Assessment Cover Page

Module Title:	Concurrent Systems
Assessment Title:	CA2: YouChat
Lecturer Name:	Sam Weiss
Student Full Name:	Bekezhan Abdykarimov
Student Number:	2020297
Assessment Due Date:	21/05/2023
Date of Submission:	25/05/2023

Declaration

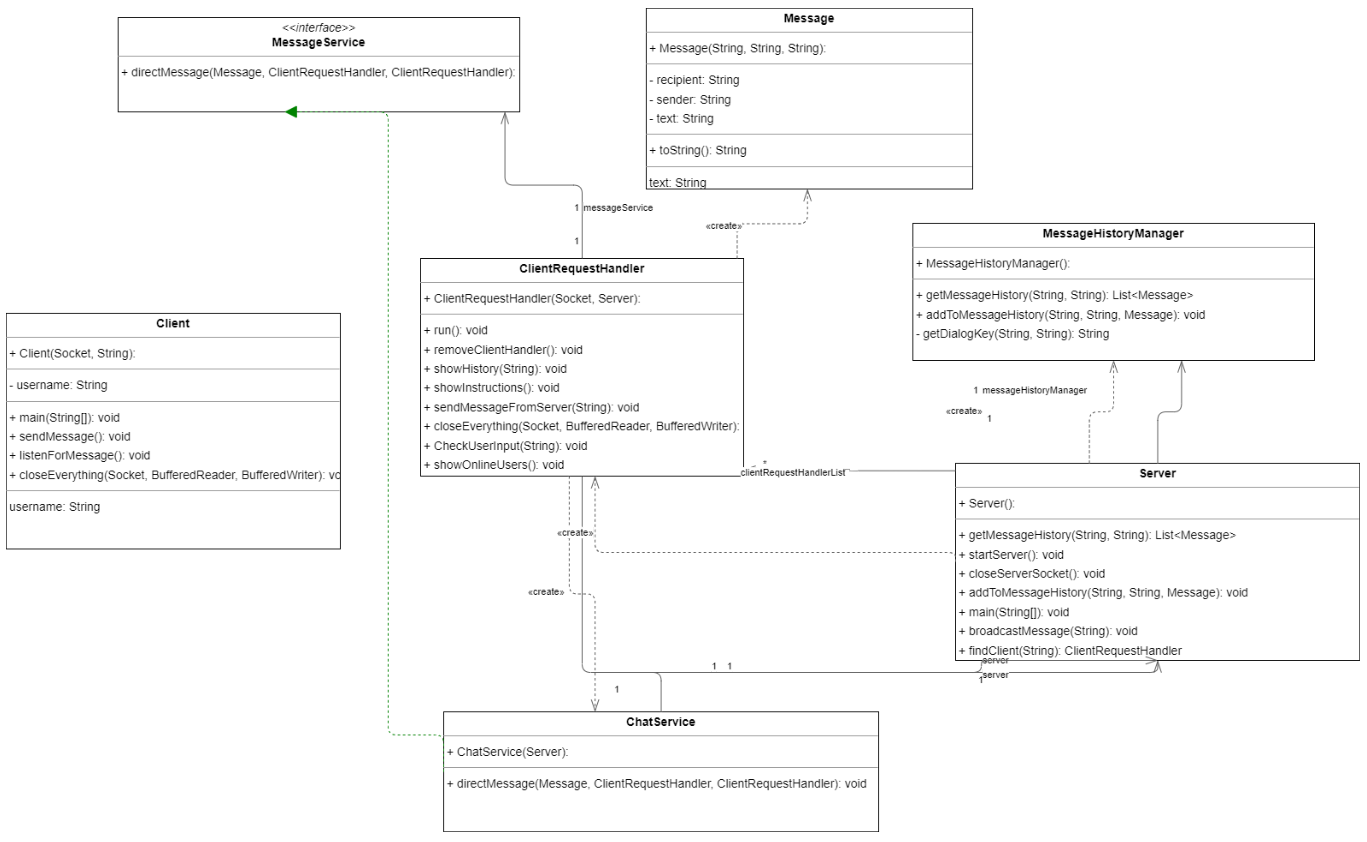
Introduction

This report describes the design and features of YouChat. YouChat - is the application in Java that I have developed as a learning project. YouChat application allows multiple users to connect to a server and communicate with each other in real time, it also allows users to see how many users are online, and to check the message history with a particular user.

Design

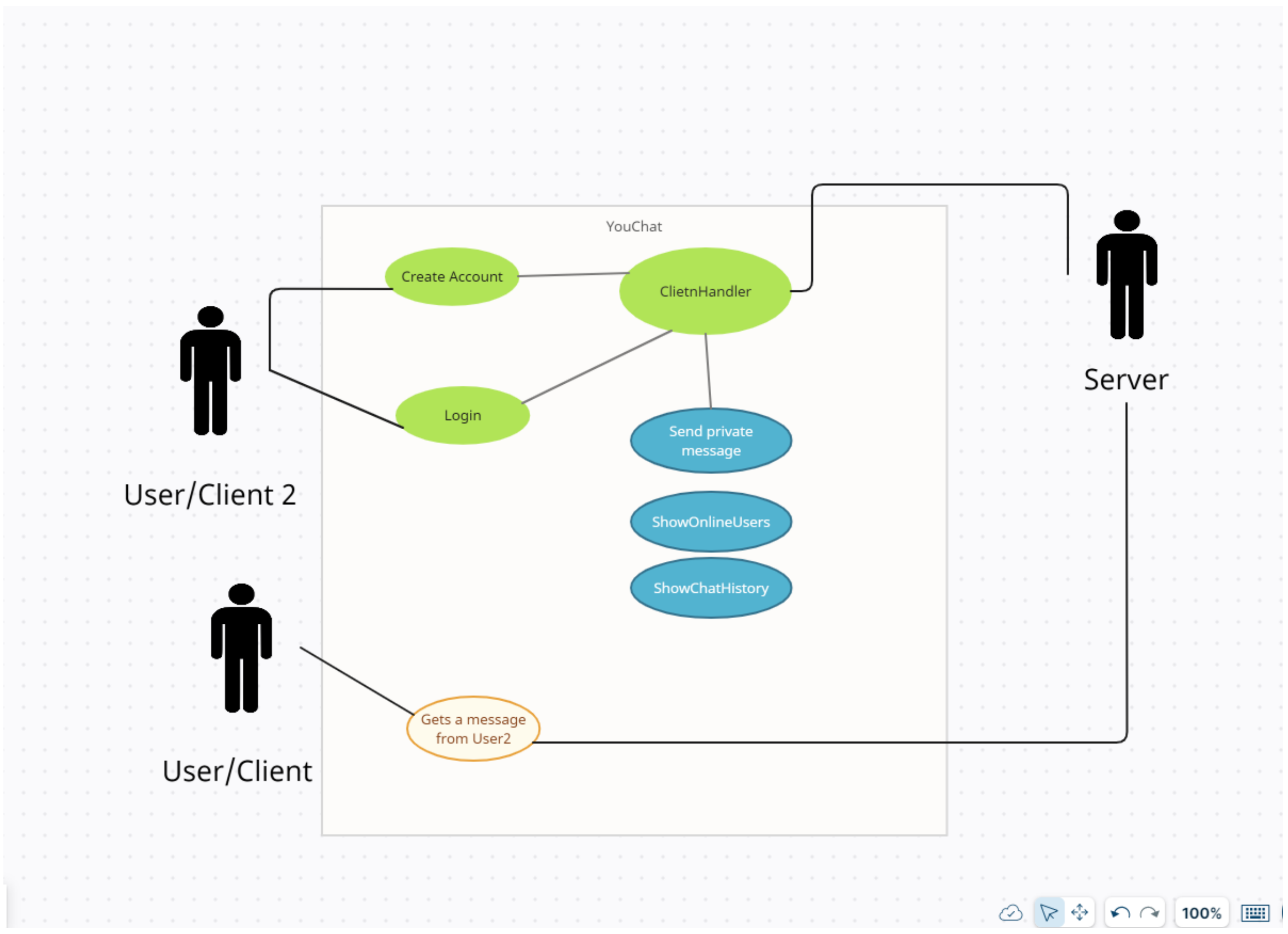
YouChat application consists of two parts: the server and the client. Each part can run independently on separate computers in the same network. Since it is only a prototype, it will run only locally (on localhost). The server can handle multiple clients at the same time using threads. The client can send and receive messages from the server and other clients.

Class Diagram:



<<interface>>
MessageService
+ directMessage(Message, ClientRequestHandler, ClientRequestHandler):
Message
+ Message(String, String, String):
- recipient: String
- sender: String
- text: String
+ toString(): String
text: String
MessageHistoryManager
+ MessageHistoryManager():
+ getMessageHistory(String, String): List<Message>
+ addToMessageHistory(String, String, Message): void
- getDialogKey(String, String): String
Client
+ Client(Socket, String):
- username: String
+ main(String[]): void
+ sendMessage(): void
+ listenForMessage(): void
+ closeEverything(Socket, BufferedReader, BufferedWriter): vo
username: String
ClientRequestHandler
+ ClientRequestHandler(Socket, Server):
+ run(): void
+ removeClientHandler(): void
+ showHistory(String): void
+ showInstructions(): void
+ sendMessageFromServer(String): void
+ closeEverything(Socket, BufferedReader, BufferedWriter):
+ CheckUserInput(String): void
+ showOnlineUsers(): void
Server
+ Server():
+ getMessageHistory(String, String): List<Message>
+ startServer(): void
+ closeServerSocket(): void
+ addToMessageHistory(String, String, Message): void
+ main(String[]): void
+ broadcastMessage(String): void
+ findClient(String): ClientRequestHandler
ChatService
+ ChatService(Server):
+ directMessage(Message, ClientRequestHandler, ClientRequestHandler): void
1 messageService
1
«create»
1 messageHistoryManager
«create» 1
*
clientRequestHandlerList
«create»
«create»
1 1
server
1 server
1

Use case diagram:



Below I will briefly explain purposes, roles and meaning of the each class:

Server

The server is implemented by two classes: `Server` and `ClientRequestHandler`.

The `Server` class starts the server, listening on a specific port. It has the following fields:

- `messageHistoryManager`: manages the message history between users.
- clientRequestHandlerList: list of all connected clients

ClientRequestHandler class

The ClientRequestHandler is the main class that is responsible to handle client requests and send an appropriate response. It implements the Runnable interface and handles the communication with multiple clients and it have the following fields:

- `clientUsername`: stores the username of the client.
- `clients_count`: counts online clients.
- `server`: a reference to the Server object that created this handler.
- `clientSocket`: represents the connection with the client.

- `bufferedReader`: a reference to a BufferedReader object that reads data from the client socket's input stream.
- `bufferedWriter`: a reference to a BufferedWriter object that writes data to the client socket's output stream.
- `messageService`: MessageService object that handles sending messages to other users.

ClientRequestHandler class main methods:

- `ClientRequestHandler(Socket clientSocket, Server server)`: the constructor

- `run()`: the method that overrides the run() method, which is called in Server class when we call a "new Thread(clientRequestHandler).start() " .
  It first sends the username of the client to the server and broadcasts a message to all online users that a new user has connected. It then shows the instructions for using the app to the client. It then enters a loop that reads messages from the client and checks their input. If the input starts with a "/", it means it is a command and it calls the `CheckUserInput()` method to handle it. Otherwise, it is a normal message and it calls the `messageService`'s `directMessage()` method to send it to all users.

- `CheckUserInput(String messageToSend)`: the method that decomposes the client's input and performs different actions depending on the input. It has three cases:
  - If the input is "/msg Username YourMessage", it means the client wants to send a private message to another user. It then extracts the recipient and the text from the input and creates a `Message` object. It then calls the `messageService`'s `directMessage()` method to send it to the recipient. If the recipient is not found, it sends an error message to the client.
  - If the input is "/showOnlineUsers", it means the client wants to see the list of online users. It then calls the `showOnlineUsers()` method to display them.
  - If the input is "/showHistory Username", it means the client wants to see the chat history with another user. It then calls the `showHistory()` method to display them.

- `showOnlineUsers()`: the method that sends a message to the client with the number and names of online users by looping through
- `showHistory(String username)`: the method that sends a message to the client with the chat history with another user by calling the server's `getMessageHistory()` method and looping through the list of messages. If the list is empty, it sends a message that no history is found.

- `sendMessageFromServer(String message)`: the method that sends a message to the client by writing it to the buffered writer and flushing it. It catches any IOException and calls the `closeEverything()` method to close the resources.
- `removeClientHandler()`: the method that removes this handler from the user threads set in the server and decrements the clients count. It also broadcasts a message to all online users that this user has left the chat.

The rationale for designing the ClientRequestHandler class in this way is:

- To use a Socket object to represent the connection with the client and access its input and output streams.
- To use a BufferedReader and a BufferedWriter objects to read and write data from and to the client socket's input and output streams.
- To use a MessageService object to handle sending messages to other users using different methods depending on the type of message.
- To use a Runnable interface to implement a run() method that handles the communication with the client in a separate thread.
- To use a String object to store and send the username of the client.
- To use commands starting with "/" to perform different actions such as sending private messages, showing online users, and showing chat history.

Client class

The `Client` class represents the client program that connects to the server and communicates with other clients. It has the following fields:

Features

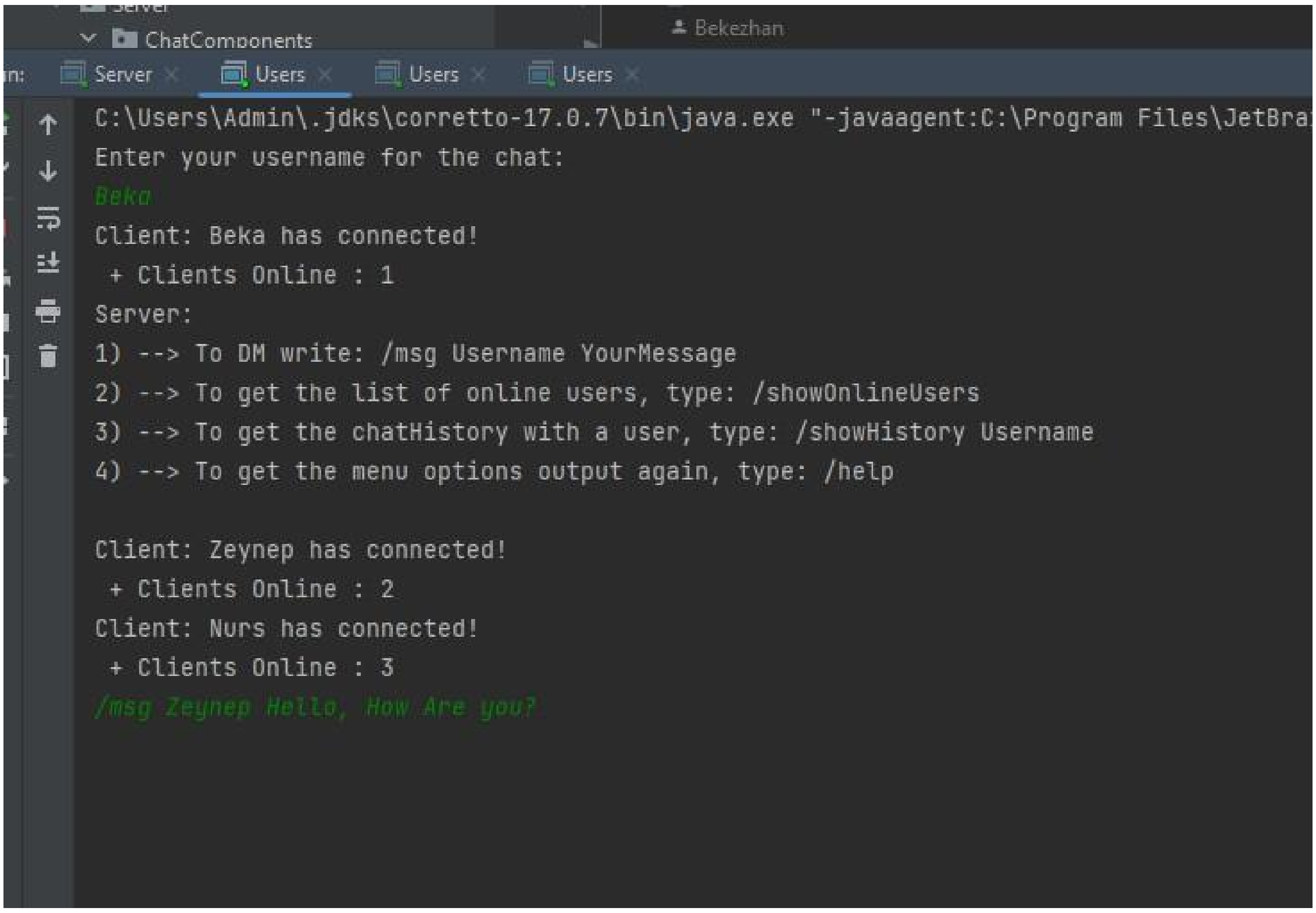
YouChat has the following features:

- The users can connect to the server by providing their usernames and the server's address and port.
- The users can send and receive messages from the server and other users in real time.
- The users can send private messages to specific users by using the command "/msg Username YourMessage".
- The users can see the list of online users by using the command "/showOnlineUsers".
- The users can see the chat history with a specific user by using the command "/showHistory Username".

Some Screenshots of the App functionality :

- Sending the Private Message

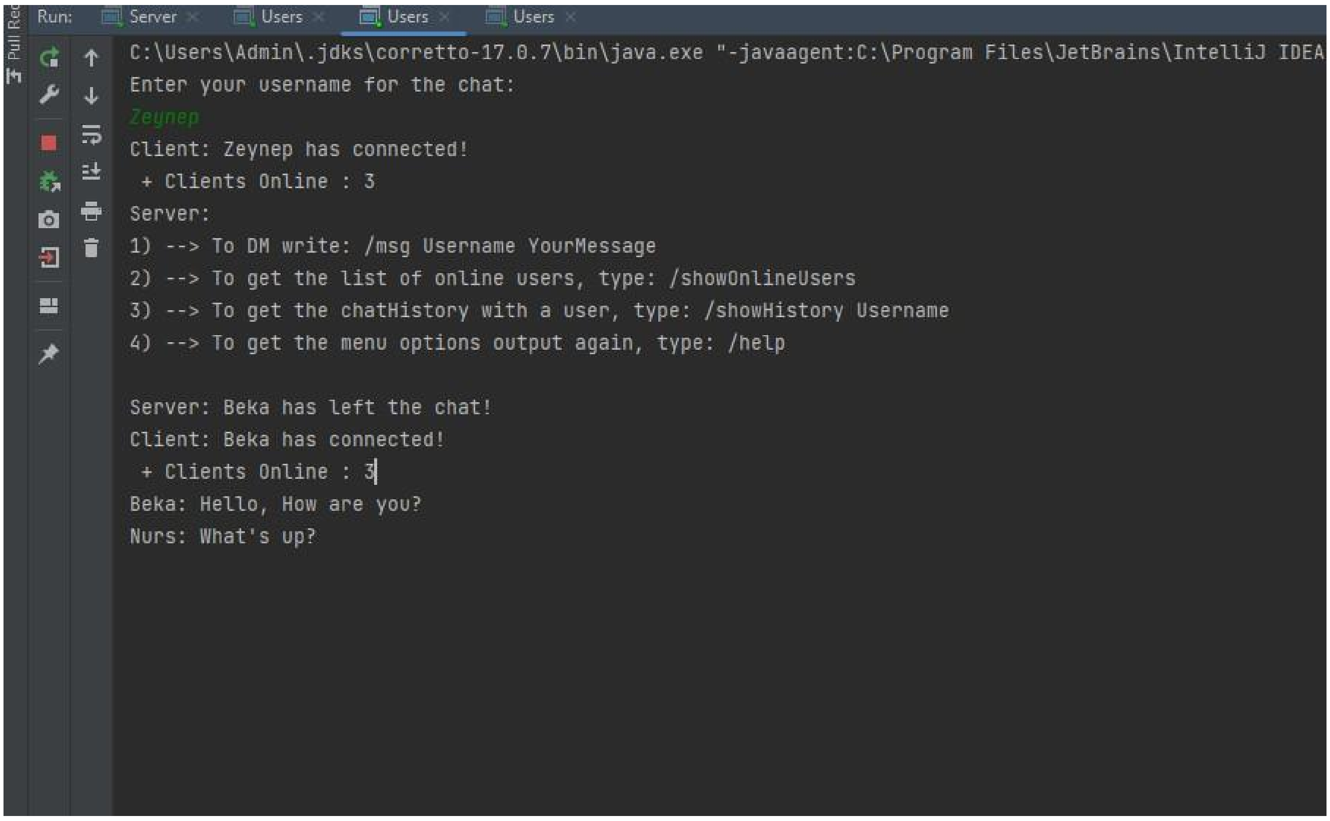
Client - Beka sends the private message to the client Zeynep :



```
C:\Users\Admin\.jdks\corretto-17.0.7\bin\java.exe "-javaagent:C:\Program Files\JetBra
Enter your username for the chat:
Beka
Client: Beka has connected!
 + Clients Online : 1
Server:
1) --> To DM write: /msg Username YourMessage
2) --> To get the list of online users, type: /showOnlineUsers
3) --> To get the chatHistory with a user, type: /showHistory Username
4) --> To get the menu options output again, type: /help

Client: Zeynep has connected!
 + Clients Online : 2
Client: Nurs has connected!
 + Clients Online : 3
/msg Zeynep Hello, How Are you?
```

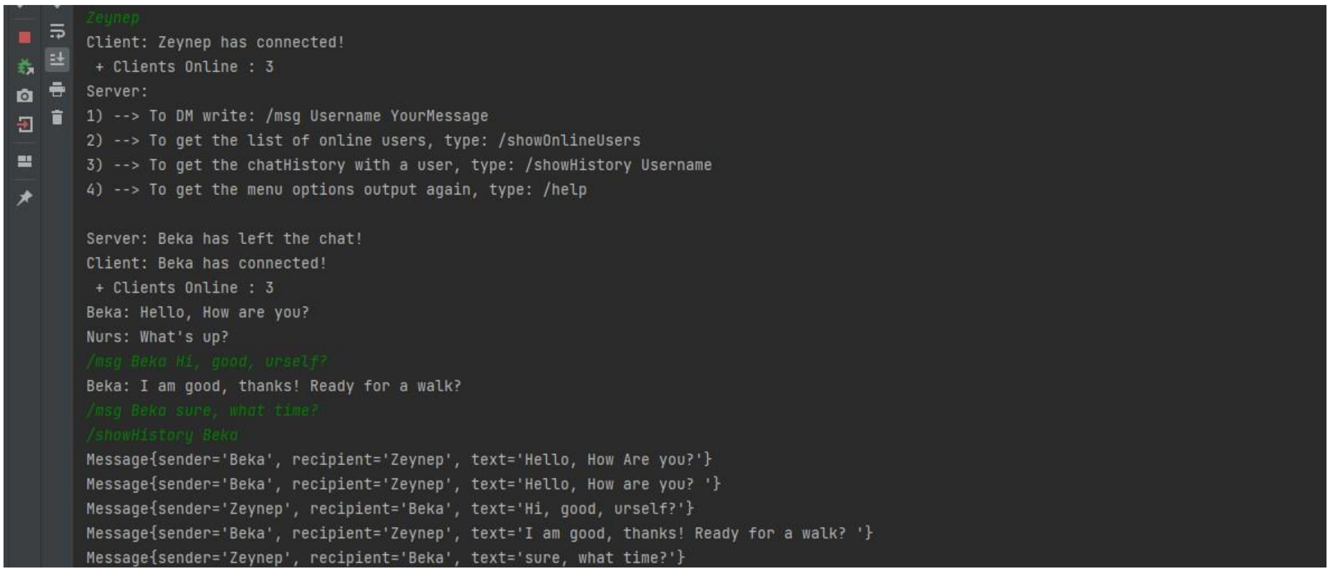
Client Zeynep gets the messages:



```
C:\Users\Admin\.jdks\corretto-17.0.7\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter your username for the chat:
Zeynep
Client: Zeynep has connected!
 + Clients Online : 3
Server:
1) --> To DM write: /msg Username YourMessage
2) --> To get the list of online users, type: /showOnlineUsers
3) --> To get the chatHistory with a user, type: /showHistory Username
4) --> To get the menu options output again, type: /help

Server: Beka has left the chat!
Client: Beka has connected!
 + Clients Online : 3
Beka: Hello, How are you?
Nurs: What's up?
```

- Checking the message History between clients from the example above:



```
Zeynep
Client: Zeynep has connected!
 + Clients Online : 3
Server:
1) --> To DM write: /msg Username YourMessage
2) --> To get the list of online users, type: /showOnlineUsers
3) --> To get the chatHistory with a user, type: /showHistory Username
4) --> To get the menu options output again, type: /help

Server: Beka has left the chat!
Client: Beka has connected!
 + Clients Online : 3
Beka: Hello, How are you?
Nurs: What's up?
/msg Beka Hi, good, urself?
Beka: I am good, thanks! Ready for a walk?
/msg Beka sure, what time?
/showHistory Beka
Message{sender='Beka', recipient='Zeynep', text='Hello, How Are you?'}
Message{sender='Beka', recipient='Zeynep', text='Hello, How are you? '}
Message{sender='Zeynep', recipient='Beka', text='Hi, good, urself?'}
Message{sender='Beka', recipient='Zeynep', text='I am good, thanks! Ready for a walk? '}
Message{sender='Zeynep', recipient='Beka', text='sure, what time?'}
```

Concurrency Issues and Solutions

YouChat faces some concurrency issues due to the use of threads and shared resources :

- The server needs to handle multiple clients at the same time without blocking or interfering with each other.
- The server needs to broadcast messages to all online users without missing or duplicating any messages.
- The server needs to store and retrieve the message history between users without losing or mixing up any messages.
- The client needs to send and receive messages from the server without blocking or interfering with each other.

To solve these issues, the chat application uses the following techniques:

- The server uses a ServerSocket object to listen for incoming connections from clients and create Socket objects for each connection. It then creates a new thread for the ClientRequestHandler object for each connection and adds it to the ArrayList<ClientRequestHandler> (list of connected clients). It also starts the thread to handle the communication with the client. This way, the server can handle multiple clients concurrently using separate threads.
- The server uses a MessageHistoryManager object to manage the message history between users. It uses a map of strings to lists of messages to store the history for each pair of users. This way, the server can avoid concurrent modification and inconsistency issues when accessing and updating the shared resources.
- The client uses a Socket object to connect to the server and access its input and output streams. It then creates a BufferedReader object and a BufferedWriter object to read and write data from and to the server socket’s input and output streams. It uses BufferedReader object to read input from the console. It then creates a new thread that listens for messages from the server by reading from the buffered reader and printing them to the console. It also enters a loop that reads messages from the console and sends them to the server by writing them to the buffered writer. This way, the client can send and receive messages concurrently using separate threads.

Conclusion

This report has presented the design and features of a chat application in Java that I have developed as a learning project.

The chat application uses various concepts and techniques in Java, such as socket programming, threads, data structures, algorithms. Socket programming allows the creation of network applications that communicate with each other using TCP/IP protocol.

It can be further improved by adding more features, such as :

1) Planning the project design and architecture. I am not really good at it, because of the lack of practice, but I tried to avoid the beginners mistake and instead of writing the code right away, I started creating diagrams for use cases and classes, I was trying to plan it as much as I could and I was trying to implement SOLID principles during that process. The final result wasn’t really satisfying and now I see that I could avoid strong dependencies between the classes, use Single Responsibility principle more and separate the tasks, which would make code maintenance easier.

2) How to make messages persist after restart? There are two options - to save the chat log in the server files (such a practice), and use the database. I would use the second option to upgrade the project and   sqlite is a great option for that.

3) Make a graphical interface for chat. By using Swing, and output all your console sout and readline() to the resulting interface.

4) We could also do simple encryption..

It can also be tested and evaluated on different machines and networks to ensure its functionality and performance.

References:

www.youtube.com. (n.d.). *Java Socket Programming - Multiple Clients Chat*. [online] Available at: https://www.youtube.com/watch?v=gLfuZrrfKes&t=649s [Accessed 24 May 2023].

www.codejava.net. (n.d.). *How to Create a Chat Console Application in Java using Socket*. [online] Available at: https://www.codejava.net/java-se/networking/how-to-create-a-chat-console-application-in-java-using-socket.

JavaRush. (2153). *Классы Socket и ServerSocket в Java*. [online] Available at: https://javarush.com/groups/posts/654-klassih-socket-i-serversocket-ili-allo-server-tih-menja-slihshishjh [Accessed 24 May 2023].

Lucidchart (2019). *UML Use Case Diagram Tutorial | Lucidchart*. [online] Lucidchart.com. Available at: https://www.lucidchart.com/pages/uml-use-case-diagram.