Server:

UserIDArray: Stores the users that are connected to the communications server
pwArray: Stores the passwords that are used by the server
ss: the server socket that will be listening on a port number for incoming connections
s : the client socket that will be the connection from the client to be passed to the client handler
activeClients: a counter for the amount of clients that are active on the server currently.
ObjectOutputStream: the tunnel that will handle the output object from a user to the server to the recipient
ObjectInputStream: the tunnel that will handle the message object that will be taken when a user sends a message.

Authenticate(String userID, String Password): Server will authenticate the credentials given by the user during the login process. Takes the userID and password entered by user and compares to the correct credentials that are saved in a text file for users

sendMessage(ArrayList<User> users, Message msg): Server will receive a request for a message to be sent. The server will send the message to the recipient with the sender's id and the timestamp of when it was sent.
Main(): The part of the class that will handle the running of the communication application, it will establish a server socket listening on a port number, listen for incoming connections from clients, and create a client handle and create a thread to handle the client. Using calls such as: new clientHandler(port, objectInputStream, outputStream) and new Thread()

Use Cases for Server:

Use Case ID: 00
Use Case Name: Server Authenticates user
Relevant Requirements: SRS
Primary Actor: Server
Pre-conditions: User must have entered a userID and Password, server must be started,
Post-conditions: User will either be accepted to login or rejected and user gains access to comms app
Basic Flow or Main Scenario:

1. User enters userID and password.
2. Server compares user and password to saved username and password
3. User is verified and logged into communications application
4. User can use communications application

Extensions or Alternate Flows:

1. User enters userID and password
2. Server compares credentials to saved credentials
3. User is not verified
4. User is not able to access the communications application

Exceptions: Incorrect Password, Incorrect Username, Server is down
Related Use Cases:

Use Case ID: 01
Use Case Name: Server sends message from sender to recipient
Relevant Requirements: SRS
Primary Actor: Server
Pre-conditions: A user must have sent a message to another user
Post-conditions: Recipient user will receive message from sender
Basic Flow or Main Scenario:

1. Server receives request for a message to be sent
2. Server finds recipient from list
3. Server sends message to recipient client
4. Message is delivered to the recipient and awaiting the recipient to view.

Extensions or Alternate Flows:
Exceptions: Server is down
Related Use Cases:

ClientHandler:

Clientsocket: the socket used to keep the connection of the server
In: Handles the receiving end of objects, in this case the object will be of message class
Out: handles to sending of objects, in this case the object will be of message class

ClientHandler(clientsocket, in, out) //Constructor: creates an instance of a clienthandler, takes the clientsocket, the input and output streams of the socket.
Run(): Allows the server to process the sending and receiving of messages while keeping the server port free to listen for other incoming connections.

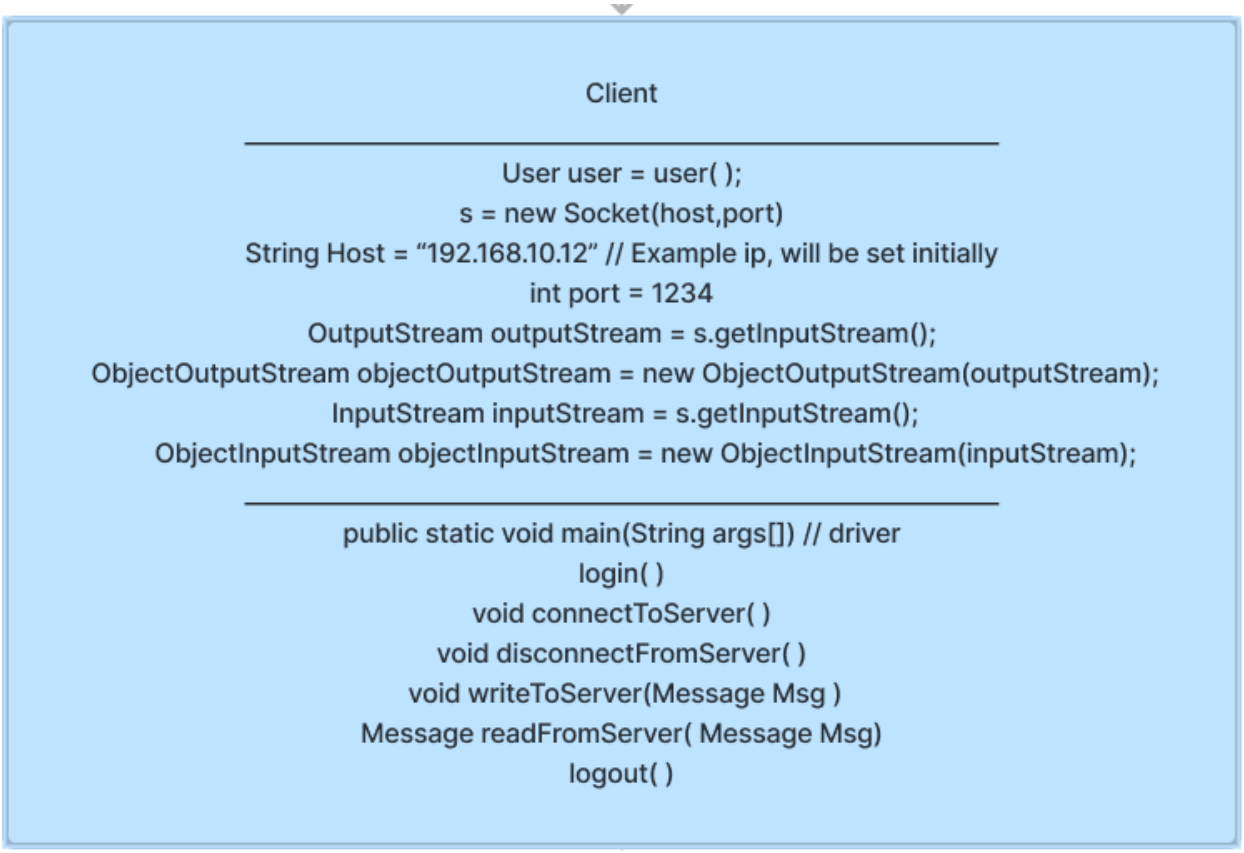
Use Cases for ClientHandler:

Use Case ID: 02
Use Case Name: Server creates a thread and client handler
Relevant Requirements: UML Class Method
Primary Actor: Server
Pre-conditions: A connection must be requested and serversocket accepts,
Post-conditions: thread will be made for handling messages
Basic Flow or Main Scenario:

1. Connection is requested
2. Server creates an instance of a clienthandler with given information
3. Thread is used for handling the client requests and sends

Extensions or Alternate Flows:
Exceptions: Server is down
Related Use Cases:

Client:



s: Creates the socket that will send the request to the server socket to establish the connection

Host: the ip address that will be used to connect to the server for the communication application
Port: the port that will be used to to connect the the server
ObjectOutputStream: the tunnel that will handle the output object from a user to the server to the recipient
ObjectInputStream: the tunnel that will handle the message object that will be taken when a user sends a message.
User: An instance of the user so that the client can get the information of a user as well as use the public methods of the user class.

main( ): The driver that will connect create and send a request to the server for a connection using methods below
login(String userID, string userPassword) - the client initiates an authentication attempt with the server.
connectToServer( ): allow for the connection to the server, by entering the correct host and port
disconnectFromServer( ): the method for to end a client's connection to the server
writeToServer(Message Msg): the method for a client to handle a user's sent messages by using the ObjectInputStream
readFromServer(Message Msg): the method for a client to receive messages from the server which were sent by other clients.
void logOut( ) - logs the user out of the system by disconnecting their client from the server. Switches boolean userIsOnline from True to False, which will cause the server to save future messages and wait for connection.

Use Cases for Client:

Use Case ID: 03
Use Case Name: Client connects to the server
Relevant Requirements: UML Class Method
Primary Actor: Client Driver
Pre-conditions: User launches the client
Post-conditions: user will be on the client, and client will be connected to the server
Basic Flow or Main Scenario:

1. Client uses host ip and port number to connect to the server
2. Input and output tunnels are established
3. Client is connected to the server
4. Messages are received if any were sent.

Extensions or Alternate Flows:
Exceptions: Server is down
Related Use Cases:

Use Case ID: 04
Use Case Name: Client sends message through server to another client
Relevant Requirements: UML Class Method
Primary Actor: Client Driver
Pre-conditions: User requests to send a message to another user and client server connection is established
Post-conditions: message is sent from user to recipient
Basic Flow or Main Scenario:

1. User types the message that they want to send
2. Client sends the message through the output stream to server
3. Server receives message and sends it off to the recipient

Extensions or Alternate Flows:
Exceptions: Server is down, user connection issue
Related Use Cases:

Use Case ID: 05
Use Case Name: Client reads message from server that is from another client
Relevant Requirements: UML Class Method
Primary Actor: Client Driver
Pre-conditions: client server connection is established and message was sent to user
Post-conditions: user will be on the client, and client will be connected to the server
Basic Flow or Main Scenario:

1. Client receives message from server through input stream
2. Client displays message to the user in the chat

Extensions or Alternate Flows:
Exceptions: Server is down, user connection issue
Related Use Cases:

Use Case ID: 06
Use Case Name: Client disconnects from server
Relevant Requirements: UML Class Method
Primary Actor: Client Driver
Pre-conditions: disconnect is requested by user
Post-conditions: client is disconnected from server
Basic Flow or Main Scenario:

1. Client stops connection with server
2. Client logs the user out of the client

Extensions or Alternate Flows:
Exceptions: Server is down, user connection issue
Related Use Cases:

Use Case ID: 07

Use Case Name: Login

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: User must have started the client, the server must be running and prepared to handle a new thread

Post-conditions: User will either be accepted to login or rejected and if authenticated user gains access to comms app

Basic Flow or Main Scenario:

1. User starts the client and the client attempts a handshake with the server
2. User enters userID and password into client GUI.
3. Server compares the entered password to the password associated with the userID in the system
4. User is verified and logged into communications application
5. User can use communications application

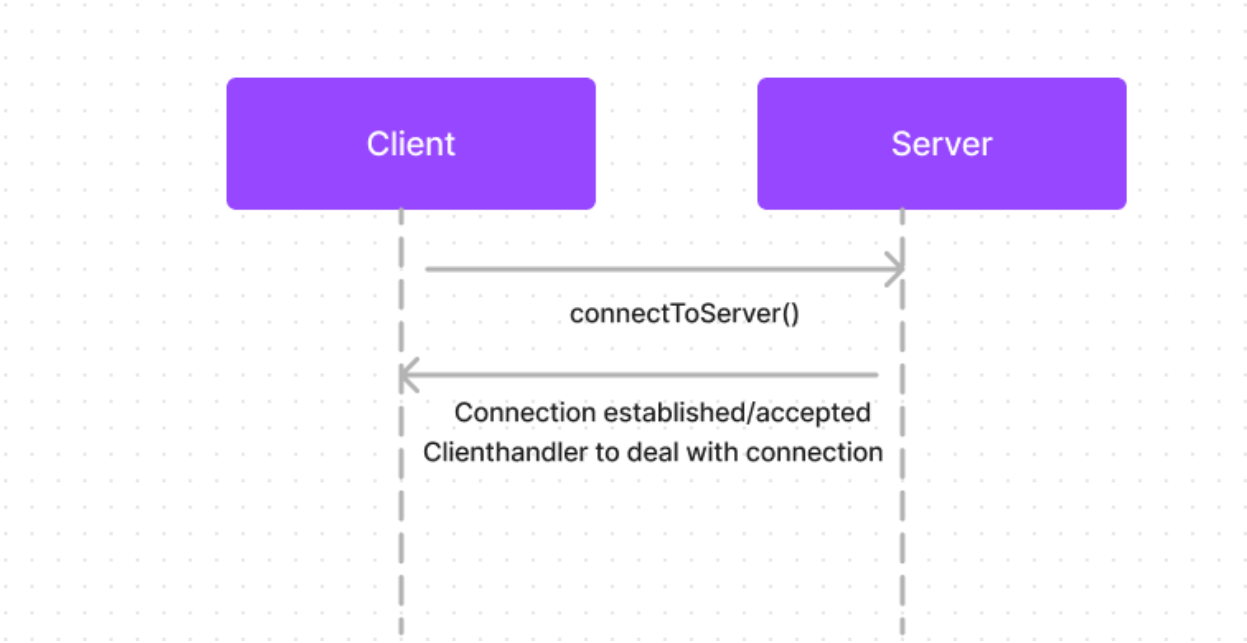
Extensions or Alternate Flows:

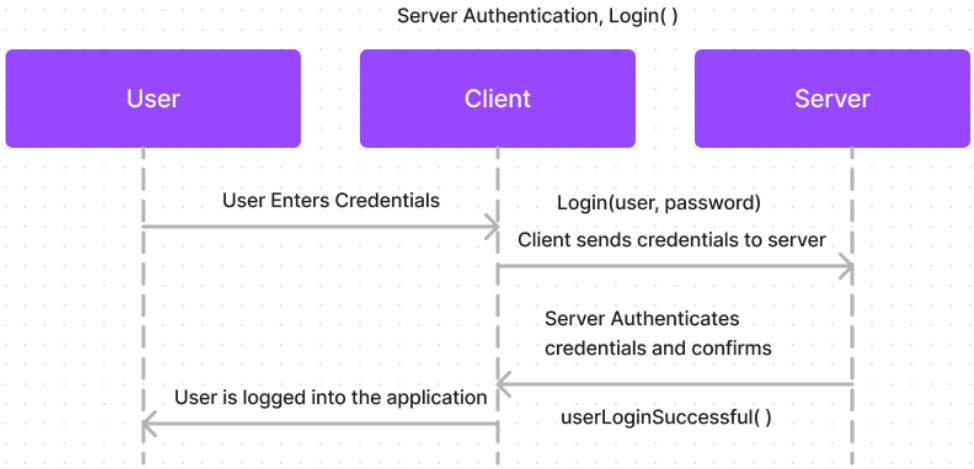
1. User enters userID and password
2. Server compares credentials to saved credentials and they do not match
3. User is not verified
4. User is not able to access the communications application

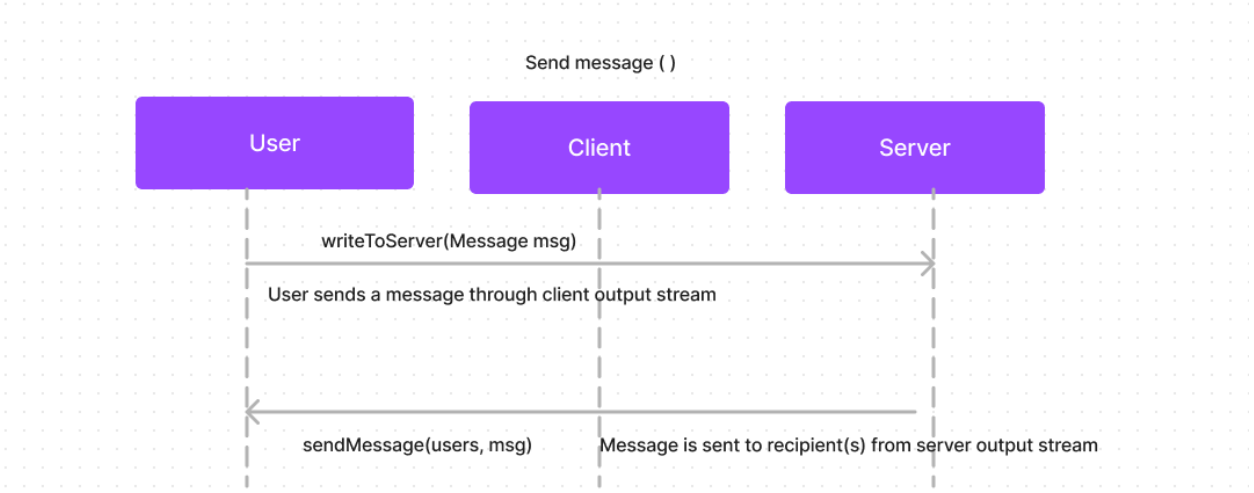
Exceptions: Incorrect Password, Incorrect Username, Server is down, Server fails to handle network threading.

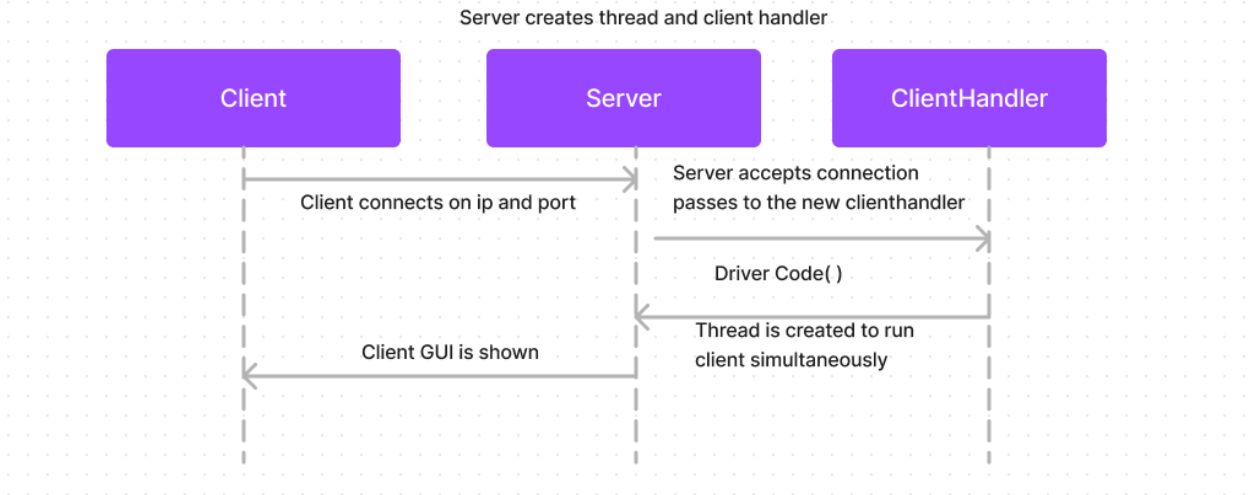
Related Use Cases: Server sends saved messages to client

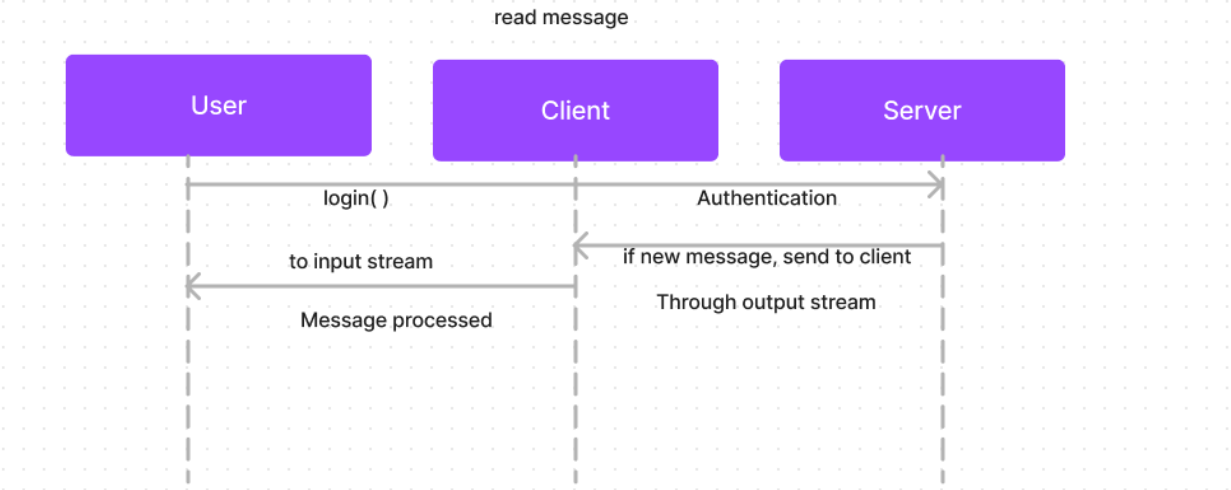
Sequence Diagrams for Server, ClientHandler, and Client:

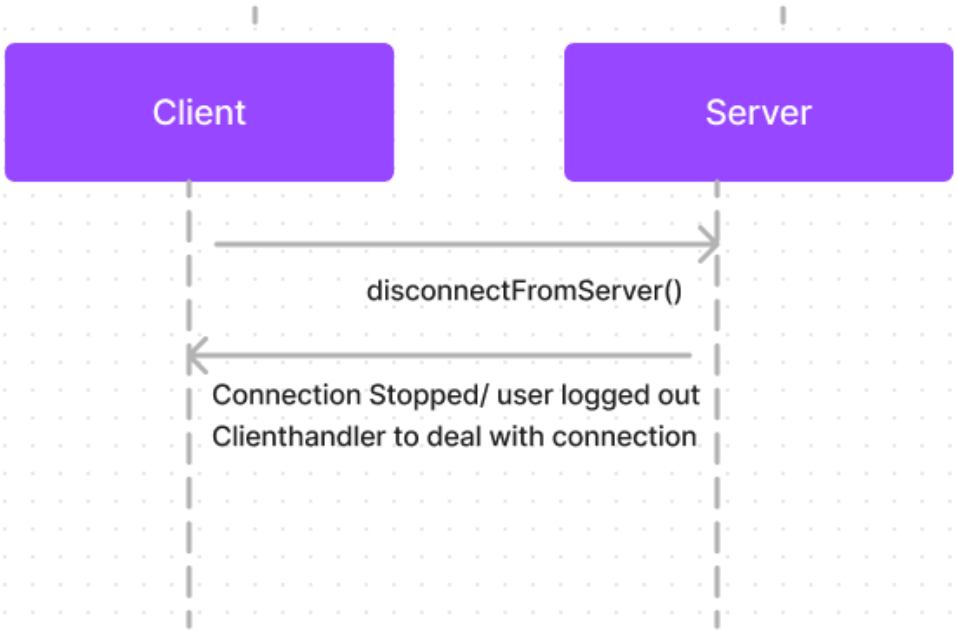




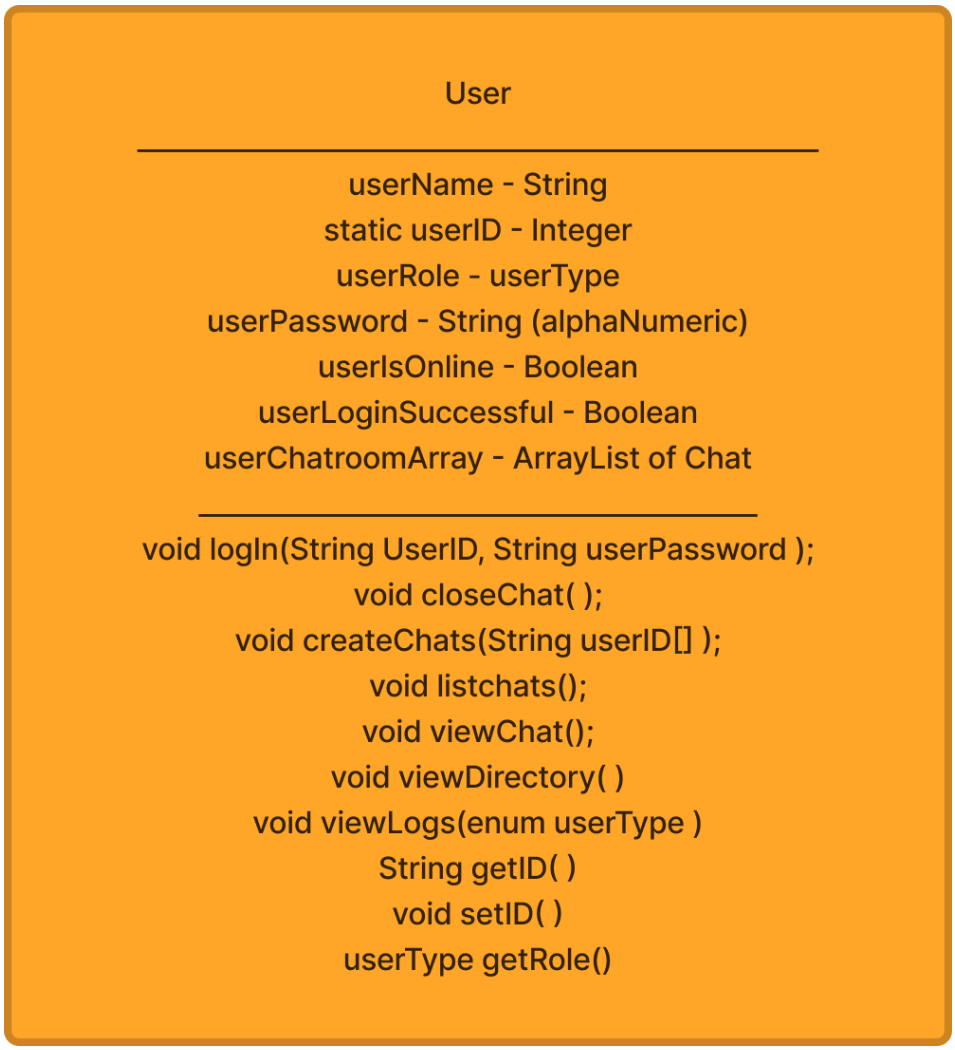

Send message ( )
User
Client
Server
writeToServer(Message msg)
User sends a message through client output stream
sendMessage(users, msg)
Message is sent to recipient(s) from server output stream


Server creates thread and client handler
Client
Server
ClientHandler
Client connects on ip and port
Server accepts connection
passes to the new clienthandler
Driver Code( )
Thread is created to run
client simultaneously
Client GUI is shown


read message
User
Client
Server
login( )
Authentication
to input stream
if new message, send to client
Through output stream
Message processed


Client
Server
disconnectFromServer()
Connection Stopped/ user logged out
Clienthandler to deal with connection

User:



userName: string holding the ID of the given user

userID - Unique user ID the system uses to find a user when given the userName

userRole - enumerated types that determine whether a user is IT or regular

userPassword - string holding the password the user will use to authenticate their connection to the server.

userIsOnline - Boolean for the server to check if a user is currently online or not

userLoginSuccessful - a boolean that checks to see if authentication was successful. Until this flag is flipped, the user cannot access their normal accessible services, and can only attempt to authenticate.

userChatroomArray - ArrayList of all of the IDs for the chatrooms the user is currently active in

userIDList [] - ArrayList of all users in the userbase, taken from a file sent over by the Server

login(String userID, string userPassword) - the client initiates an authentication attempt with the server.
closeChat() - close a chat, removing the messages from the UI and ending the connection to the chat (further messages will be held on server side). On client disconnect, close all chats.
CreateChat(String []UserIDs) - given a list of users, create a chatroom that all users are added to. Chatroom is created on the initiator's side and on other client sides upon first message to the chatroom.
listchats() - displays the list of existing chat rooms that the user should have access to. Allowing the user to select one to view.
viewChat() - opens the UI of the actual chatroom, allowing the user to view and send messages.
void **viewDirectory()-** displays the arraylist of existing users pulled from a file sent by the server to the current user so they can choose who they'd like to message.
void signOut() - logs the user out of the system by disconnecting their client from the server. Switches boolean userIsOnline from True to False, which will cause the server to save future messages and wait for connection.
void viewLogs(enum userType) - if the user has the IT enum type, pull previous chats by serial number or userID.
Integer getID() - returns the userID of the person currently signed into the client
userType getRole() - returns the role of the person currently signed into the client.
Void setID() - sets the ID by adding one to the current static integer for ID number.

Use Cases for User:

Use Case ID: 00

Use Case Name: Login

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: User must have started the client, the server must be running and prepared to handle a new thread

Post-conditions: User will either be accepted to login or rejected and if authenticated user gains access to comms app

Basic Flow or Main Scenario:

6. User starts the client and the client attempts a handshake with the server
7. User enters userID and password into client GUI.
8. Server compares the entered password to the password associated with the userID in the system
9. User is verified and logged into communications application

10. User can use communications application

Extensions or Alternate Flows:

5. User enters userID and password
6. Server compares credentials to saved credentials and they do not match
7. User is not verified
8. User is not able to access the communications application

Exceptions: Incorrect Password, Incorrect Username, Server is down, Server fails to handle network threading.

Related Use Cases: Server sends saved messages to client

Use Case ID: 01

Use Case Name: View Directory

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: A user would like to view the current directory of users, but cannot

Post-conditions: Client creates a current directory of users via a file that the server sends when requested. Clients may then view the existing directory.

Basic Flow or Main Scenario:

1. User requests a directory
2. Client uses the list of users sent over by the file on client connection to create a directory for the user
3. User views the directory, which can be used to identify user IDs for chats

Extensions or Alternate Flows:

Exceptions: Client fails to retrieve the file, Client fails to generate a list of names, Server fails to update the file and an incomplete or broken list is created

Related Use Cases:

Use Case ID: 02

Use Case Name: Create Chat

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: A user would like to create a chat, and the chat has not been created yet

Post-conditions: The user has created a chat by entering a list of users ranging from one user to as many users as the original user would like into the “create chat” dialog box. A chat is then created.

Basic Flow or Main Scenario:

1. Users enter as many usernames as they would like linked to valid users into a “create chat” page.
2. The client sends the IDs over to the server, which links these clients together in a chat room.
3. Chat room is created and ready to distribute messages

Extensions or Alternate Flows: Group chat is created, individual chat is created

Exceptions: Client fails to send the chatroom ID list to the server, the server fails to create the chatroom

Related Use Cases: Chats are sent, close chat room, create logs

Use Case ID: 03

Use Case Name: View Chat

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: A user would like to view a chat they are currently active in

Post-conditions: The user can view a chat linked to their account and currently active. They have opened the chat and all messages have been marked read.

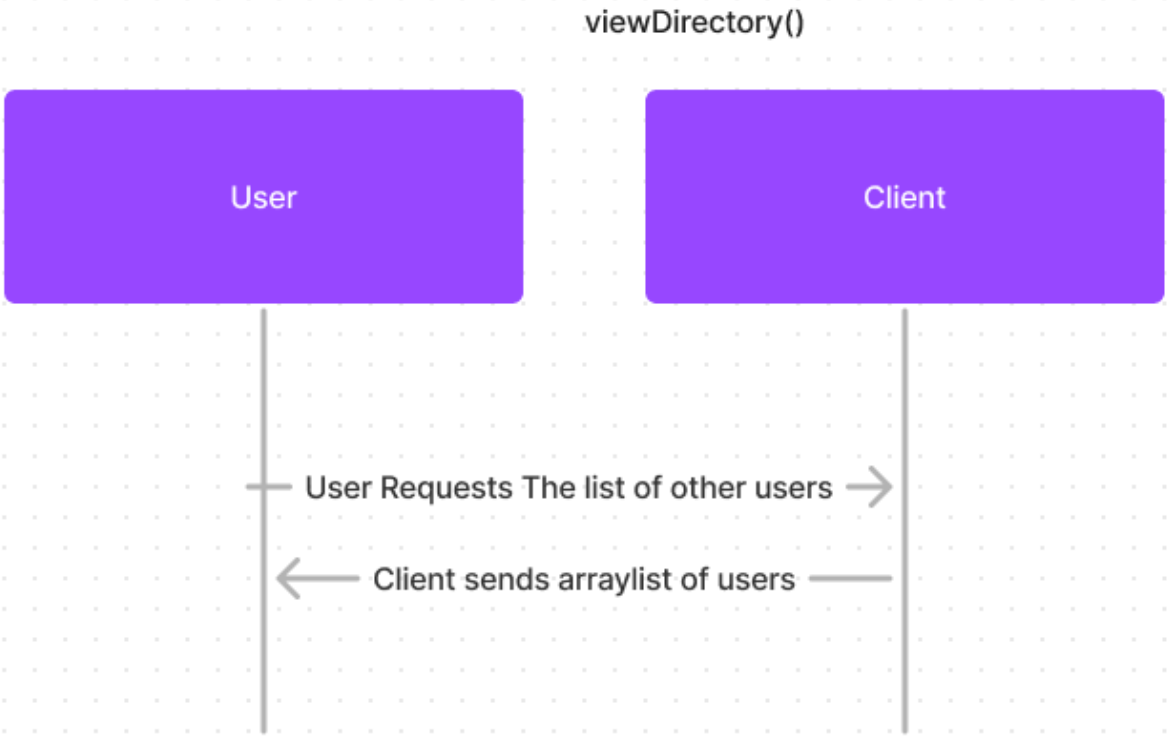
Basic Flow or Main Scenario:

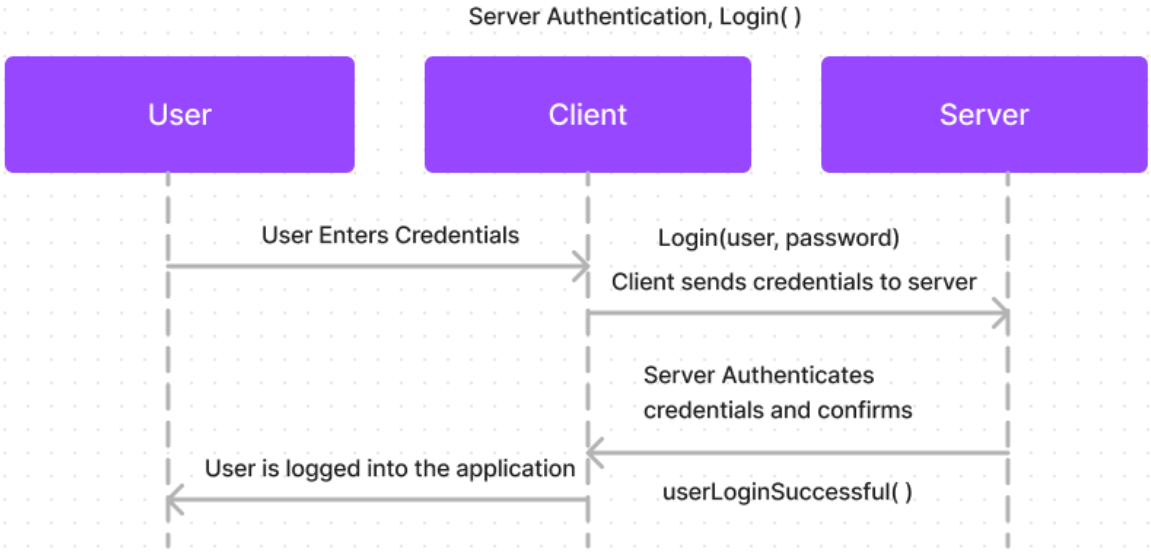
1. The user sees that there are unviewed and available chats via a notification pushed on the server side
2. The user clicks on this previous chat message and opens the chat room
3. The chat is viewed and the messages are marked as read.

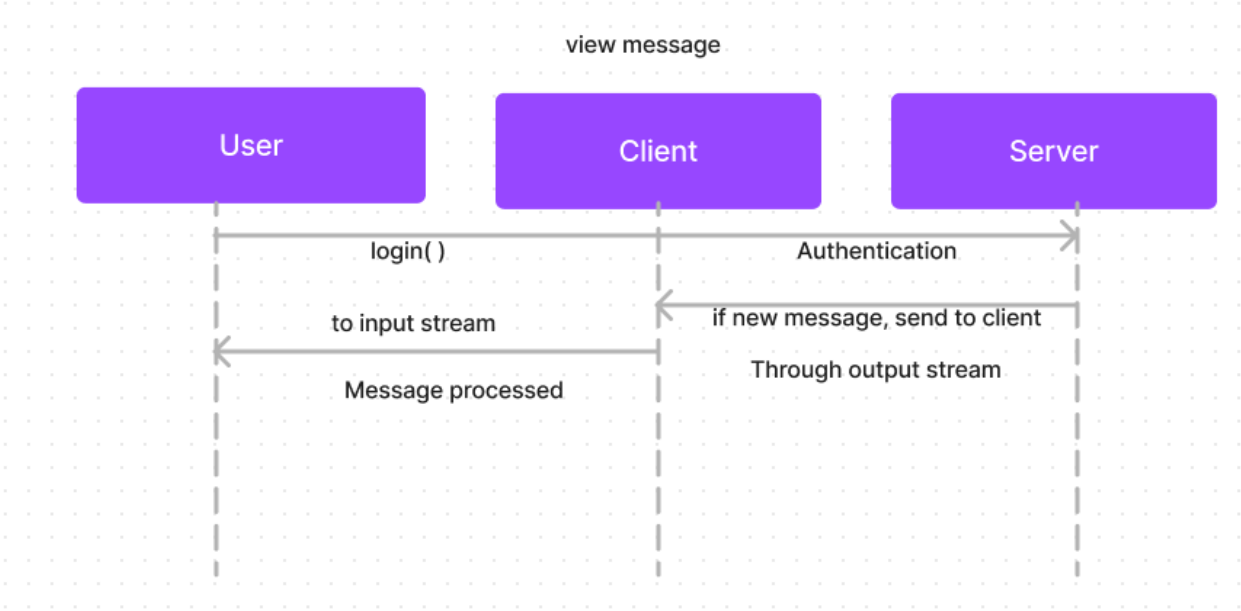
Extensions or Alternate Flows: A message is marked as read, a message is delivered asynchronously

Exceptions: Server fails to update the chat/notify the user, the chatroom refuses to open, user does not view the messages so they are marked as “sent” but are not also viewed and then get lost

Related Use Cases: Chats are sent, close chat room, create logs


viewDirectory()
User
Client
User Requests The list of other users
Client sends arraylist of users


Server Authentication, Login( )
User
Client
Server
User Enters Credentials
Login(user, password)
Client sends credentials to server
Server Authenticates
credentials and confirms
User is logged into the application
userLoginSuccessful( )


view message
User
Client
Server
login( )
Authentication
to input stream
if new message, send to client
Through output stream
Message processed

Message:

messageID - an integer serial number for the message

messageStatus - enumeration that marks a message as “CREATED”, “DELIVERED”, “RECEIVED”, or “ERROR”.

messageCreated - stores the date the message was created

messageString - String that stores the content of the message

messageSender - stores the ID of the sender of the message

messageReciever - stores the ID of the receiver of the message

String getMessageID(): creates an instance of a clienthandler, takes the clientsocket, the input and output streams of the socket.

Enum **getMessageStatus():** sets the enumeration type to CREATED, DELIVERED, RECEIVED, or ERROR.

Void updateMessageStatus(MessageStatus): updates the message status from its current status to another (for example, from CREATED to RECEIVED).

Date getMessageDate(): returns the date the message was sent

String getMessageString(): Returns the string holding the contents of the message.

Use Cases for Message:

Use Case ID: 00

Use Case Name: User creates a message

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: A user must want to send a message and adds user(s) to the chat, then sends a message

Post-conditions: a message will have been made, all of the attributes of the message have been assigned

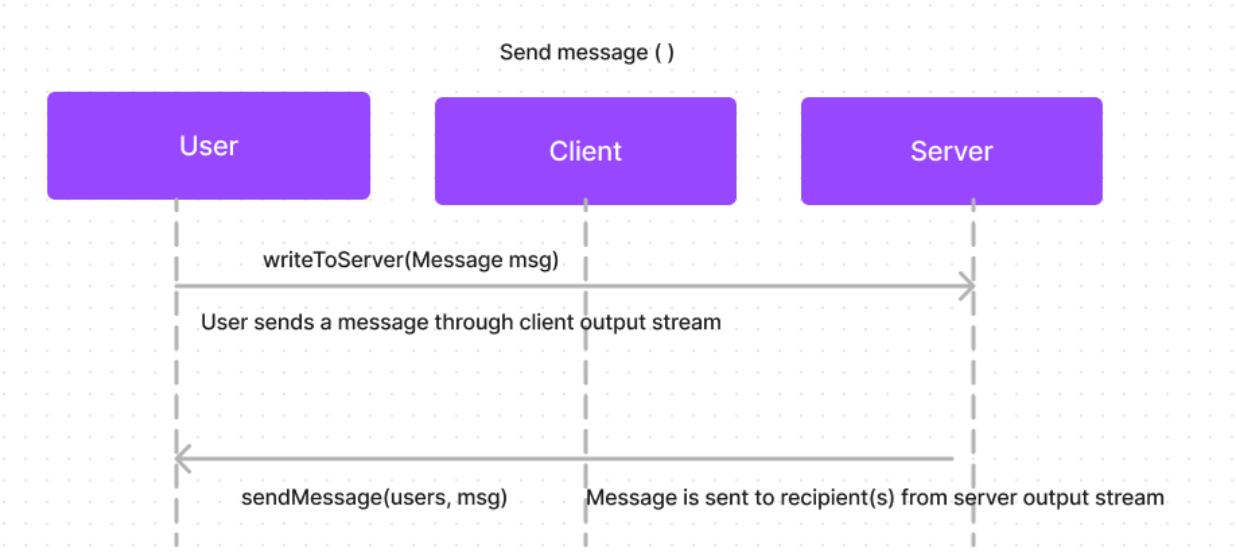
Basic Flow or Main Scenario:

1. Client adds username(s) to a list of chat recipients
2. A chatroom is created
3. A message is created. The date, sender, message string, and recipient are determined on construction.
4. MessageStatus is set to “created”

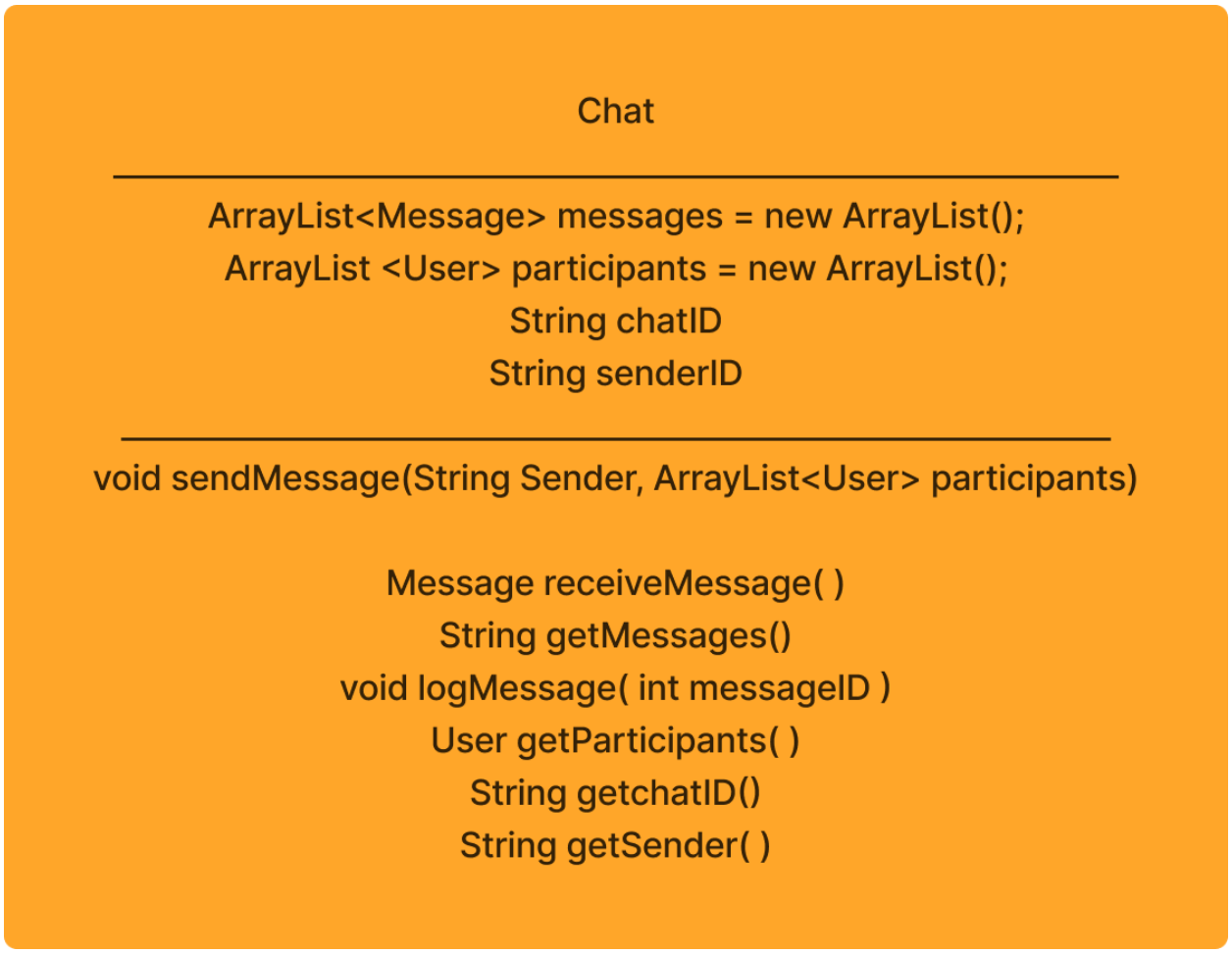
Extensions or Alternate Flows: message is delivered, message is received, message is of type error

Exceptions: Server is down, client fails to send message,

Related Use Cases: Client sends a message, Client sends a group message, Client receives a message, Server holds a message for delivery


Send message ( )
User
Client
Server
writeToServer(Message msg)
User sends a message through client output stream
sendMessage(users, msg)
Message is sent to recipient(s) from server output stream

CHAT



messages: Stores the messages that will exist in the chat instance.
participants: Stores the users that will exist in the chat instance.
chatID : A string variable, unique to each chatroom.
senderID: A string variable to store the userID of the sender.

void sendMessage(String sender, ArrayList <User> participants): Expects an arraylist of User objects, and also expects the senderID. The sendMessage() method will allow a message to be pushed out to be initialized. Using the elements from the participants arrayList and the userID of the sender.
recieveMessage(): returns the message that has been pushed out towards this chat and also adds it to the array of messages.
getMessages(): returns all the message that should exist in the chat
logMessage(messageID): allows for the message object with the matching messageID to be logged
getParticipants(): returns the participants ArrayList that exist in the chat room instance.
String chatID(): returns the string of the unique ID of that chatroom.
String getSender(): returns the String of the UserID that sent message

Use Cases for Chat:

Use Case ID: 00

Use Case Name: A message in the chat room is logged

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: A chat room has been created by the user, and the initial message has been sent to all chat rooms.

Post-conditions: A message has been sent out towards other users, and logged

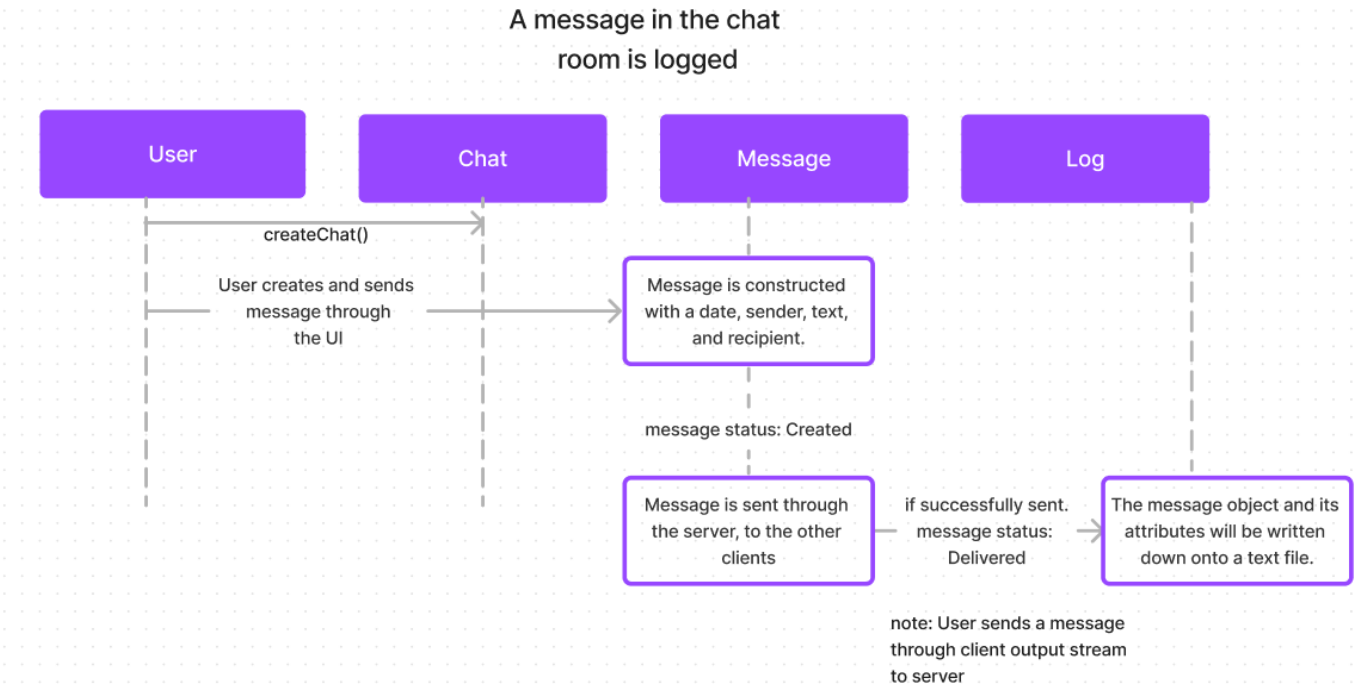
Basic Flow or Main Scenario:

- A chat room has been created by the user, by selecting users that will be the receivers and by typing the initial message to be sent
- That message has its attributes initialized upon construction. The date, sender, text, and recipient.
- MessageStatus is set to “created”
- The message has been sent to all the other clients
- If a message is successfully sent across the server to the recipient(s). The MessageStatus is set to “delivered”
- Upon being sent successfully, the message object and its attributes will be written down onto a text file.

Extensions or Alternate Flows: Whether a message was delivered synchronously or asynchronously, the message will still be logged.

Exceptions: If server is down, or connection is lost, then the message is created, but doesn’t send so the message will not be logged.

Related Use Cases: A message is created


A message in the chat room is logged
User
Chat
Message
Log
createChat()
User creates and sends message through the UI
Message is constructed with a date, sender, text, and recipient.
message status: Created
Message is sent through the server, to the other clients
if successfully sent. message status: Delivered
The message object and its attributes will be written down onto a text file.
note: User sends a message through client output stream to server

LOG:

boolean authenticateITUser(integer userID): Expects an integer (the user ID), to check if that user has the IT role. Returns true or false.
getLogs(): returns all of the information from the log, storying the messages, as a string.
filterChatLogsBySenderID(): returns the messages from a specific user, as a string
filterChatLogsByDate(): returns the messages as a string, from a specific date.

Use Cases for Log:

Use Case ID: 00

Use Case Name: display the logs for an ITUser

Relevant Requirements: SRS

Primary Actor: Client

Pre-conditions: The user currently logged in, has the IT role. And wishes to view all the messages from the past.

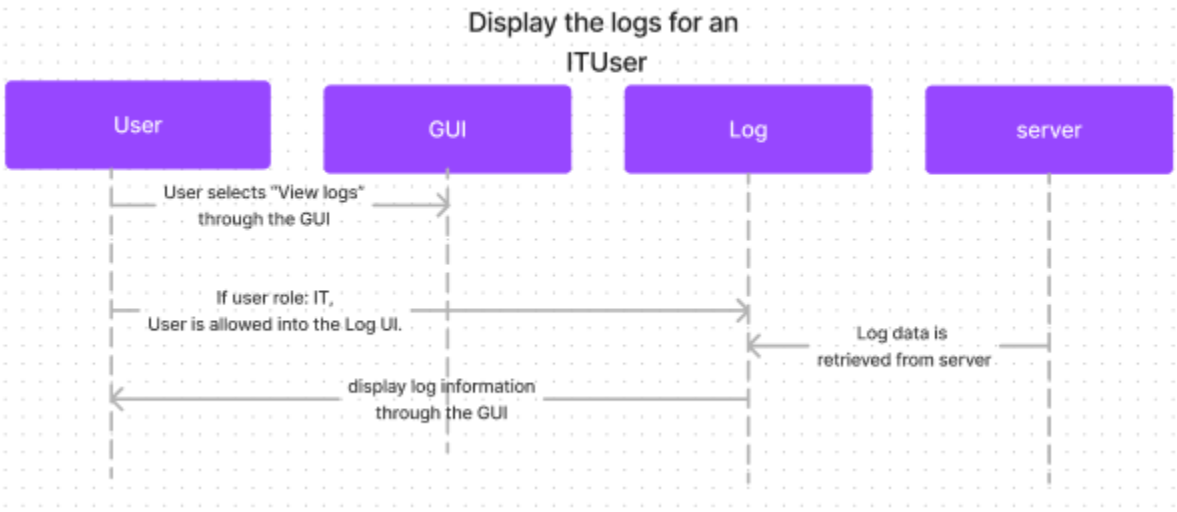
Post-conditions: none. Log info is only displayed on the GUI, the system experiences no changes.

Basic Flow or Main Scenario:

- The current logged in user, uses the GUI to select view logs
- User's role is checked, to see if they are IT.
- If successful, then the user is allowed into the showlogs UI.
- The logs text file will be retrieved, and the contents will be displayed for the user.

Extensions or Alternate Flows: After displaying the messages in the UI, the user may wish to filter the log by senderID or Date.

Exceptions: If server is down or connection is lost, log data may not be able to be retrieved.



GUI

client - Each GUI has a client, that means whenever a person tries to login using their credentials, the client will pass those credentials to the server and if authenticated, then the client will hold the user information for further actions/tasks that needs to be performed.

void authenticateUser( ) - This function is called whenever a person enters their credentials to be authenticated and clicks on the login button, hence the authentication function will pass the credentials of the user to the server for the verification.
void openApplication( ) - This function is called whenever a person is authenticated and it is an actual person. This function will load a different interface for the user to interact with which will have options like create a new chat room, or open already existing chats.
void showDirectory( ) - This function is called whenever a user tries to access the directory by clicking on the directory button, hence it displays the directory of the company
void startNewChat( ) - This function is called whenever a user tries to start a new chat with a person from the directory, so it creates the room by asking the server to add a new room to this account via client.
void openChatWindow( ) - This function is called whenever a user tries to open an already existing chat from the list. It opens a new interface with different options to display where the user can type in their message and read messages from the other participants.
void closeChatWindow( ) - This function is called whenever a user exits from the chat room they were in, and it brings up the old interface which openApplication( ) displayed.
void sendMessage( ) - This function is used whenever a user wants to send a message in the chat room. This function passes the message to the client and the client does the further transmission of the message.
void refreshWindow( ) - This function is used whenever a user receives a new message that needs to be displayed to the user
void showLogs( ) - This function is used whenever an IT user wants to view the logs, so the IT user clicks on the show logs button which brings up the new interface which has all the chat logs.
void closeLogs( ) - this function is used whenever an IT user wants to exit out of the logs interface, and it returns back to the old interface which openApplication( ) displayed.
void filterLogs( ) - This function is used whenever an IT user wants to filter out logs based on the senderID, that the IT user will input.
void signOut( ) - The function is used whenever the user presses the sign out button, and it calls the client to sign out and ends the 2-way connection with the server.

Use Cases for GUI:

Use Case ID: 00

Use Case Name: Software starts up and user signs out

Relevant Requirements:

Primary Actor: GUI and Client

Pre-conditions: A user has run the software.

Post-conditions: Depending on the credentials, the software can close or can open the synchronous chat rooms for users to send and receive messages.

Basic Flow or Main Scenario:

- The user starts the software, and the user has been asked about their username and password.
- Once the user provides their credentials, the client takes that information, and communicates with the server regarding the authenticity of those credentials.
- If the server authenticates the credentials, there is a new User Interface that pops up which has options like opening an already existing chat or creating a new one or opening a directory.
- The user presses the sign out button to sign out.
- The client asks the server via client handler to end the connection, and hence gets disconnected.

Extensions or Alternate Flows: User enters the credentials which the server can't authenticate, then the user needs to log in again and possibly can lead to shutting the software down.

Exceptions: Server is down, can't authenticate user's credentials.

Related Use Cases: