Introduction

The chat application comprises two main components: the server-side program and the client-side program. These components work together to enable users to communicate with each other in real-time over a network, and in this report, we explain the design and functionality of the application.

Design and Functionality

The server program is designed to handle multiple client connections concurrently using threading. It initializes a socket and binds it to a specific port (5555). Upon receiving a connection request from a client, it spawns a new thread to handle that client's communication independently. For functionality, the server maintains a list of connected clients and their states (such as visibility and blocked status), server allows users to register with unique usernames and passwords or log in if they already have accounts, it facilitates communication between clients by relaying messages and managing chat requests, and clients can perform various actions, such as changing visibility, listing active clients, starting chats, and blocking/unblocking users.

The client program connects to the server using a socket and communicates using encoded messages. Threading is utilized to handle both message reception and user input concurrently, ensuring smooth interaction. For functionality, clients can register with unique usernames and passwords or log in if they already have accounts. User authentication is handled by the server. Clients can initiate and participate in chats with other users. The chat functionality supports sending text messages and files using UDP sockets. The client provides a simple command-line interface for user interaction, allowing users to choose options from a menu. Clients can view a list of active clients who are available for communication and clients can gracefully disconnect from the server when done.

Features

User Registration/Login: Users can create accounts with distinctive usernames and passwords thanks to the user registration and login features. This feature guarantees that users can safely access the chat program and provide the server with their identity.

Real-time Chat Communication: The primary purpose of the chat program is to facilitate user-to-user real-time chat communication. Users can rapidly share files and text messages using this tool, which promotes efficient communication.

Client Management: The server's ability to manage clients on the server side guarantees that it can manage several client connections at once. It makes it possible for the server to keep track of all connected clients' states, including visibility and blocked status.

User Visibility Status: Allowing users to change their visibility status (public or private) determines whether other users can see them in the active clients list. This feature gives users control over their online presence and privacy.

Active Clients List: Users can see a list of other users who are online and available for communication by using the active clients list feature. This makes it easier for users to find someone to communicate with in real time.

Blocking/Unblocking Users: Allowing users to block/unblock other users prevents unwanted communication from specific users. This feature enhances user control over their chat experience and privacy.

File Transfer: When file transfer is supported during chat sessions, users can share files in addition to text messages, including documents, photos, and more. The chat application's adaptability and utility are increased by this functionality.

Graceful Disconnect: Offering a graceful disconnect feature enables users to quietly end their time spent using the chat program. This guarantees that resources are released in a correct manner and that there are no problems when the user's session ends.

Error Handling: Implementing robust error handling mechanisms ensures that the application can gracefully handle unexpected errors or exceptions. This feature enhances the stability and reliability of the chat application.

Security Measures: Sensitive user data is protected, and secure communication channels are ensured by using security measures, such as encryption for client-server communication. The protection of user privacy and data integrity depends on this functionality.

Protocol Specification

The chat application protocol specifies the format and structure of messages exchanged between the client and server for communication and interaction. For the message structure, each message consists of a request type which specifies the type of request or action to be performed, and arguments which are additional data or parameters required for the request. All messages are encoded in DICTIONARY format for easy parsing and communication between the client and server.

Message Format:

User Registration:

Request: "register"

Arguments

"name": The username that the user choose while registering.

"password": The password that the user selected when registering.

User Sign-in:

Request: "login"

Arguments

"name": The login username that the user has supplied.

"password": The user-provided password used to log in.

Modify Visibility:

Request: "visibility"

Arguments: "arg": The desired visibility status (such as "public" or "private").

Enumerate Current Clients:

Request: "Active_Clients"

Arguments: null

Request to Start a Chat:

Request: "chat_request"

Arguments: "arg": The recipient user's username for the chat request.

Block User:

Request: "block_user"

Arguments: "arg": Username of the user to be blocked.

Unblock the user:

Request: "unblock_user"

Arguments: "arg": Username of the user to be unblocked.

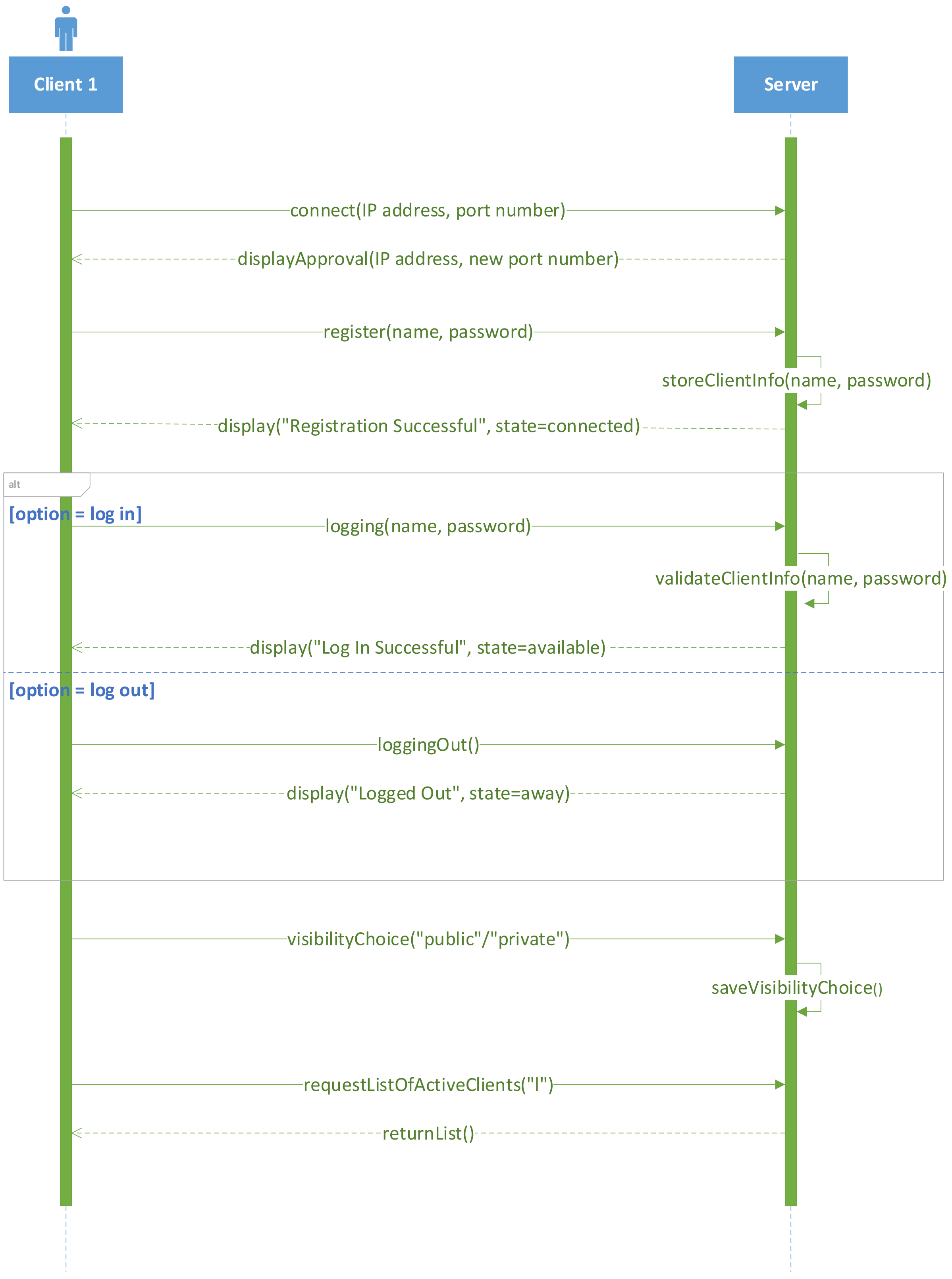
Logout/Disconnect:

Request: "exit"

Arguments: null

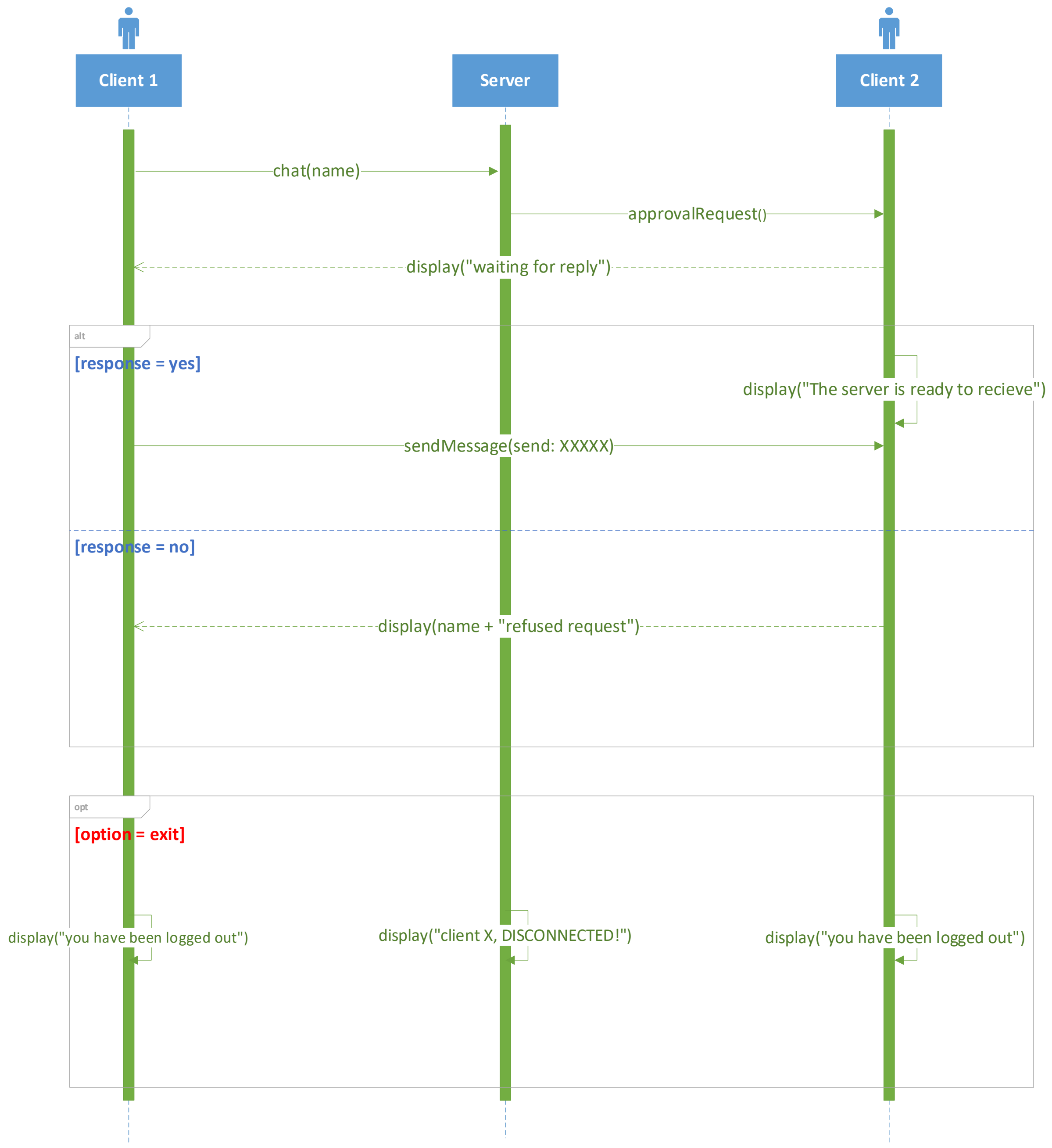
Client-server communication

The sequence diagram below shows the communication- requests and return messages between the client and the server. The server helps to store all the information and preferences of the client. The server also keeps track of all the available clients.


Client 1
Server
connect(IP address, port number)
displayApproval(IP address, new port number)
register(name, password)
storeClientInfo(name, password)
display("Registration Successful", state=connected)
alt
[option = log in]
logging(name, password)
validateClientInfo(name, password)
display("Log In Successful", state=available)
[option = log out]
loggingOut()
display("Logged Out", state=away)
visibilityChoice("public"/"private")
saveVisibilityChoice()
requestListOfActiveClients("l")
returnList()

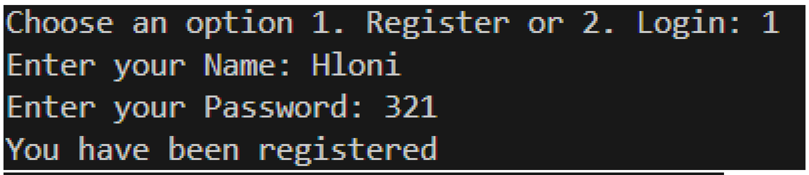
Client-client communication

The sequence diagram below shows the steps that it takes or protocol to ensure that there is efficient communication between the clients. It also shows an optional choice for clients to exit the chat.

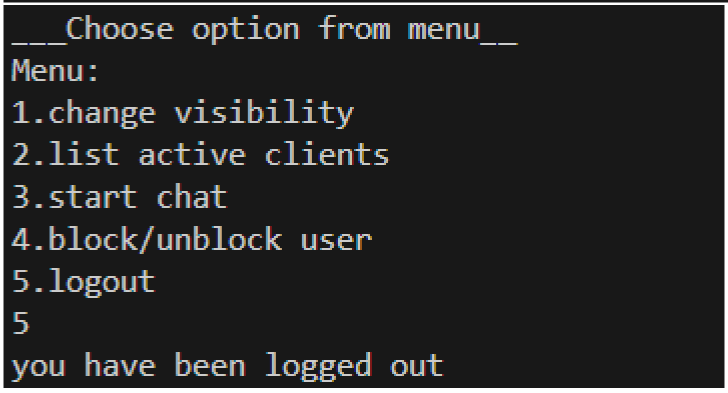

Client 1
Server
Client 2
chat(name)
approvalRequest()
display("waiting for reply")
alt
[response = yes]
display("The server is ready to recieve")
sendMessage(send: XXXXX)
[response = no]
display(name + "refused request")
opt
[option = exit]
display("you have been logged out")
display("client X, DISCONNECTED!")
display("you have been logged out")

<u>Interface of Application Features</u>

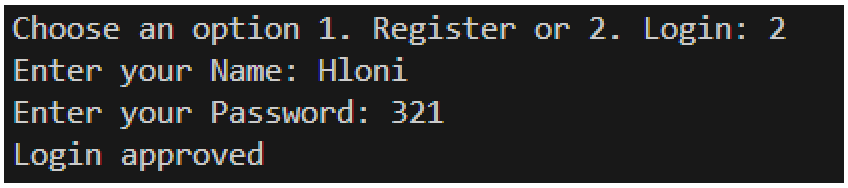
Register/Login:



```
Choose an option 1. Register or 2. Login: 1
Enter your Name: Hloni
Enter your Password: 321
You have been registered
```

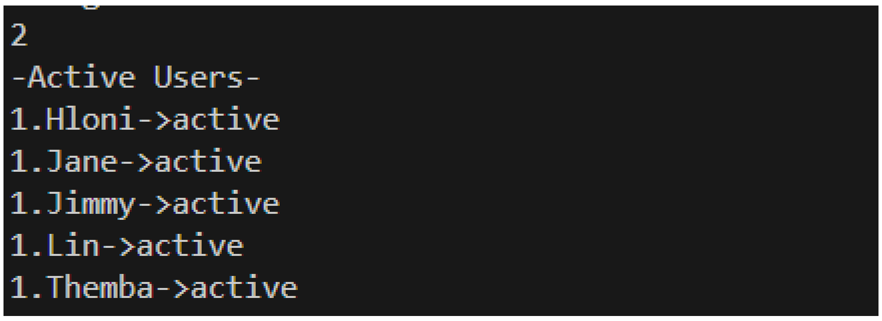


```
___Choose option from menu__
Menu:
1.change visibility
2.list active clients
3.start chat
4.block/unblock user
5.logout
5
you have been logged out
```



```
Choose an option 1. Register or 2. Login: 2
Enter your Name: Hloni
Enter your Password: 321
Login approved
```

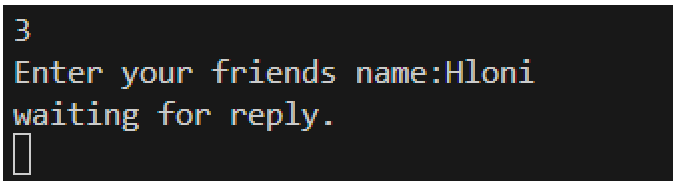
Active Clients List:



```
2
-Active Users-
1.Hloni->active
1.Jane->active
1.Jimmy->active
1.Lin->active
1.Themba->active
```

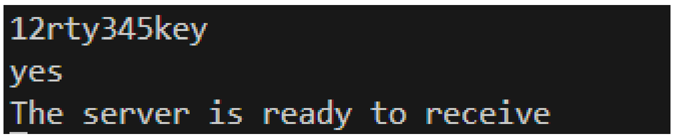
Real-time Chat Communication:

Client 1(initiates communication)

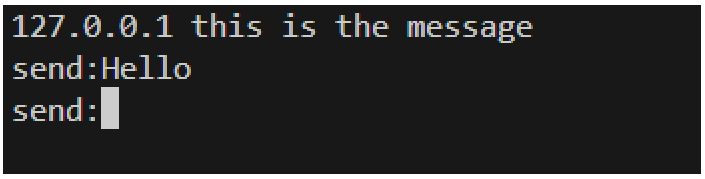


```
3
Enter your friends name:Hloni
waiting for reply.
```

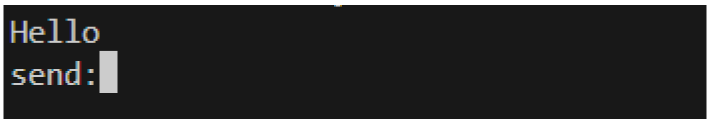
Client 2(approves request)


```
12rty345key
yes
The server is ready to receive
```

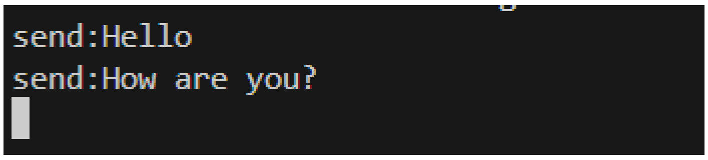
Client 1(receives approval from Client 2 and sends message)


```
127.0.0.1 this is the message
send:Hello
send:
```

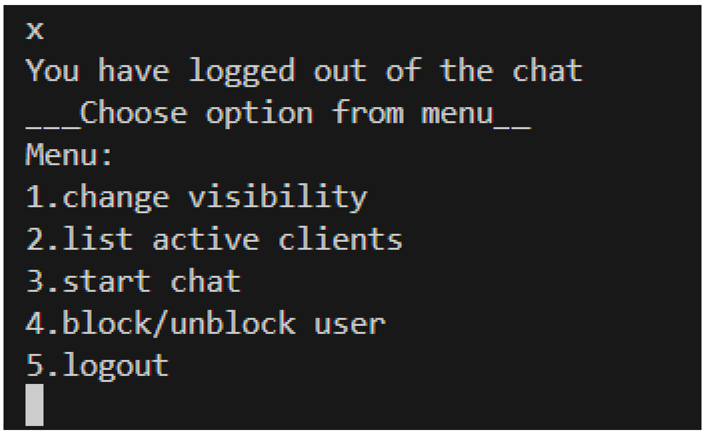
Client 2(receives message and replies)


```
Hello
send:
```

Client 1(receives reply message)


```
send:Hello
send:How are you?
```

Clients can input "x" to log out of the chat


```
x
You have logged out of the chat
___Choose option from menu__
Menu:
1.change visibility
2.list active clients
3.start chat
4.block/unblock user
5.logout
```