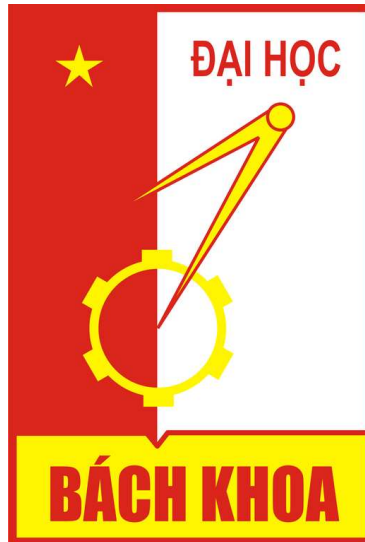


HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology



Network Programming Capstone Project Report

P2P CHAT APPLICATION

Group 13

Nguyen Duc Hung – 20176779

Nguyen Manh Khang – 20176792

Hanoi, June, 2021

Table of Contents

Introduction	2
P2P Chat Application	2
Network programming techniques	3
Architectural Design	4
UseCase	4
Basic flow of the program	5
Interaction Diagram	6
Data Modeling	9
Structure to manage user	9
Linked list struct for onlineLst and chatLst	10
Structure to manage chat message	10
Structure to exchange message over network	11
Message Exchange Model	12
Functions to send and receive message over the network	12
Serialize/Deserialize functions	12
Client & Server Design	13
Server Prototype	13
Client Prototype	13
GRAPHICAL USER INTERFACE	15
Login Screen	15
Register Screen	16
Online List Screen	17
Chat Screen	18
Login Fail Dialog	19
Request Chatting Dialog	21
Response Dialog	22
Register Success Dialog	23
Work Contributions	23

1. Introduction

P2P Chat Application

P2P chat application is a program that allows users to communicate with each other conveniently, privately and directly through the p2p protocol. This application supports functions such as creating an account, logging in, viewing a list of people who are online at the same time, and most prominently, the ability to communicate one-to-one directly with all people who are online.

Basically, a p2p chat application has two main components: a server and a client. The server, in this model, can also be called a tracker with the main role of maintaining and updating the list of users who are online, managing logins, registrations and managing user status (online, offline, lock).).

The client, which basically represents the user, in addition to using the features provided by the server, can select the online user from the list to do a one-on-one chat in a chat window individually. This one-to-one communication has absolutely no support from the server, but rather through the p2p protocol for clients to communicate directly with each other using IP and port information previously provided by the server.

About the communication between server and client, (visually shown in the figure below), every 2s, the client side will send HI message by UDP to the server, if after 8s the server does not receive any HI message from one specific user, the server will remove that user from the online list. Every 10 seconds, the user

will send a request to the tracker asking for an online list via TCP, and the server sends back an online list with info including (name, ip:port). When getting the ip port. of the user that needs to chat, user 1 will send a connection request to user 2 to proceed with the chat, user 2 can reject the request.

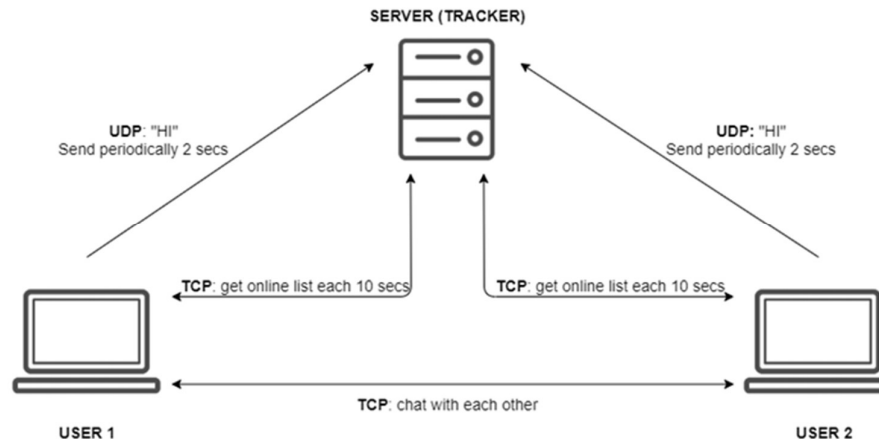


Figure 1.1: brief model of system

Network programming techniques

In this program we used the networking techniques listed below:

- TCP, UDP protocol
- Multithreading:
 - ✓ Used by peers to serve multiple other peers at a time (chat feature)
- Multiplexing:
 - ✓ Used in opening a listen socket in tracker and peers to receive and handle message
- Non-blocking IO:
 - ✓ Used when receive request message, then the peer is not blocked by waiting for receiving message from

other peers. In addition, we can receive Cancel operation from user and receiving response message at the same time

2. Architectural Design

UseCase

First of all, the following is the Use Case diagram of our P2P chat application:

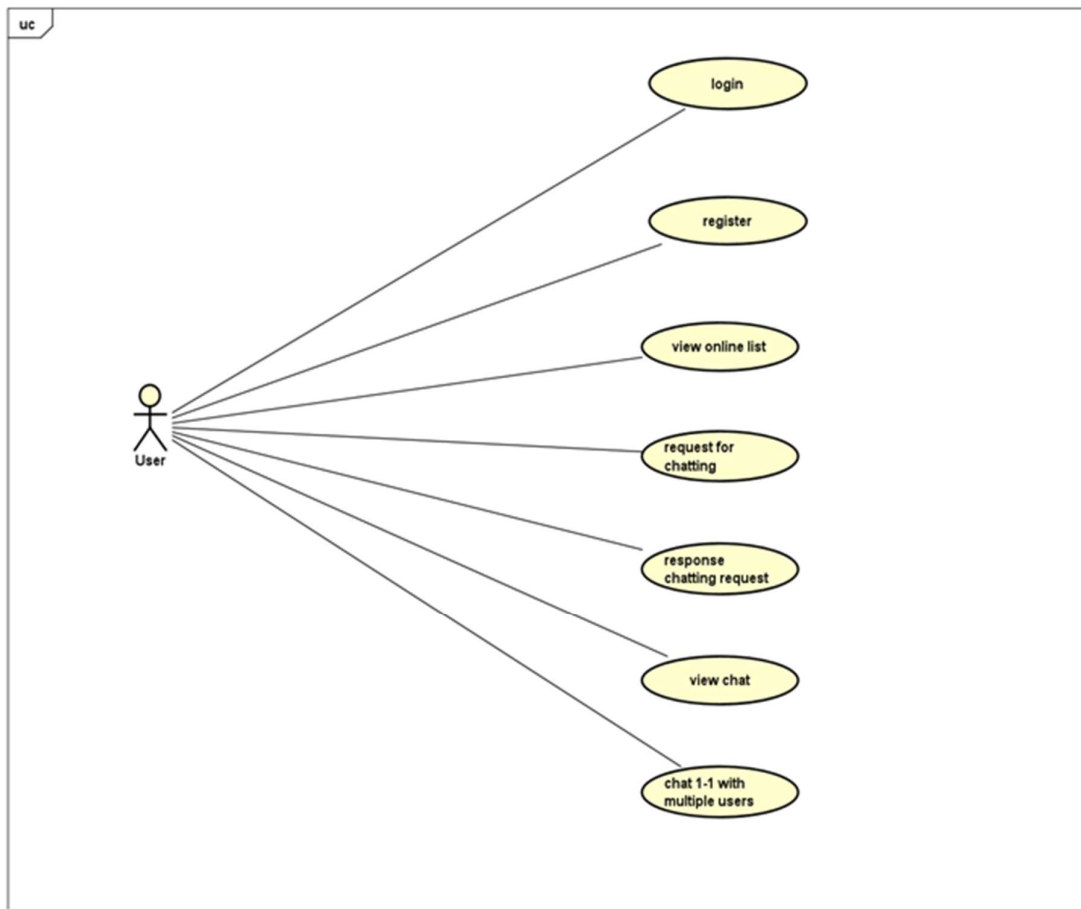


Figure 2.1. Use case Diagram

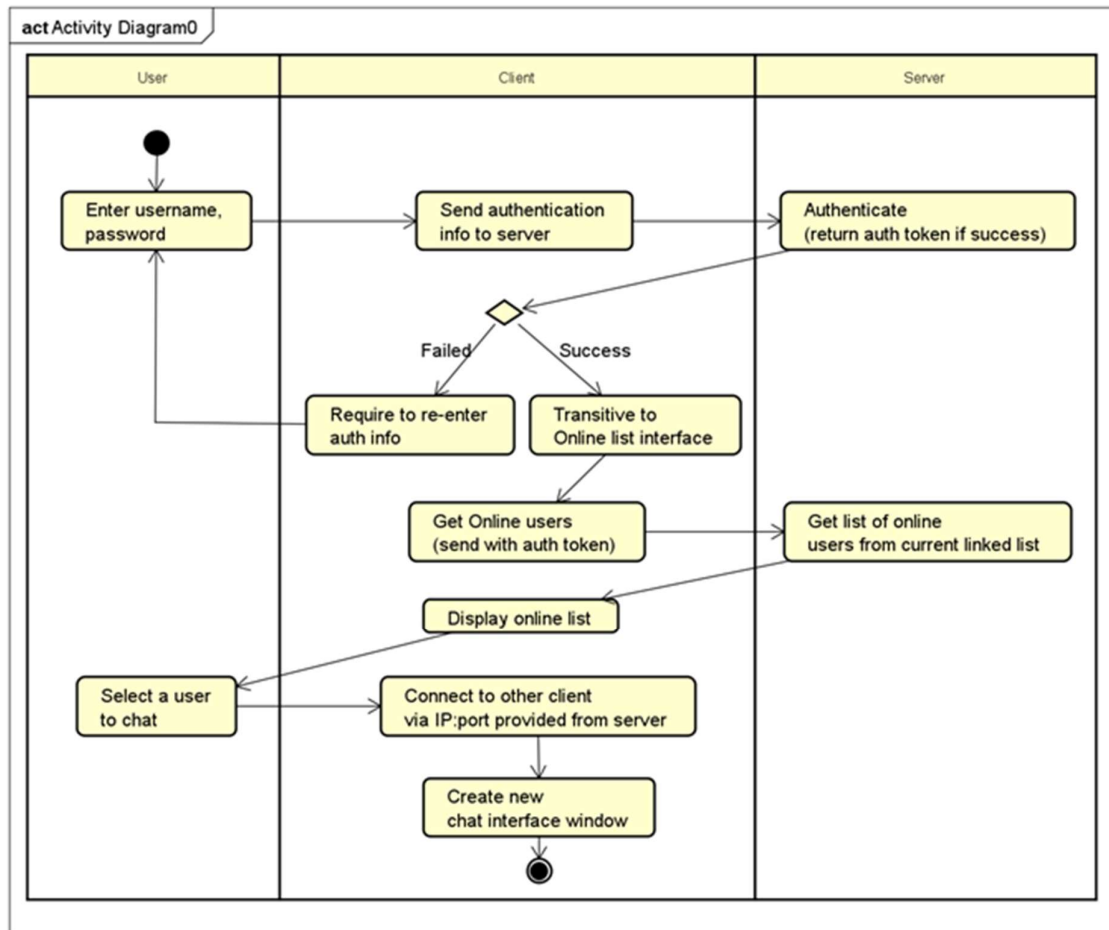
Based on the diagram, the main actor user can perform the following use cases: register, login, view online list, request for chatting, response request for chatting, view chat, chat 1-1 with multiple users

Basic flow of the program

The basic flow of our application is shown as the figure below. In this diagram, we focus on the main flow of the program and do not care too much about the alternative flow.

From the start, the user will enter the username and password, and the client will send this authentication information to the server. Server will authenticate it, if the information is correct, the server will return a certificate token and send it back to the client. When the client receives the order, it will transitive to the next screen, which is the Online list screen and also request the server for the list of online users. When the screen is displayed, the user can see other online users and can select 1 to chat. After that, the client will send the request to the targeted user (via IP:port provided before by the server) and show the dialog that the application is sending the request to this targeted user and waiting for his/her response. When everything is done, the application will create a new chat window for the user to chat 1-1 with the targeted user.

Figure 2.2. Activity Diagram



Interaction Diagram

To have a better look into the dynamic architectural design of the software, we come up with interaction diagrams of the application:

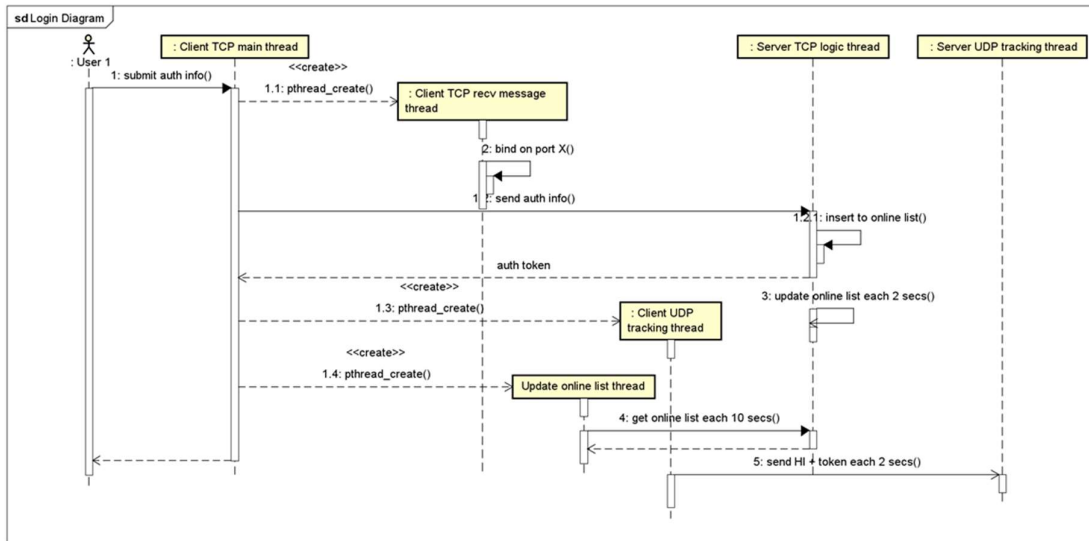
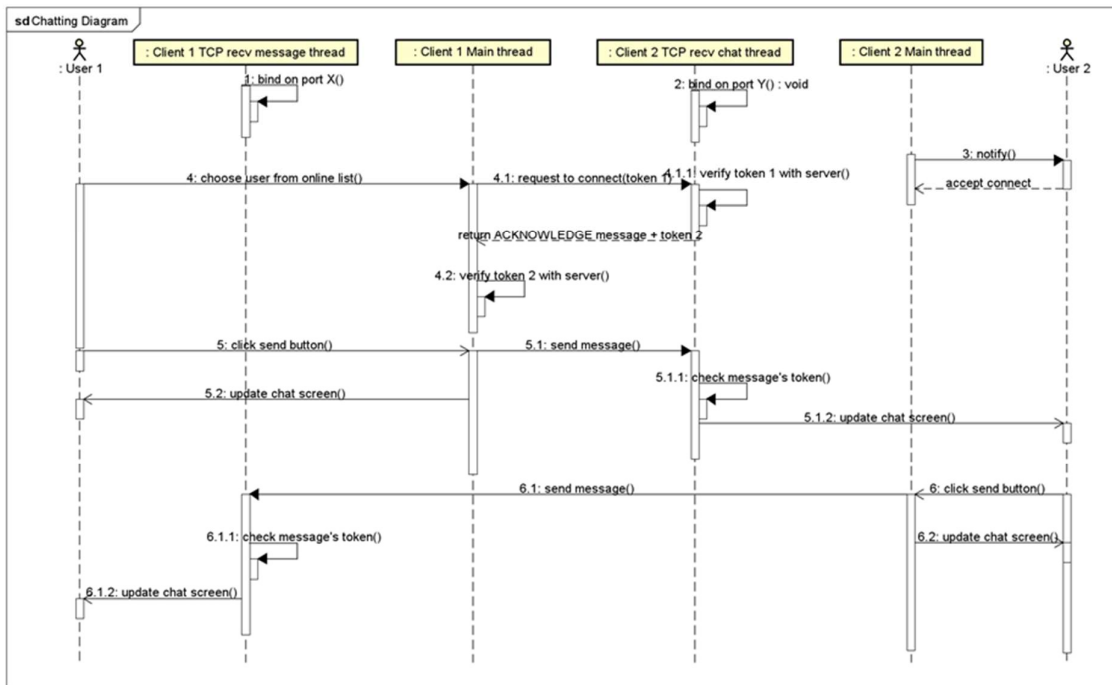


Figure 2.2. Login interaction diagram

In this part of the diagram, first of all, the user will submit the login information to the Client TCP main thread, this thread will create a Client TCP revc message thread for binding this user to 1 available port, and also send the information to the Server TCP logic thread. The Server TCP logic thread will authenticate the information, add this user to the list of online users, and send back certificate token to the Client TCP main thread. When receiving the token, the Client TCP main thread will create 2 more threads, which are Client UDP tracking thread and Update Online List thread for sending Hi message and token to the server every 2s and getting the online list periodically 10s, respectively.

Figure 2.3. Chatting interaction diagram



In this diagram, we can see the sequence of chatting flow after successfully login. We have 2 users showing up in this diagram. For instance, user1 will request user2 for chatting. After logging in, 2 users are bonded to 2 available ports X and Y (this part is shown before in the login diagram and done by Client TCP recv message thread). When user1 select user 2 to connect, the Client 1 main thread will create a request message contains user1's certificate token, this message is sent to Client 2 TCP recv message thread, in this thread, user1's token will be verified, if success, Client 2 main thread will notify and ask for answer from user 2 (pop up 1 dialog). If the request is accepted, Client 2 main thread will send the response and user2's token to Client 1 main thread. After that, 2 users will redirect to a new separate window for 1-1 chatting. For chatting, when 1 user clicks the send button, this Client Main thread will send the chat message to the other user Client TCP recv message thread, and will update the screen in both 2 users.

3. Data Modeling

Structure to manage user

- usr : username
- pwd : password
- token : for authentication
- usrAddr : address of peer
- status : account status (see below)
- logCnt : login attempt count
- offCnt : offline interval time count

The code for the struct user will be shown below:

```
// Account status  
#define INACTIVE -1  
#define LOCKED 0  
#define ACTIVE 1
```

```
typedef struct {  
    char usr[USRLEN];  
    char pwd[PWDLEN];  
    char token[TOKENLEN+1];  
    struct sockaddr_in usrAddr;  
    char status;  
    char logCnt;  
    char offCnt;  
} account_t;
```

Linked list struct for onlineLst and chatLst

```
typedef struct Node {
    account_t* acc;
    struct Node* next;
} accNode_t;

accNode_t* createNode (account_t* acc);
accNode_t* createLinkedList ();
accNode_t* insertNode (accNode_t* head, accNode_t* inode);
accNode_t* searchAccount (accNode_t* head, const char* usr);
void freeNode (accNode_t* p);
void freeLinkedList (accNode_t* head);
int removeAccount (accNode_t* head, const char* usr);
int overrideAhead (accNode_t* curr, const accNode_t* nextNode);
```

```
accNode_t* onlineLst;
accNode_t* chatLst;
```

Structure to manage chat message

- from : username of sender
- token : token of sender
- to : username of receiver
- txt : chat message
- at : send time

```
typedef struct {  
    char from[USRLEN];  
    char token[TOKENLEN+1];  
    char to[USRLEN];  
    char txt[MSGLEN];  
    time_t at;  
} chat_msg_t;
```

Structure to exchange message over network

- type : message type
- len : payload length
- payload : message content

```
typedef struct {  
    char type;  
    unsigned short len;  
    char payload[PAYLOADSIZE];  
} net_msg_t;
```

4. Message Exchange Model

Functions to send and receive message over the network

- sendMsg : caller prepares net_msg_t message to send through sockfd
- recvMsg : caller have to allocate space for storing network message read in sockfd

```
int sendMsg (int sockfd, const net_msg_t* msg);  
  
int recvMsg (int sockfd, net_msg_t* msg);
```

Figure 4.1. sendMsg and recvMsg function

Serialize/Deserialize functions

- We need to turn an chat message structure into a buffer to store in net_msg_t.payload, and a network message structure into a memory buffer for easily exchange over network -> solution: serialize at sender, deserialize at receiver
- Serialize : convert message structure (which is organized by C structure standard) into a formatted memory buffer (which can send through over network without worrying about memory padding)
- Deserialize : convert a formatted memory buffer into allocated data structure

```
int serializeNetMsg (char* msgBuff, const net_msg_t* msg);  
  
net_msg_t* deserializeNetMsg (net_msg_t* msg, char* msgBuff);
```

```
int serializeChatMsg (char* msgBuff, const chat_msg_t* msg);  
chat_msg_t* deserializeChatMsg (chat_msg_t* msg, char* msgBuff, int msgLen);
```

5. Client & Server Design

Server Prototype

- startServer: bind and listen to a port
- processLogic: receive and handle message (authenticate message, get online list message)
- maintainOnlineLst: run on a separate thread to manage online list each time interval
- hiMsgTracking: run on a separate thread to receive HI message each time interval

```
int processLogic (int connfd) { ...  
  
int startServer () { ...  
  
void* maintainOnlineLst (void* none) { ...  
  
void* hiMsgTracking (void* none) { ...
```

Client Prototype

- sendHiMsg: run on a separate thread to send HI msg each time interval
- updateOnlineLst: run on a separate thread to retrieve online list each time interval
- sendChatMsg: run on a separate thread to send chat message
- clientMsgHandler: run on separate thread to receive control message

- clientMsgReceiver: run on the same thread with clientMsgHandler (called by clientMsgHandler to read and handle control message)
 - connectChat: run on a separate thread to request connect to other client
 - showOnlineList: run on main thread to show online list (called by loginSuccess)
- loginSuccess: run on main thread to prepare things when login success (called by login)
- Login: run on main thread to do authentication

```

void* sendHIMsg (void* none) { ...

void* updateOnlineLst (void* none) { ...

void* sendChatMsg (void* accVoid) { ...

int clientMsgReceiver (int sockfd) { ...

void* clientMsgHandler (void* none) { ...

void* connectChat (void* accVoid) { ...

void showOnlineList () { ...

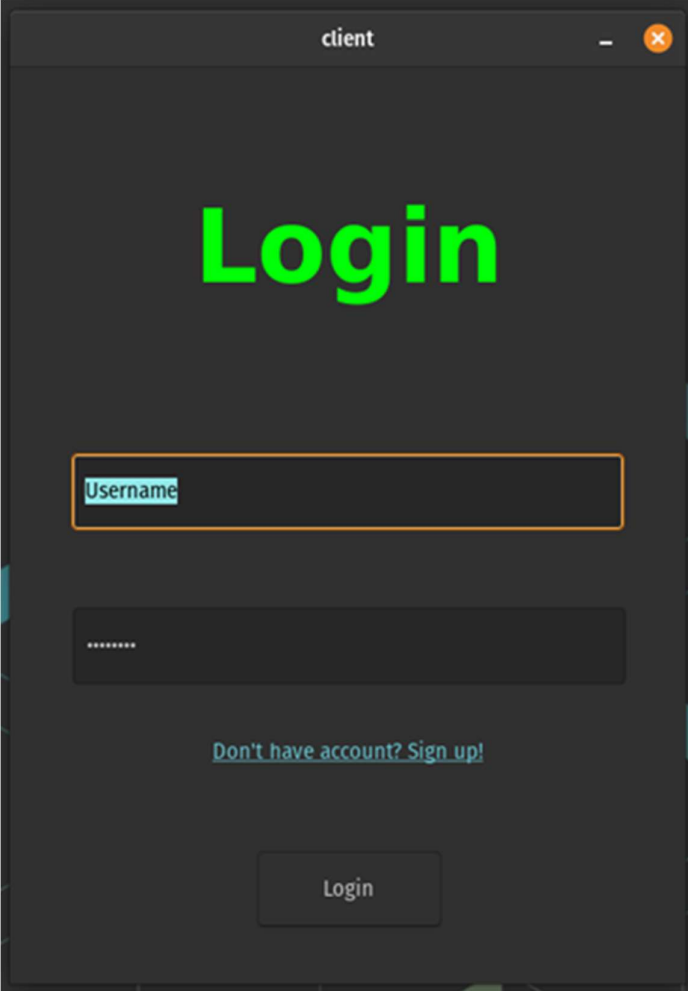
void loginSuccess (int sockfd) { ...

void login () { ...

```

6. GRAPHICAL USER INTERFACE

Login Screen



A mockup of a login screen within a window titled "client". The window has a dark gray background. At the top, the title bar shows "client" and standard window controls (minimize, maximize, close). The main content area features the word "Login" in a large, bold, red font. Below this, there are two input fields: the first is labeled "Username" in a light blue font and has a light blue border; the second is a password field with a dark gray background and a light gray border, containing a series of dots. Below the password field, there is a link that says "Don't have account? Sign up!" in a light blue font. At the bottom, there is a "Login" button with a light gray border and a light gray background.

client

Login

Username

.....

[Don't have account? Sign up!](#)

Login

Register Screen

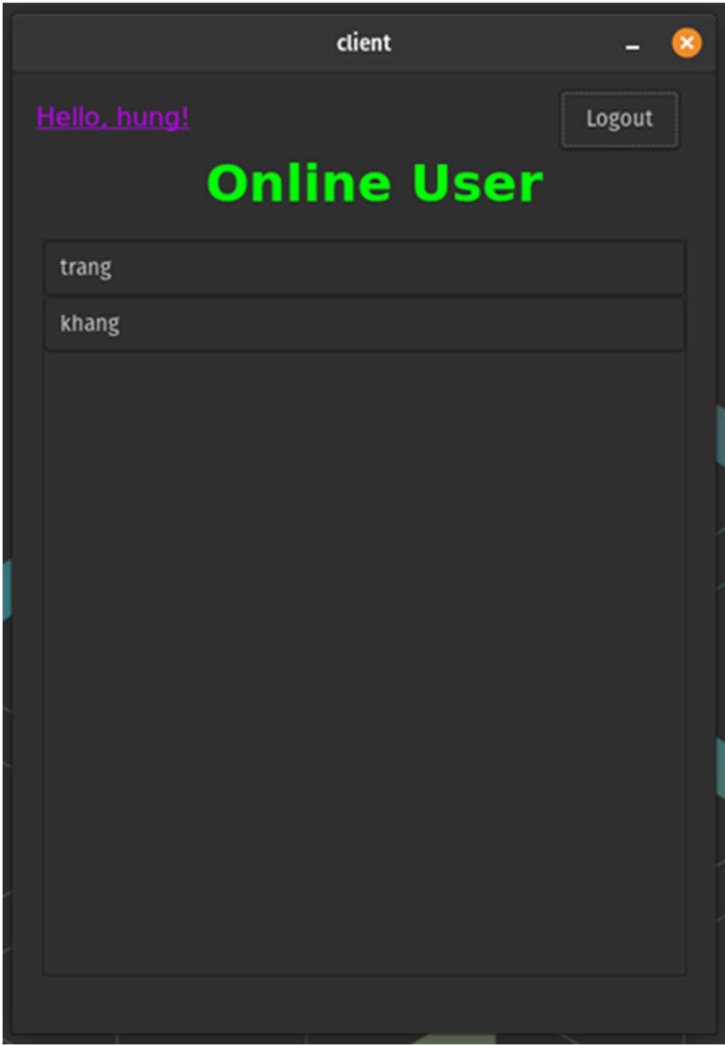
client

Sign Up

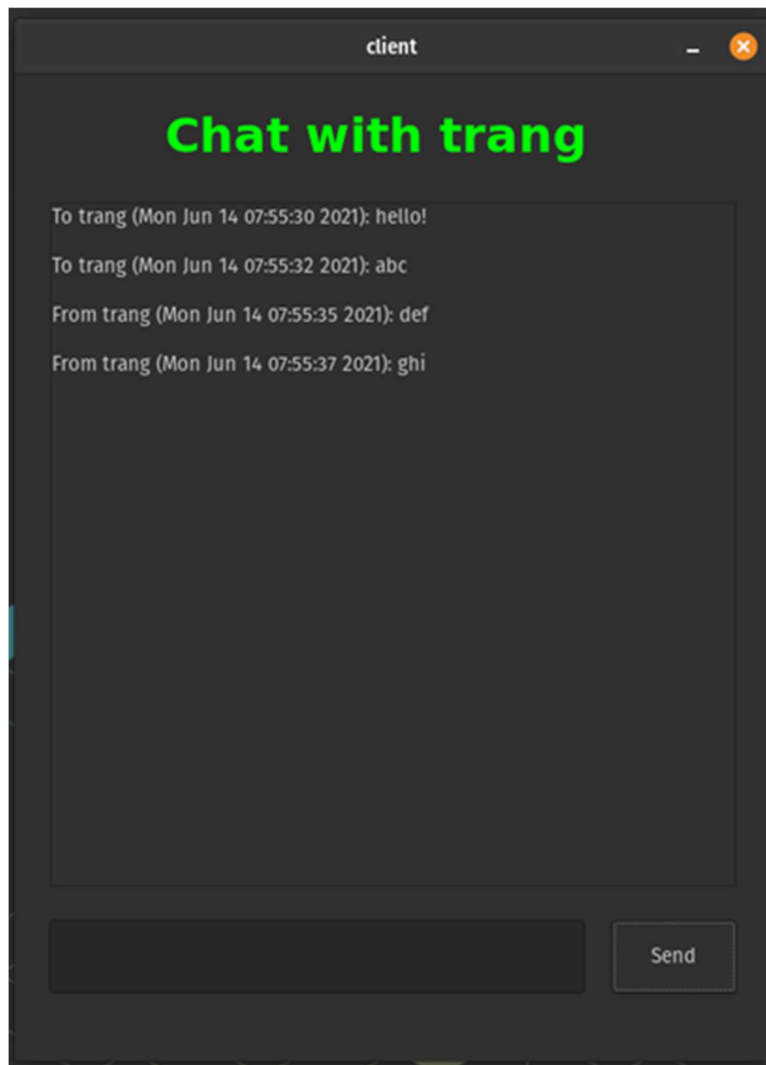
Username

Sign up

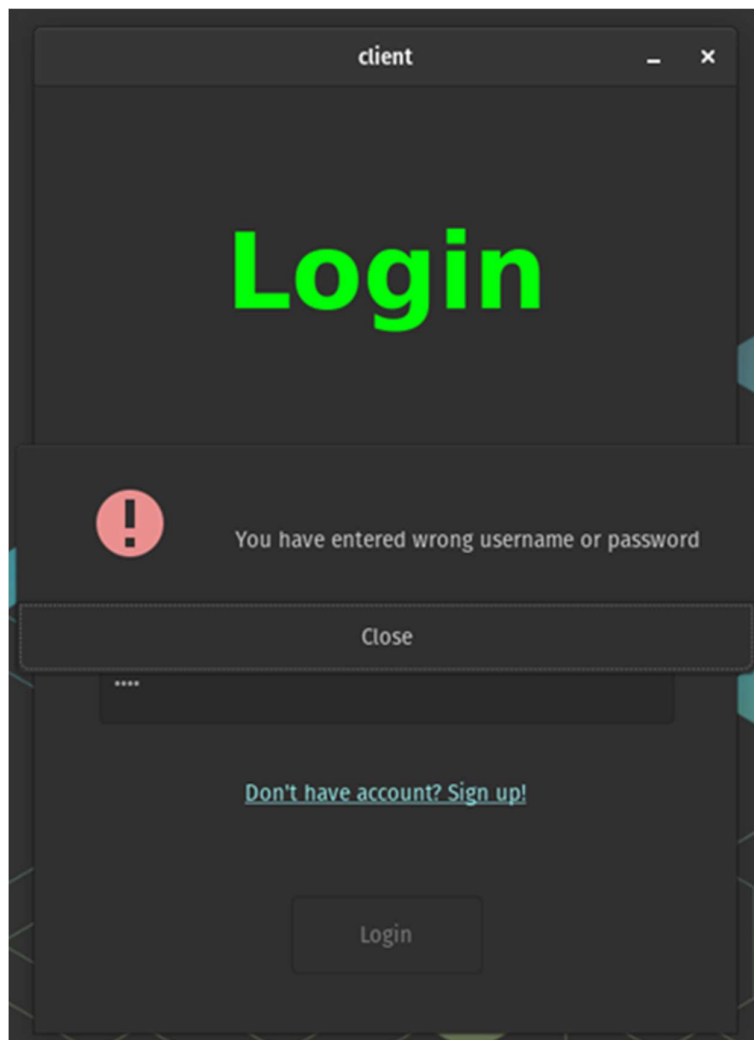
Online List Screen

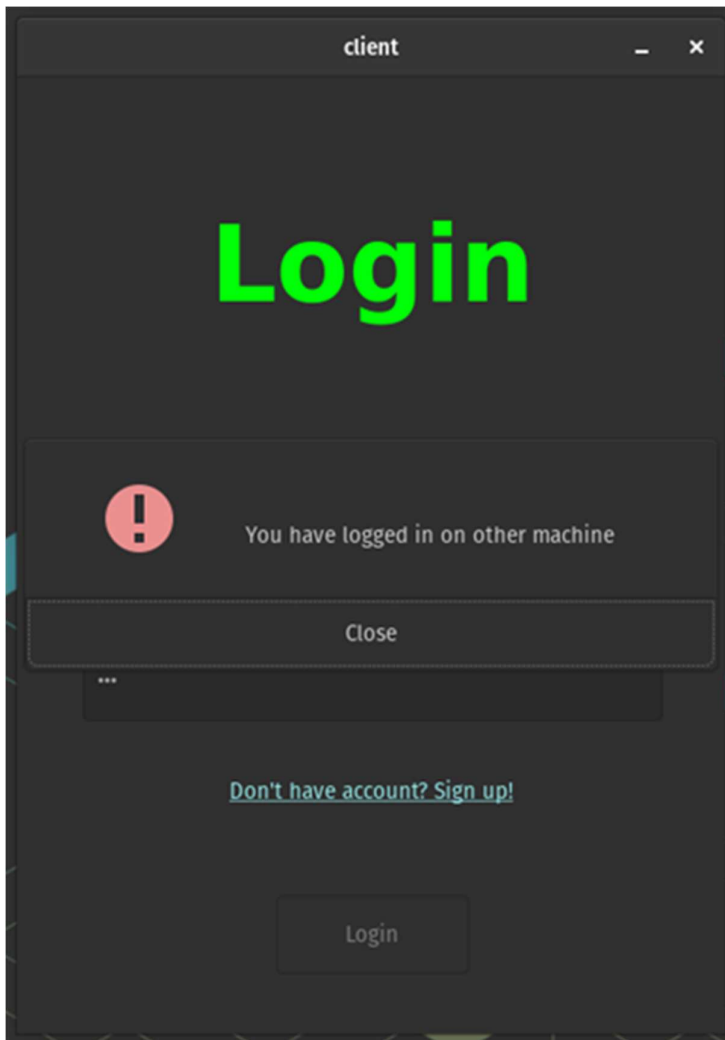


Chat Screen

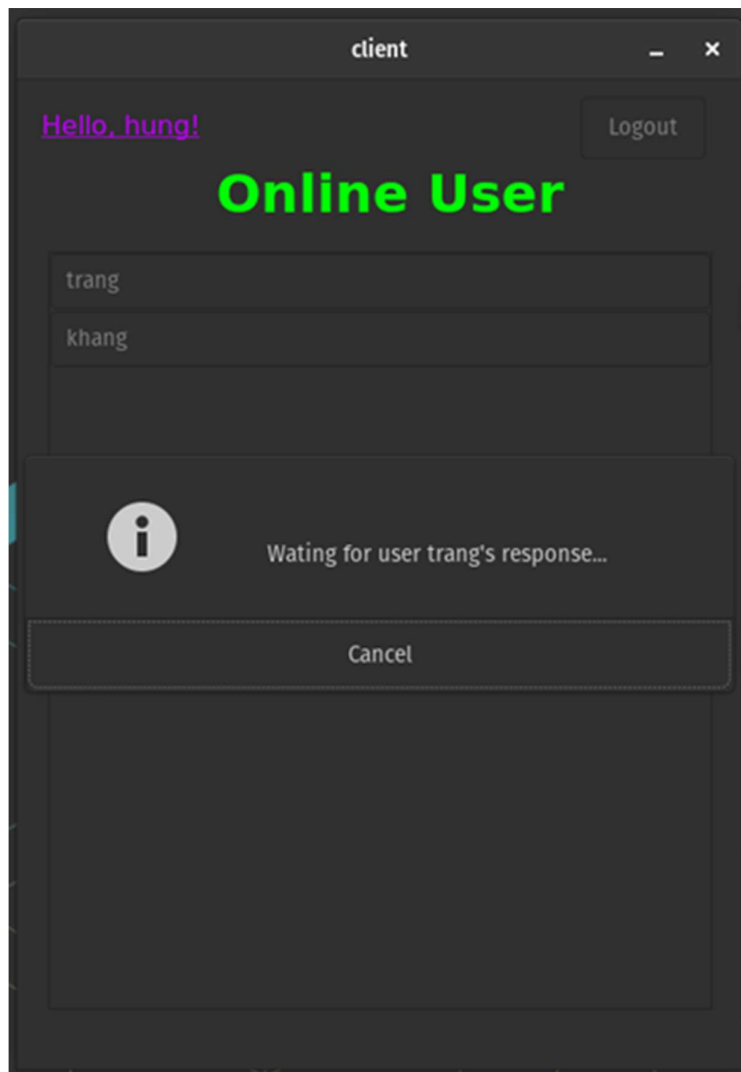


Login Fail Dialog

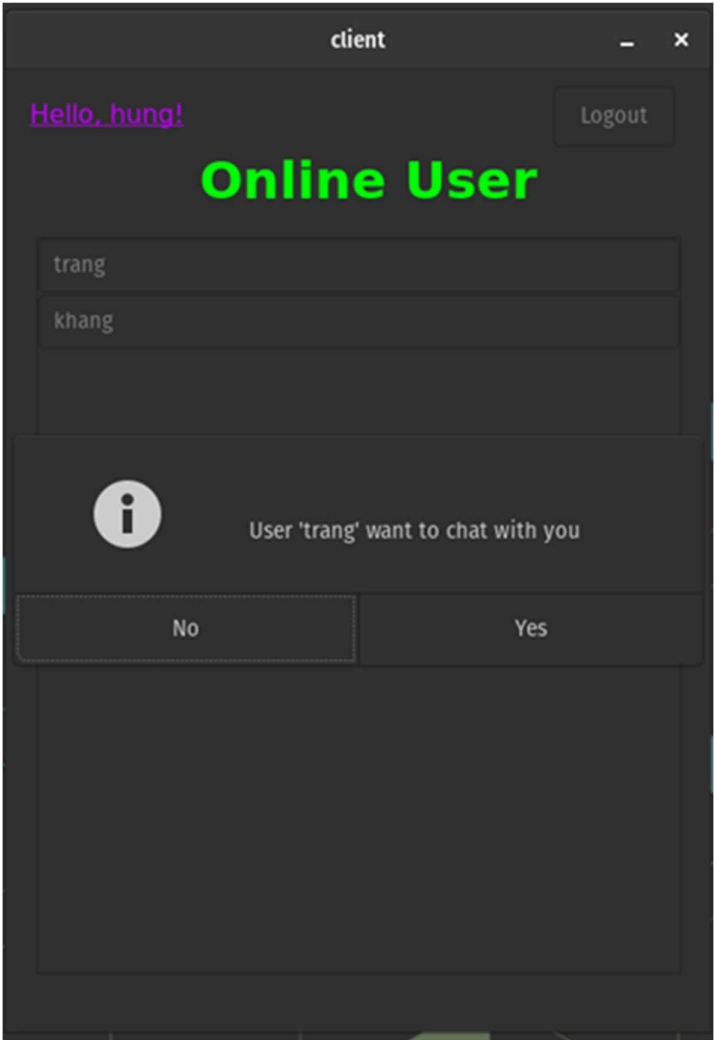




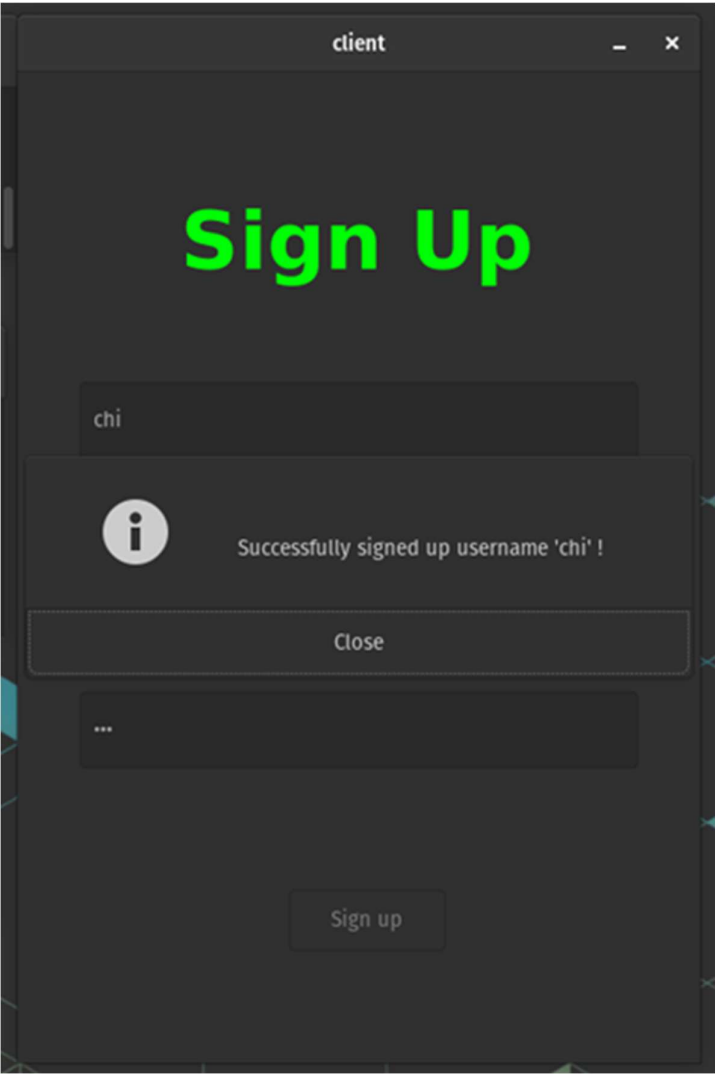
Request Chatting Dialog



Response Dialog



Register Success Dialog



7. Work Contributions

Work	Code Base	Interface	Presentation, Report	Architectural Design
Hung	60%	40%	30%	70%
Khang	40%	60%	70%	30%