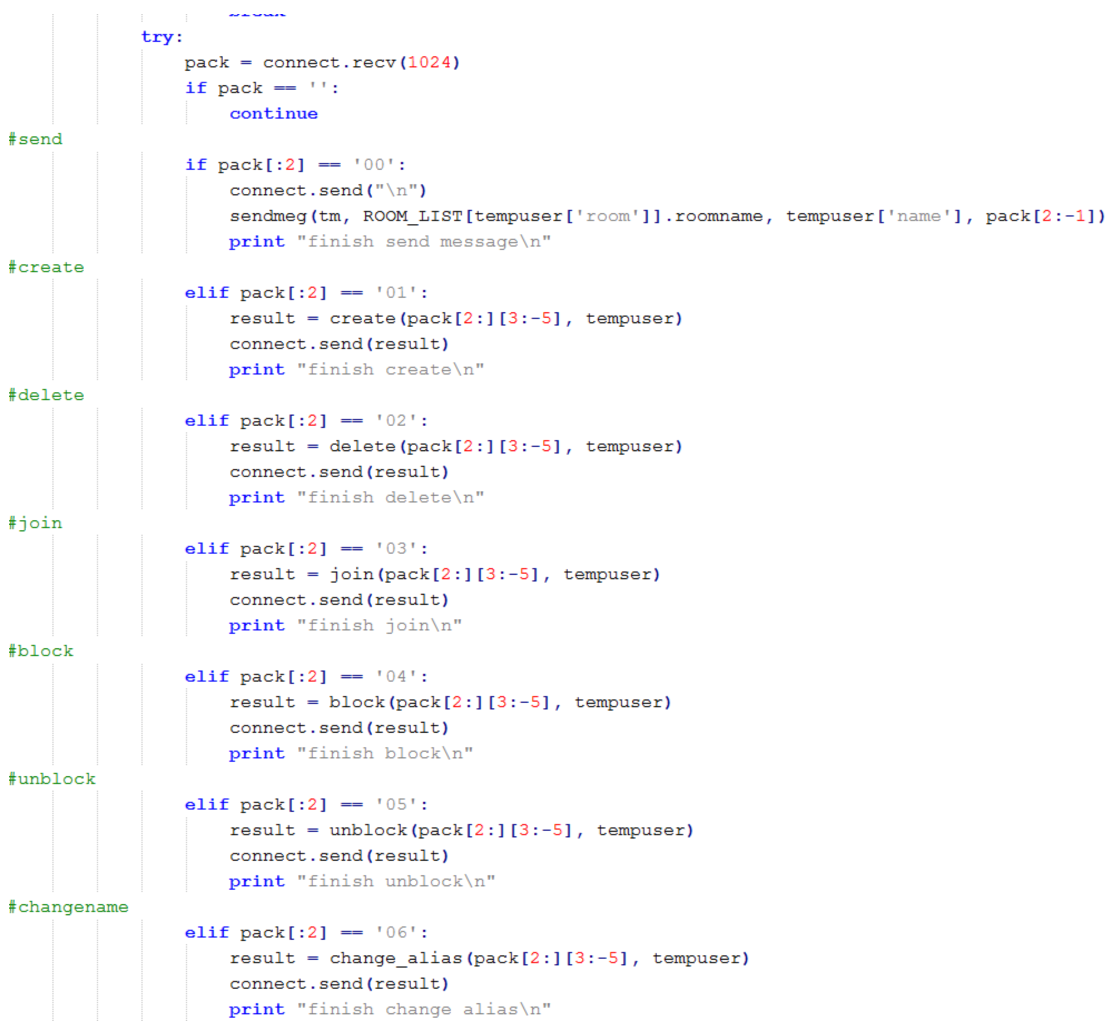
The code snippet (figure 11) shows the code that implement changing alias functionality in server class. This method will check if the given name availability. If it is, it sets the user's new alias in the chatroom.



```
def change_alias(new_name, user5):
    """
    This method will check the given alias is avalible.
    If yes, change the alias.
    """
    global CLIENT_LIST
    if new_name == 'admin':
        return "\n[Error, you can not use this name.]\n\n"
    for temp9 in CLIENT_LIST:
        if new_name == temp9['name']:
            return "\n[Error, the name is used by other client.]\n\n"
    for temp6 in CLIENT_LIST:
        if user5 == temp6:
            old_name = temp6['name']
            temp6['name'] = new_name
            message4 = "\n[User %s change his/her alias to %s]\n\n" %(old_name, new_name)
            sendmeg(tm, ROOM_LIST[temp6['room']].roomname, 'admin', message4)
            return "\n[change alias success]\n"
```

Figure 11

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



```
            try:
                pack = connect.recv(1024)
                if pack == '':
                    continue
#send
                if pack[:2] == '00':
                    connect.send("\n")
                    sendmeg(tm, ROOM_LIST[tempuser['room']].roomname, tempuser['name'], pack[2:-1])
                    print "finish send message\n"
#create
                elif pack[:2] == '01':
                    result = create(pack[2:][3:-5], tempuser)
                    connect.send(result)
                    print "finish create\n"
#delete
                elif pack[:2] == '02':
                    result = delete(pack[2:][3:-5], tempuser)
                    connect.send(result)
                    print "finish delete\n"
#join
                elif pack[:2] == '03':
                    result = join(pack[2:][3:-5], tempuser)
                    connect.send(result)
                    print "finish join\n"
#block
                elif pack[:2] == '04':
                    result = block(pack[2:][3:-5], tempuser)
                    connect.send(result)
                    print "finish block\n"
#unblock
                elif pack[:2] == '05':
                    result = unblock(pack[2:][3:-5], tempuser)
                    connect.send(result)
                    print "finish unblock\n"
#changename
                elif pack[:2] == '06':
                    result = change_alias(pack[2:][3:-5], tempuser)
                    connect.send(result)
                    print "finish change alias\n"
```

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.

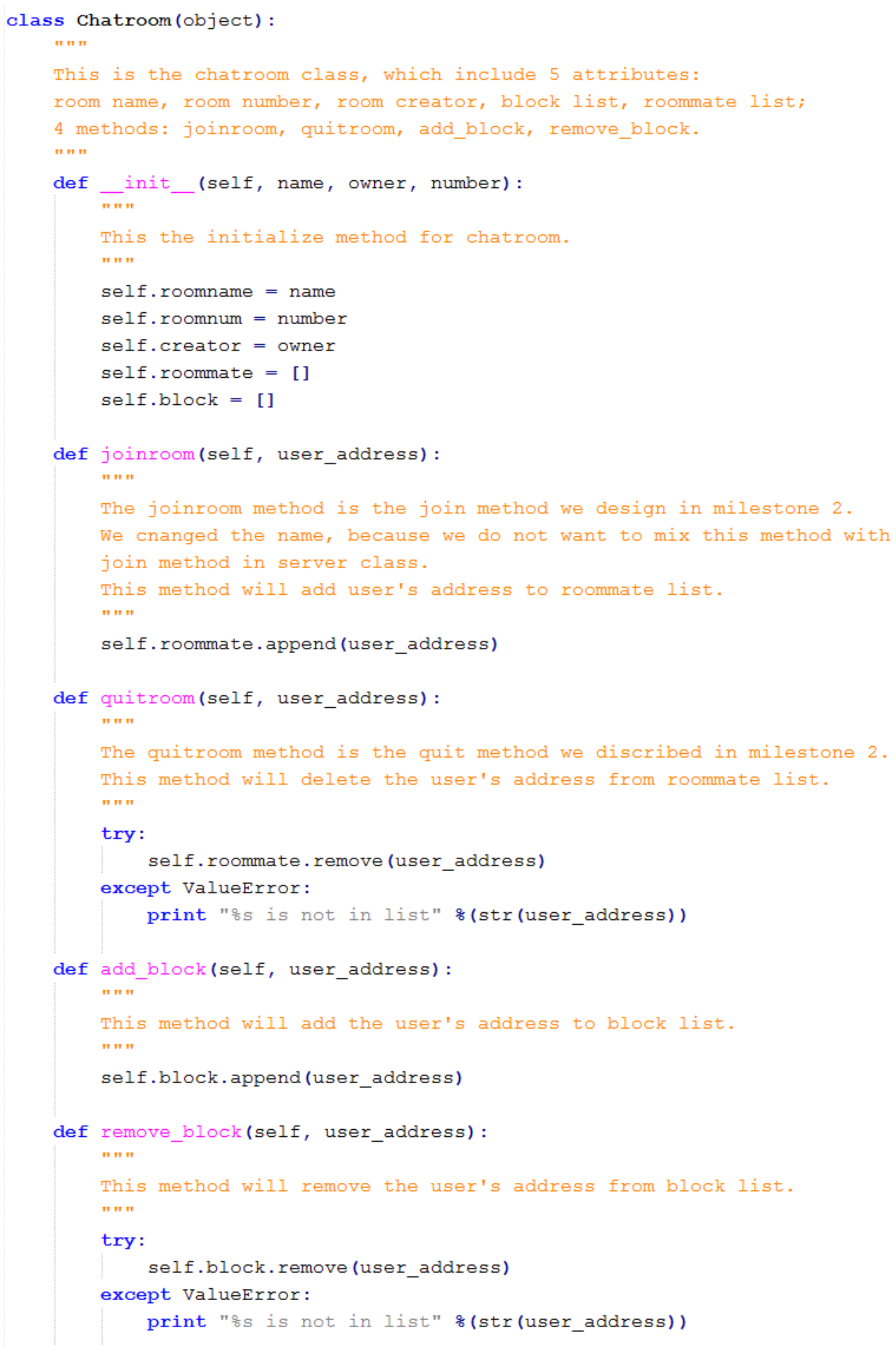
```
while True:
    READ_LIST = [sys.stdin, S]
    for sock in READ_LIST:
        if sock == S:
            try:
                data = sock.recv(1024)
                if data:
                    sys.stdout.write(data)
                    sys.stdout.flush()
                else:
                    print "disconnected"
                    sys.exit()
            except:
                sys.exit()
        else:
            userinput = sys.stdin.readline()
            input_split = userinput.split(" ")
            if len(input_split) > 1:
                MSG = str(input_split[1:])

            if input_split[0] == "/quit":
                S.close()
                quit()
            elif input_split[0] == "/create":
                action = '01'
            elif input_split[0] == "/delete":
                action = '02'
            elif input_split[0] == "/join":
                action = '03'
            elif input_split[0] == "/block":
                action = '04'
            elif input_split[0] == "/unblock":
                action = '05'
            elif input_split[0] == "/set_alias":
                action = '06'
            else:
                action = '00'
                MSG = input_split[0:]
                MSG = reduce((lambda x, y: x+' '+y), MSG)
            try:
                send(action, MSG)
            except:
                print "disconnected"
                sys.exit()
            sys.stdout.flush()
```

Figure 13

```
def send(actions, msgs):
    """
    The send method is to merge the action
    and message, and send it to server.
    """
    message = actions + msgs
    S.send(message)
    return
```

Figure 14

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



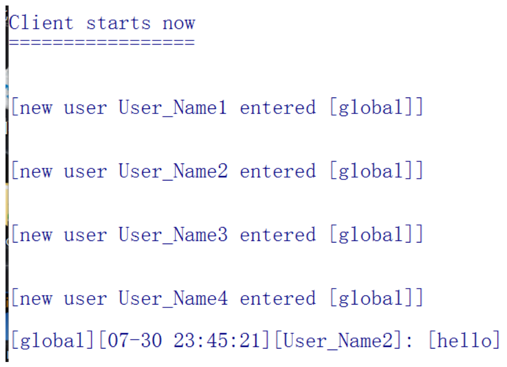
```
class Chatroom(object):
    """
    This is the chatroom class, which include 5 attributes:
    room name, room number, room creator, block list, roommate list;
    4 methods: joinroom, quitroom, add_block, remove_block.
    """
    def __init__(self, name, owner, number):
        """
        This the initialize method for chatroom.
        """
        self.roomname = name
        self.roomnum = number
        self.creator = owner
        self.roommate = []
        self.block = []

    def joinroom(self, user_address):
        """
        The joinroom method is the join method we design in milestone 2.
        We cnanged the name, because we do not want to mix this method with
        join method in server class.
        This method will add user's address to roommate list.
        """
        self.roommate.append(user_address)

    def quitroom(self, user_address):
        """
        The quitroom method is the quit method we discribed in milestone 2.
        This method will delete the user's address from roommate list.
        """
        try:
            self.roommate.remove(user_address)
        except ValueError:
            print "%s is not in list" %(str(user_address))

    def add_block(self, user_address):
        """
        This method will add the user's address to block list.
        """
        self.block.append(user_address)

    def remove_block(self, user_address):
        """
        This method will remove the user's address from block list.
        """
        try:
            self.block.remove(user_address)
        except ValueError:
            print "%s is not in list" %(str(user_address))
```

Figure 15

Use Cases

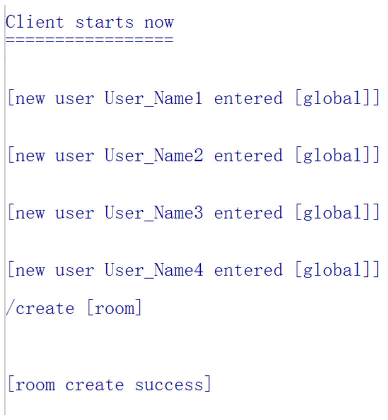
1. Send Message:

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).



```
Client starts now
=================

[new user User_Name1 entered [global]]

[new user User_Name2 entered [global]]

[new user User_Name3 entered [global]]

[new user User_Name4 entered [global]]
[global][07-30 23:45:21][User_Name2]: [hello]
```

Figure 16

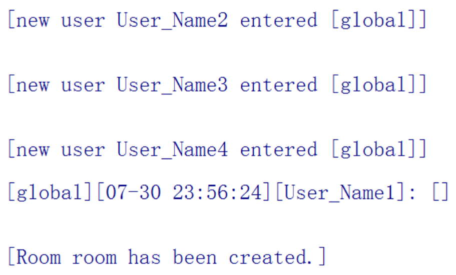


```
Client starts now
=================

[new user User_Name1 entered [global]]

[new user User_Name2 entered [global]]

[new user User_Name3 entered [global]]

[new user User_Name4 entered [global]]
/create [room]

[room create success]
```

Figure 17

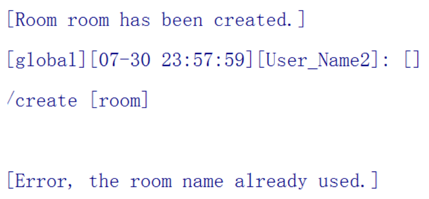
2. Create Chatroom:

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).



```
[new user User_Name2 entered [global]]

[new user User_Name3 entered [global]]

[new user User_Name4 entered [global]]
[global][07-30 23:56:24][User_Name1]: []

[Room room has been created.]
```

Figure 18

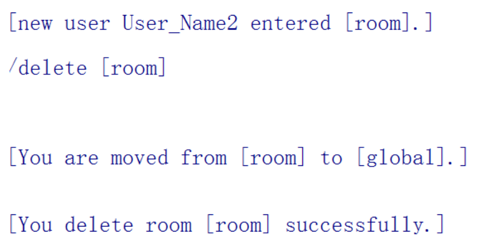


```
[Room room has been created.]
[global][07-30 23:57:59][User_Name2]: []
/create [room]

[Error, the room name already used.]
```

Figure 19

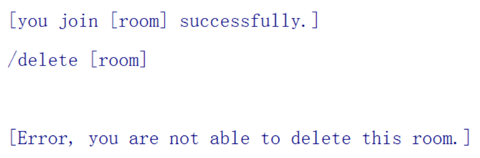
3. Delete Chatroom:

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).



```
[new user User_Name2 entered [room].]
/delete [room]

[You are moved from [room] to [global].]

[You delete room [room] successfully.]
```

Figure 20

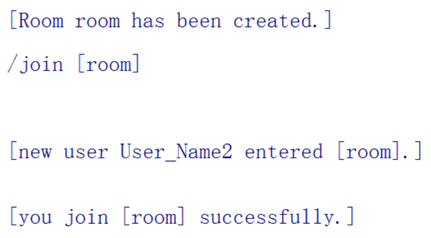


```
[you join [room] successfully.]
/delete [room]

[Error, you are not able to delete this room.]
```

Figure 21

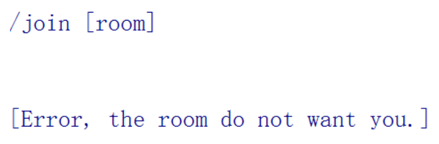
4. Join Chatroom:

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).



```
[Room room has been created.]
/join [room]

[new user User_Name2 entered [room].]

[you join [room] successfully.]
```

Figure 22

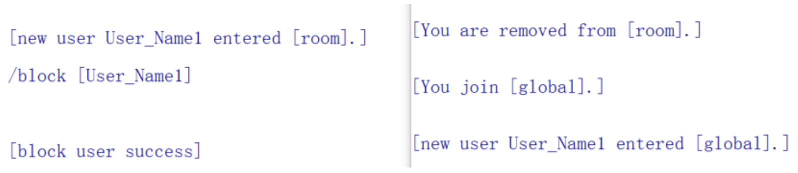


```
/join [room]

[Error, the room do not want you.]
```

Figure 23

5. Block User:

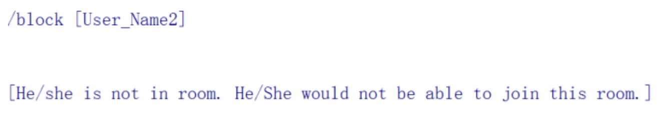
User types “/block [xxx]”. If a room creator types: “/block [xxx]”, the corresponding user xxx is removed from chatroom (figure 24) and/or denied to join this chatroom (figure 25). If other user (not the room creator) types this, that user gets error message (figure 26).



```
[new user User_Name1 entered [room].]
/block [User_Name1]

[block user success]
```

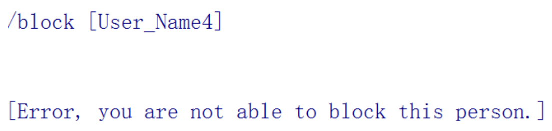
```
[You are removed from [room].]

[You join [global].]

[new user User_Name1 entered [global].]
```

Figure 24



```
/block [User_Name2]

[He/she is not in room. He/She would not be able to join this room.]
```

Figure 25

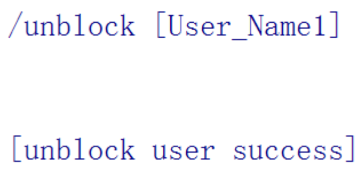


```
/block [User_Name4]

[Error, you are not able to block this person.]
```

Figure 26

6. Unblock User:

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

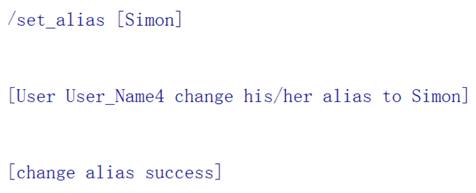


```
/unblock [User_Name1]

[unblock user success]
```

Figure 27

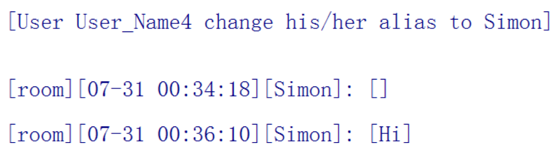
7. Change Alias:

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).



```
/set_alias [Simon]

[User User_Name4 change his/her alias to Simon]

[change alias success]
```

Figure 28



```
[User User_Name4 change his/her alias to Simon]

[room][07-31 00:34:18][Simon]: []

[room][07-31 00:36:10][Simon]: [Hi]
```

Figure 29

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]	Not met It is created in the design phase but found not necessary to have all instruction included in a message.

		2.1.4.5 /unblock [user_alias]	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.

2. **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.

3. **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.

4. **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	Class diagram	Zelan Xiang &

	textual		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		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.