# CCT College Dublin

| | |
|---|---|
| Module Title: | Concurrent Systems |
| Assessment Title: | CA2 - YouChat |
| Lecturer Name: | Sam Weiss |
| Student Full Name/ID: | Heber Junio dos Anjos Mota – 2020317 |
| Assessment Due Date: | 21/05/2023 |
| Date of Submission: | 20/05/2023 |

# Application for Chat on Youtube: YouChat

## Rationale & Design

In this project I was challenged to incorporate a real-time messaging feature called YouChat, where YouTube, the widely used video-sharing platform, hopes to improve user engagement and conversation. The design and execution of a YouChat prototype that will be smoothly integrated into the current YouTube application are described in this project proposal. In my role as YouTube's hired designer and developer, my objective is to build a productive, simple to use messaging system.

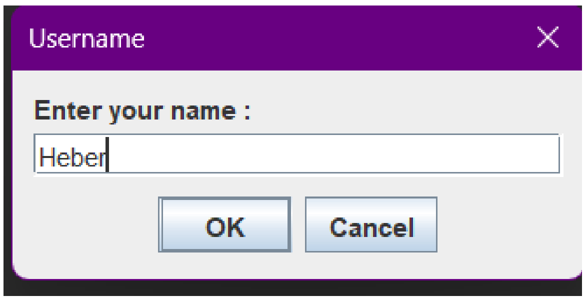
This report gives some insight in the key features of the application, which aims to facilitate communication between users in a one-on-one messaging environment. As a prototype, the primary focus is on local functionality and the ability to handle multiple users and chats simultaneously, as especified below:

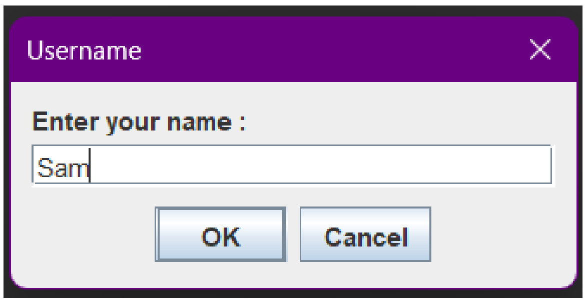
- Client-server architecture where server enables communication between clients;
- Users can send/receive messages to each other (one-on-one);
- Users can view their message history with other users (non-persistent);
- Server can handle multiple users and multiple chats for each user.

Once the application was ready, that was the final design:

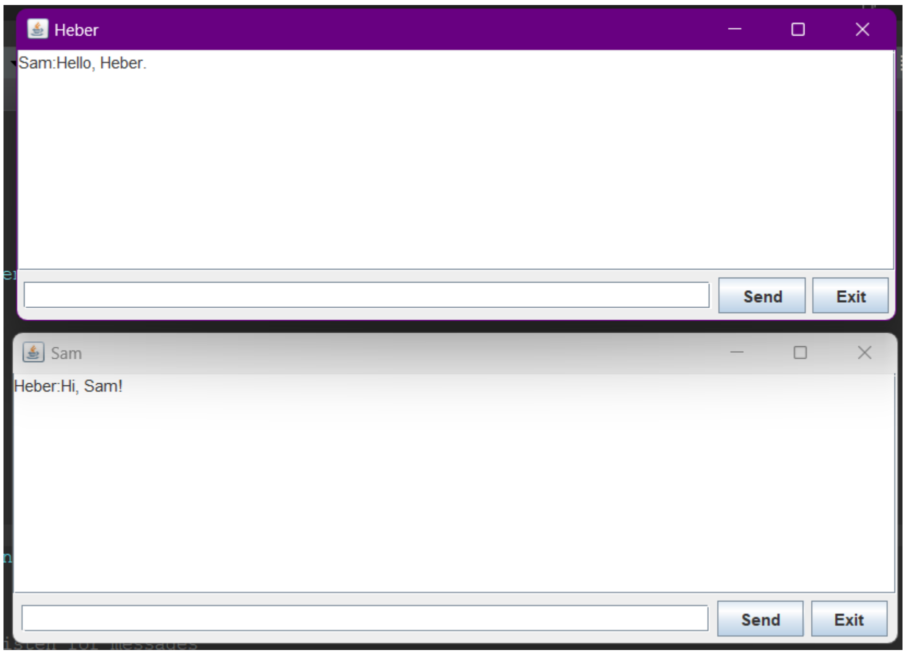
**Step 1:** Once the class ChatServer.java is up, we then name the Users, they can be created by running the class "ChatClient.java" as many times as desired – each time you setup a new user, they will be able to exchange messages with each other!

Here we are setting up the users Heber & Sam:





**Step 2:** After the users have registered, they will be able to see their conversation tabs on the screen. There, they can start typing their message in the text box. They have two options: they can either click on the **Send** button or the **Exit** chat button, both located at the bottom right corner of the chat window.

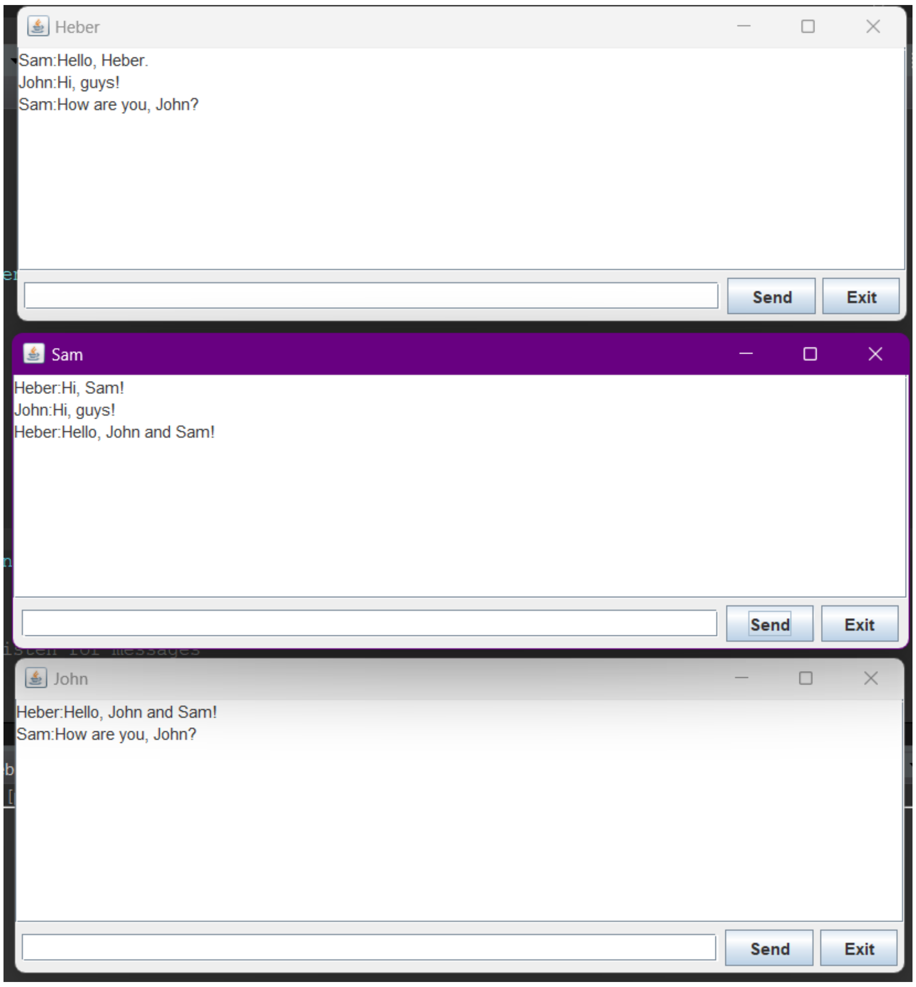

Heber

Sam:Hello, Heber.

Sam

Heber:Hi, Sam!

**Step 3:** Another feature of the program is that it allows multiple users to communicate with each other. For example, I created a third user called John and he can send messages to both Sam and Heber. This way, they can have a group chat as shown below:


Heber

Sam:Hello, Heber.
John:Hi, guys!
Sam:How are you, John?

Sam

Heber:Hi, Sam!
John:Hi, guys!
Heber:Hello, John and Sam!

John

Heber:Hello, John and Sam!
Sam:How are you, John?

## Troubleshooting

When developing a messaging application such as YouChat, I faced challenges such as **synchronization of the real-time conversation in chat, scalability and performance, message history and storage, error handling and reliability**. These challenges include managing message queues, handling concurrent connections, optimizing resource usage, and implementing error handling mechanisms.

- Synchronization of the real-time chat: I had issues to ensure that the messages sent and received by different users were synchronized. By using a socket I was able to stablish a connection between clients and server. I also had to implement a real-time communication protocol that could handle concurrent connections and deliver the messages sent instantly.

- Scalability & performance: I had to design the application architecture and the backend infrastructure to support a scalable number of users and messages. .

- Error handling and reliability: I had to handle various errors and exceptions that could occur during the communication process. I had to implement retry and fallback mechanisms to ensure that the messages were delivered even in case of network failures or server outages. I also had to test and debug the application several times to ensure the correct functioning.

## Conclusion

In this project, we developed a chat application using multi threads in Java. The main goal was to create a server that can handle multiple clients simultaneously and allow them to communicate with each other.

I have learned how to use **sockets, threads, input/output streams, and data structures** to implement the functionality of the chat application. We also learned how to design a user interface and how to handle different types of messages and commands. I faced some challenges such as synchronizing the threads, managing the client list, and handling exceptions and errors. However, I was able to overcome them by applying the concepts and techniques we learned in the course and from other sources.

The chat application is a useful and practical example of how multi-threading can enhance the performance and interactivity of a network-based program.

# References

- Point, J. (2023). Java Socket Programming. [online] Java Point. Available at: https://www.javatpoint.com/socket-programming [Accessed 13 May 2023].

- Lobster, C. (2023). Creating a Chat Server Using Java : 8 Steps (with Pictures) - Instructables. [online] instructables.com. Available at: https://www.instructables.com/Creating-a-Chat-Server-Using-Java/ [Accessed 13 May 2023].

- Code, W. (2023). Java Socket Programming - Multiple Clients Chat. [online] youtube.com. Available at: https://www.youtube.com/watch?v=gLfuZrrfKes&ab_channel=WittCode [Accessed 13 May 2023].

- Ven, R. (2022). Java - Chat Application ( Part 1 ). [online] youtube.com. Available at: https://www.youtube.com/watch?v=8e-Q5KeVTPw&ab_channel=RaVen [Accessed 14 May 2023].

- Barhoumi, J. (2021). Create a chat app with java sockets. [online] medium.com. Available at: https://medium.com/nerd-for-tech/create-a-chat-app-with-java-sockets-8449fdaa933 [Accessed 14 May 2023].