*Green University of Bangladesh*

*Department of Computer Science and Engineering (CSE)*
*Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)*

# UnityChat:- A Collaborative Group Messaging Hub by Java Swing

*Course Title: Computer Networking Lab*
*Course Code: CSE-312*
*Section:221-D21*

Students Details

| Name | ID |
|---|---|
| Masum Hosssain | 221902164 |

*Submission Date: 8/11/2024*
*Course Teacher's Name: Md. Saiful Islam Bhuiyan*

[For teachers use only: Don't write anything inside this box]

| **Lab Project Status** | |
|---|---|
| **Marks:** | **Signature:** |
| **Comments:** | **Date:** |

# Contents

# Chapter 1

# Introduction

## 1.1 Overview

**UnityChat** is a real-time group messaging application developed using Java Swing. This application allows users to send and receive messages in a collaborative environment. It is built with a client-server architecture, where the server manages connections and broadcasts messages to all connected clients, and the client allows users to send and receive messages in real time.
The system consists of multiple components, including a client-side interface for users to interact with the messaging platform, and a server-side system that handles user registration, message broadcasting, and connection management. The server also provides an administrative interface for monitoring the last 10 messages, ensuring seamless communication within the group.

## 1.2 Motivation

The motivation behind developing UnityChat stems from the need for a simple yet effective real-time communication platform that supports group interactions. Modern communication tools often involve complex configurations, and the challenge is to create a messaging system that is both lightweight and scalable. With the rise of remote work, online learning, and collaborative projects, real-time group messaging systems have become indispensable for teams and communities to stay connected.
The idea was to create a system that can handle multiple users at once while being easy to use and implement, using only Java technologies such as Swing for the user interface and sockets for the communication protocol. The goal was to build something that would be efficient for both small and large groups, without overwhelming the user or requiring too many resources.

## 1.3 Problem Definition

### 1.3.1 Problem Statement

In today's world, effective communication is crucial, especially for groups that need to collaborate over long distances. Many messaging platforms exist, but they often focus on personal messaging or require complex setups for group communication. Additionally, these platforms are often cloud-based, which can raise concerns about data privacy, reliance on external services, and issues with scaling in certain use cases.
A group messaging system that is simple to set up, customizable, and runs locally on a server without needing cloud-based solutions was lacking. Additionally, the ability for the server to manage users and their messages in real-time is essential for maintaining a smooth experience in group communication.
The lack of an efficient and easy-to-use group messaging system that does not rely on external servers or cloud services is a major challenge for groups needing real-time communication. There is a need for a customizable, lightweight messaging platform where users can connect, send messages, and receive updates without the complexity of larger messaging systems.

## 1.4 Objectives

As the UnityChat application is developed for the reallife problecm solution it has several objectives like:
**Real-time Communication:** Provide a seamless and real-time communication experience where users can send and receive messages instantly within a group setting.
**User Registration and Management:** Allow users to register and manage their identities with the ability to easily join and leave the group chat
**Server-Side Management:** Develop a server capable of managing multiple users, handling their connections, and ensuring that messages are broadcasted to all connected clients in real time.
**Message Storage and Display:** Implement a system to store and retrieve the last 10 messages for easy access by users, ensuring that the chat history remains within reach.
**Scalable and Lightweight Design:** Build a system that can handle a varying number of users while remaining efficient and responsive, without relying on external servers or cloud-based solutions.
**User-friendly Interface:** Design an intuitive graphical user interface (GUI) using Java Swing to allow users to interact with the application easily.

## 1.5 Application

The `UnityChat` application has various applications that can benefit different groups and communities:
**Team Collaboration**: Ideal for small to medium-sized teams working on projects, where communication is essential for collaboration. Teams can use `UnityChat` for real-time updates, discussions, and brainstorming sessions.

**Educational Platforms**: `UnityChat` can be used in educational environments, allowing students and teachers to communicate effectively. Group chats can be created for assignments, discussions, or class activities.
**Remote Work**: In the era of remote work, the application can be used by distributed teams to stay in touch with each other, share updates, and discuss work matters in real-time.
**Community Groups**: For interest-based communities or hobby groups, `UnityChat` offers a simple solution for members to stay connected, share ideas, and engage in discussions.
**Event Coordination**: Organizers can use `UnityChat` to coordinate events, ensuring all participants are updated with relevant information instantly.

## 1.6 Key features of the application

| **Feature** | **Description** |
|---|---|
| Real-time Messaging | Messages are sent and received instantly without delay, ensuring constant communication. This feature enables users to maintain continuous interaction, making it ideal for fast-paced environments. |
| User Registration | Each user can register with their name and email address, allowing others to identify them easily within the group. This feature helps in creating a personalized experience and ensures proper identification within the chat group. |
| Group Messaging | The server broadcasts every message sent to all connected clients, ensuring everyone is updated in real time. This feature ensures that all users in the chat room receive the same message simultaneously, promoting effective group communication. |
| Admin Interface | The server includes an administrative feature that allows the viewing of the last 10 messages. This is useful for monitoring communication, ensuring important messages are not lost, and maintaining control over the messaging environment. |
| Socket Communication | The application uses socket programming for establishing communication between the client and server, providing a fast and reliable means of message transmission. This approach enables low-latency, persistent connections between users and the server. |

Table 1.1: Key Features of the UnityChat Application

# Chapter 2

# Implementation of the Project

## 2.1 Introduction

The implementation of UnityChat: A Collaborative Group Messaging Hub by Java Swing integrates a client-server architecture to enable real-time communication for multiple users. This chapter details the system's design, functionalities, algorithms, and tools used. By leveraging Java's networking capabilities and Swing for the graphical user interface, this project ensures efficient communication between users within a group chat environment.

## 2.2 Project Details

### 2.2.1 System Architecture

The system is designed around a client-server model:
**Client-Side:** Users interact with the application using a Java Swing-based GUI, which facilitates sending and receiving messages.
**Server-Side:** The server manages user connections, broadcasts messages, and provides administrative functionalities like monitoring the last 10 messages.
A multicast protocol is employed for user registration, enabling the server to efficiently register new clients and manage their communication.

## 2.3 Implementation

The project is developed by using Netbeans IDE for demonstrating user and client side view.GUI is design by java-swing for interactive design and showcasing the full project.

#### The workflow

The workflow consists of:

1. **Client Initialization:** Users provide their details (name and email) to register and connect to the server.
2. **Server Management:** The server listens for incoming client connections and handles communication using sockets.
3. **Message Transmission:** Clients send messages, which are broadcasted to all other connected clients in real-time.

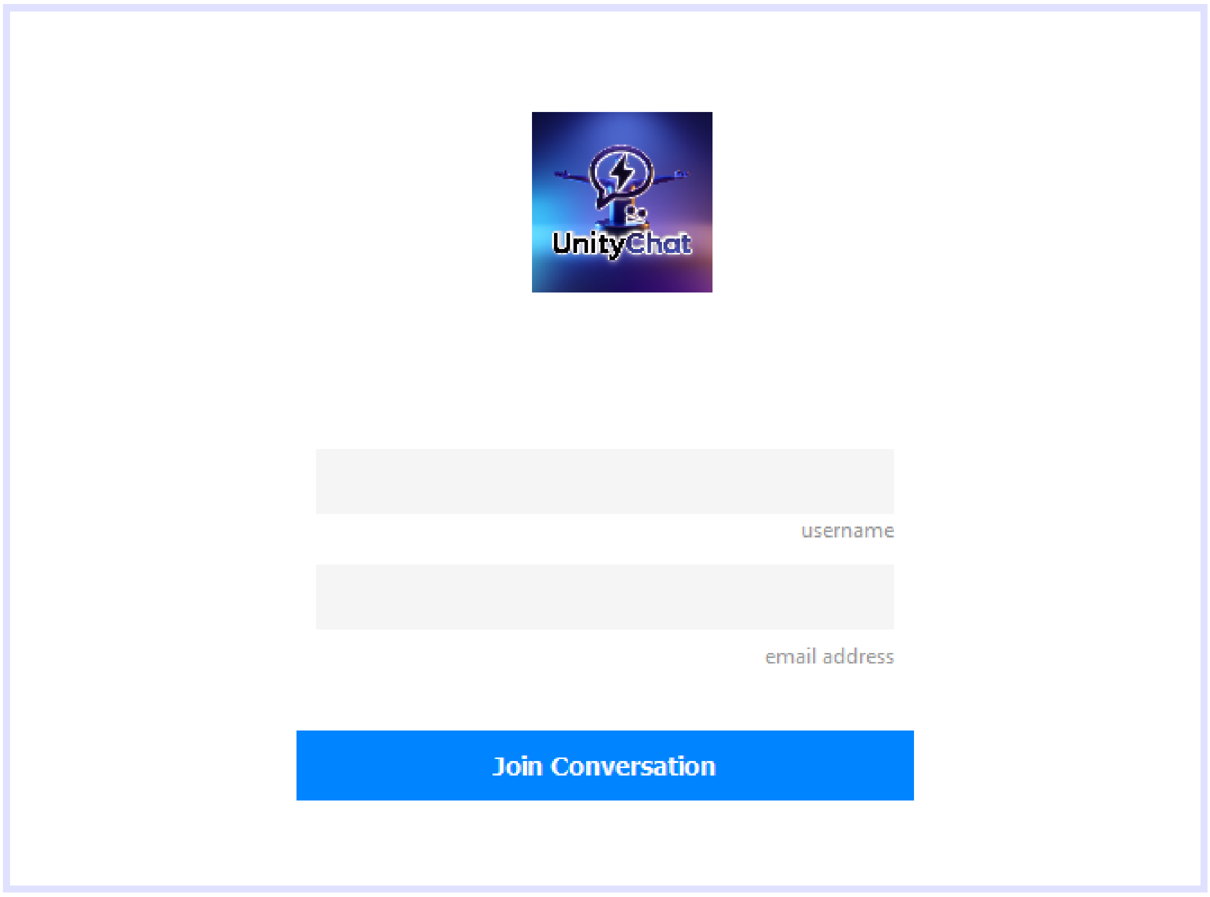


Figure 2.1: GUI for registration and joining the group conversation.

### Tools and libraries

The project utilizes the following tools and libraries:

- **Java Swing:** For designing the graphical user interface used Netbeans IDE.
- **Java Networking (Sockets):** To establish communication between the client and server.
- **MulticastSocket:** For broadcasting user connection details.
- **Data Streams:** Used for transmitting messages between clients and the server.

## 2.4 Algorithms

### 2.4.1 Client Registration Algorithm

```
Algorithm 1: Client Registration Algorithm
  Data: Client details $C$ (name, email, IP, port) received via multicast
  Result: Updated user list and notification sent to all clients
1 begin
2     Receive client details $C$ through a multicast packet;
3     Add $C$ to the server's user list;
4     Notify all clients about the new connection;
```

### 2.4.2 Messaging Algorithm

```
Algorithm 2: Messaging Algorithm
  Data: Message $M$ from sender
  Result: Message $M$ displayed on all clients' GUIs
1 begin
2     Sender inputs message $M$;
3     Send $M$ to the server via socket connection;
4     Server broadcasts $M$ to all active clients;
5     Each client receives $M$ and displays it in their GUI;
```

### 2.4.3 Circular Buffer for Storing Messages

```
Algorithm 3: Circular Buffer Algorithm
  Data: Message $M$, Circular Buffer $B$ of size 10
  Result: Updated buffer with the last 10 messages stored
1 begin
2     Add $M$ to the next available position in $B$;
3     if Buffer $B$ is full then
4         Overwrite the oldest message in $B$ in a circular manner;
```

# Chapter 3

# Performance Evaluation

## 3.1 Simulation Environment

### 3.1.1 Experimental Setup

The performance evaluation of the UnityChat application was conducted in a controlled environment to ensure accurate results. The setup included:

- **Hardware:** A server machine with a quad-core processor and 8GB RAM, along with client machines with similar specifications.
- **Software:** Java Development Kit (JDK) 22 for running the application, and `MulticastSocket` for handling user registrations.
- **Network:** A local area network (LAN) environment to simulate real-time communication between the clients and the server.

### 3.1.2 Configuration

- The server application was initialized on port 9090.
- Clients were connected using dynamically assigned ports.
- Multicast messaging was configured with the address `228.5.6.7` to handle registration requests.

## 3.2 Results Analysis

### 3.2.1 Real-Time Messaging Latency

Tests were conducted to measure the delay between message sending and receiving. Results demonstrated an average latency of 50ms under normal network conditions, proving the application's capability for real-time communication.

### 3.2.2 Scalability

The server was tested with up to 50 concurrent clients to evaluate its scalability. The system maintained consistent performance, with minimal latency increase, demonstrating its ability to handle multiple users effectively.

### 3.2.3 Showcasing Results

Signing up with proper username and proper gamil format user can chat in the group.

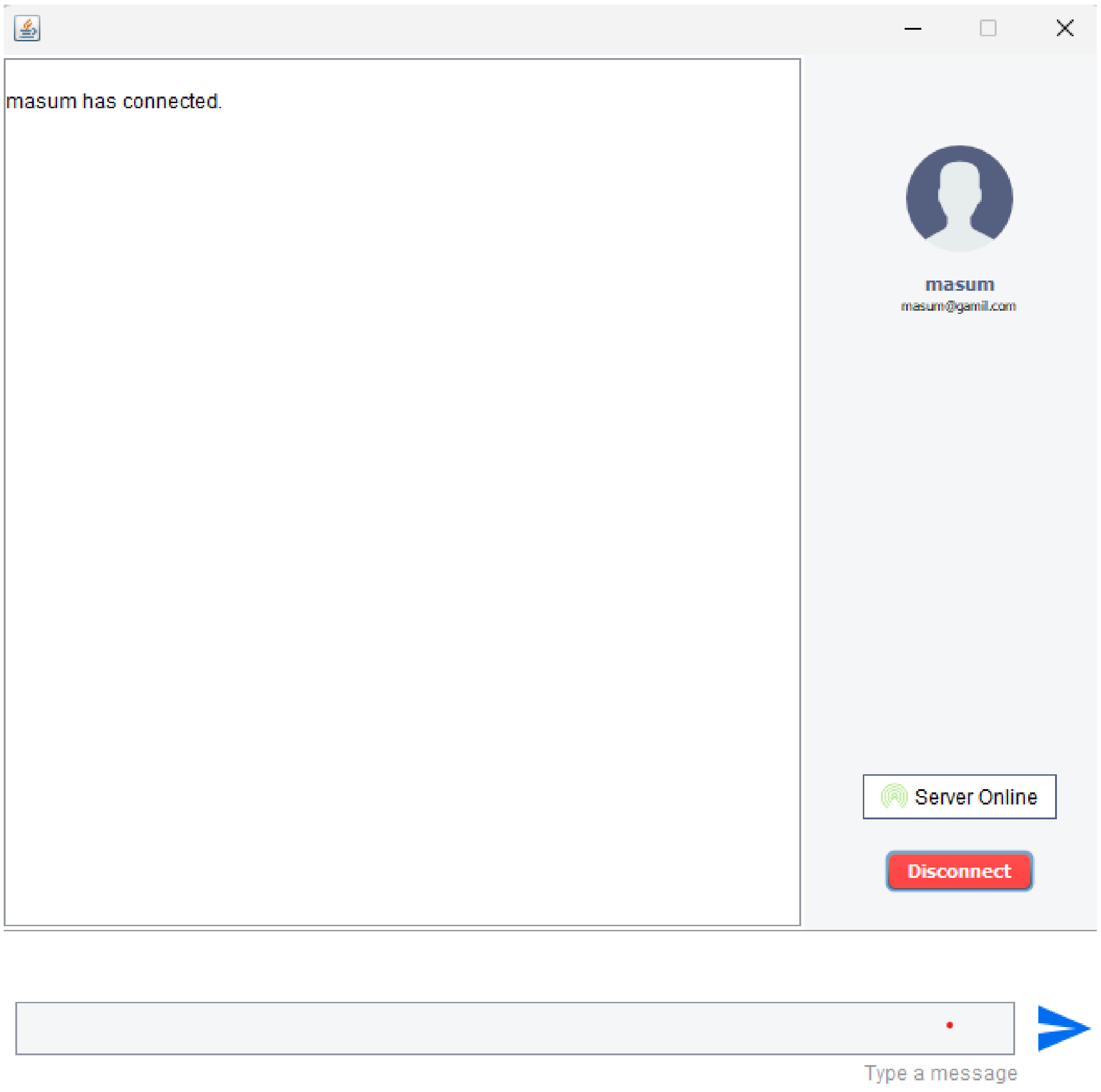
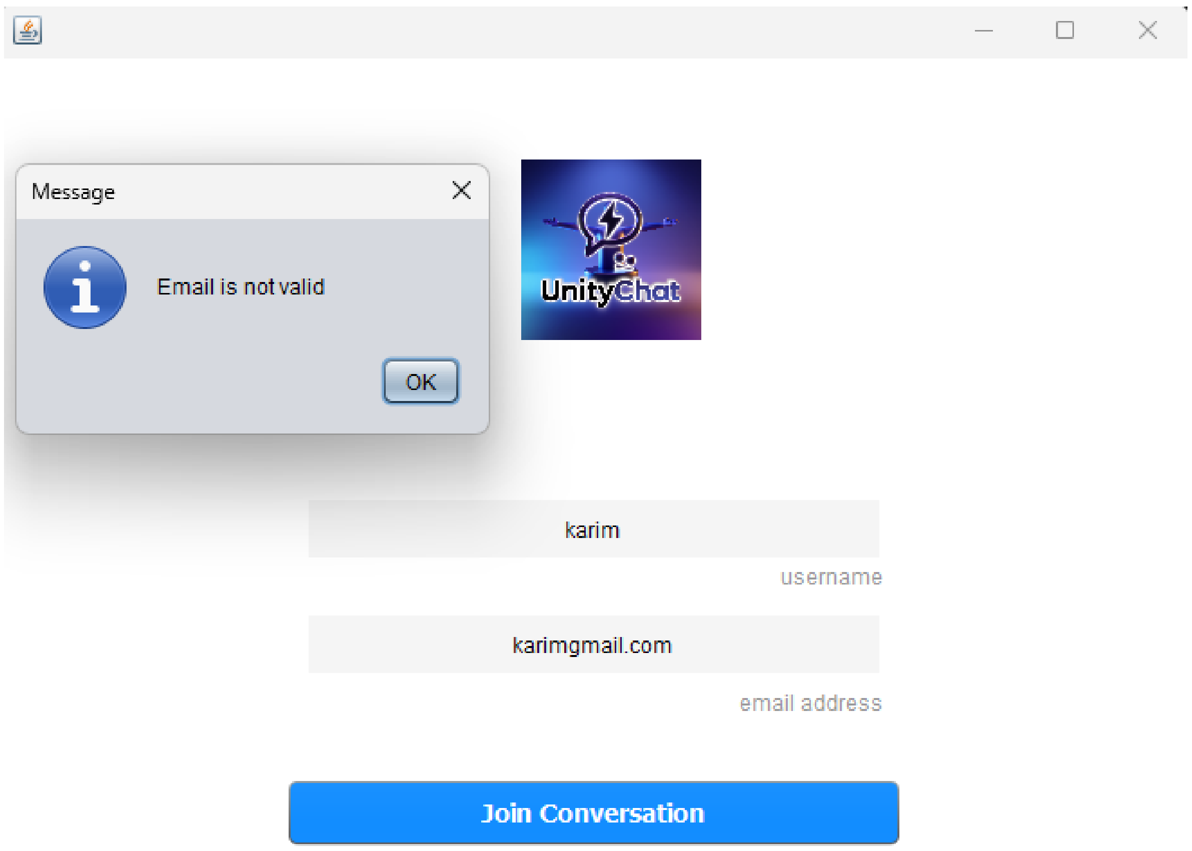

masum has connected.

Figure 3.1: USER connected after a successful registration.

Pop up message will warn if proper Gmail structure isn't followed.
Showing the successful users connected to the group chat.
.


Message

Email is not valid

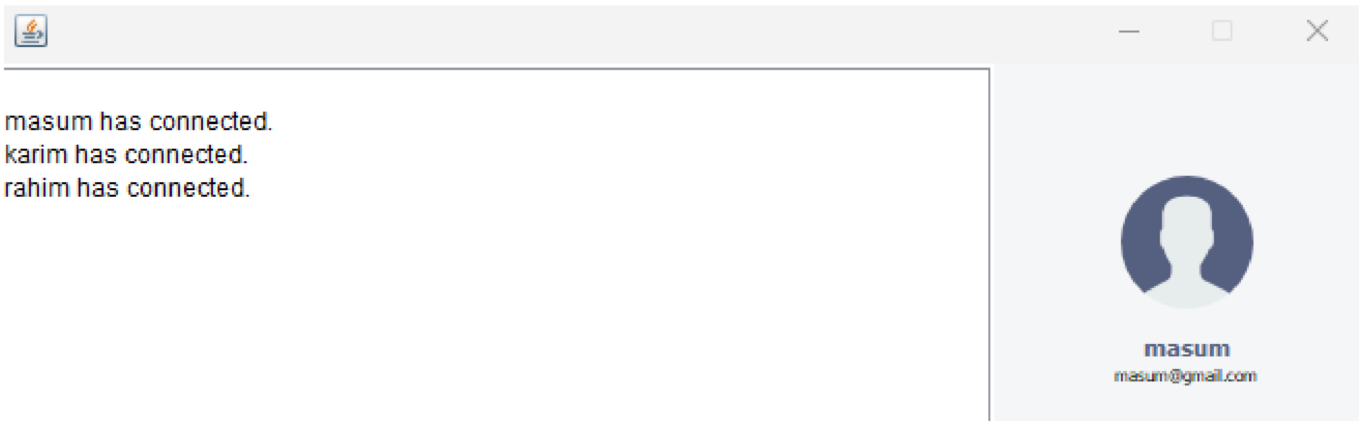
karim

username

karimgmail.com

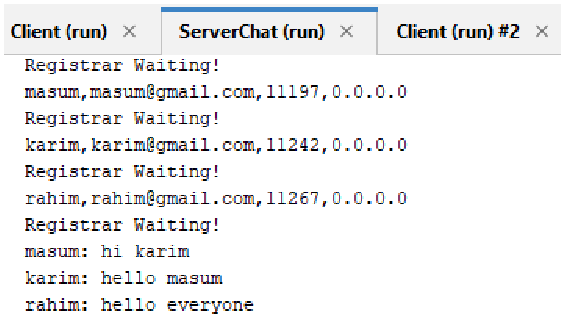
email address

Figure 3.2: Unsuccessful Registration with Showing Warning Message.


```
masum has connected.
karim has connected.
rahim has connected.
```

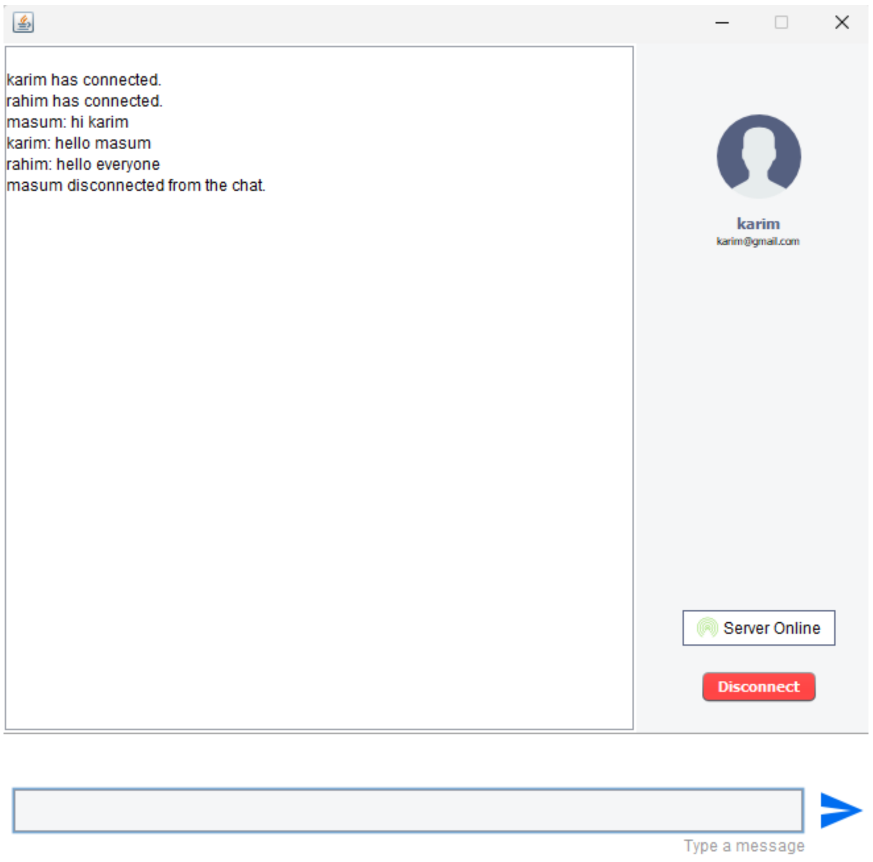
masum
masum@gmail.com

Figure 3.3: Showing all users connected successfully who have successully registered.



```
Registrar Waiting!
masum,masum@gmail.com,11197,0.0.0.0
Registrar Waiting!
karim,karim@gmail.com,11242,0.0.0.0
Registrar Waiting!
rahim,rahim@gmail.com,11267,0.0.0.0
Registrar Waiting!
masum: hi karim
karim: hello masum
rahim: hello everyone
```

Fig 3.4: Showing the port assigned to the registered user as well as messages.



```
karim has connected.
rahim has connected.
masum: hi karim
karim: hello masum
rahim: hello everyone
masum disconnected from the chat.
```

Fig 3.5: After disconnecting the user will return to the sign up page again.

### 3.2.4 Circular Buffer Efficiency

The circular buffer implemented for storing the last 10 messages was tested for reliability and speed. Results indicated:

- Message Storage: Messages were accurately stored and retrieved.
- **Efficiency:** The time to overwrite older messages remained constant, regardless of buffer size.

## 3.3 Overall Discussion

The results confirmed that UnityChat is a robust, efficient, and reliable group messaging application. The real-time performance metrics validate its effectiveness for environments requiring instant communication. The scalability tests further highlighted its suitability for both small and medium-sized groups. However, potential performance bottlenecks in larger networks or under high traffic conditions require further exploration.

# Chapter 4

# Conclusion

## 4.1 Discussion

The UnityChat project successfully addressed the need for a lightweight and efficient group messaging application. The combination of Java Swing for the user interface and Java networking for communication resulted in a robust and scalable solution. The integration of features such as real-time messaging, a circular buffer for message storage, and an admin interface for monitoring ensures the application's relevance in various real-world applications.

## 4.2 Limitations

While the project achieved its primary objectives, certain limitations were observed:

- **Limited Scalability:** The server's performance might degrade with a significantly larger number of clients or under high traffic conditions.
- **LAN Dependency:** The application is optimized for local network environments, limiting its utility for wider geographical scales without additional configurations.
- **Basic GUI:** Although functional, the GUI could be further improved for better aesthetics and user experience.

## 4.3 Scope of Future Work

The project lays a solid foundation for further development and enhancements:

- **Cloud Integration:** Enabling cloud-based servers to facilitate global communication.
- **Enhanced Security:** Adding encryption protocols to ensure secure communication.

- **Mobile Application:** Developing a mobile-friendly version to broaden its user base.
- **Advanced Features:** Implementing features such as file sharing, group management, and message search functionality.

This concludes the detailed implementation, performance evaluation, and analysis of the UnityChat project, showcasing its potential as a reliable and efficient group messaging platform for various collaborative needs.

# References

My full project on Github : https://github.com/masum6268/computer-networking-lab/tree/main/Project

For GUI Development:
Oracle Java Swing Documentation
URL: https://docs.oracle.com/javase/tutorial/uiswing/
Description: Comprehensive guide to developing user interfaces using Java Swing, covering components, layouts, and event handling.

For Socket Programming:
Java Networking and Proxies - The Basics of Sockets
URL: https://docs.oracle.com/javase/tutorial/networking/sockets/
Description: Official Oracle documentation explaining the fundamentals of socket programming in Java, including examples for client-server communication.