# Server-Side Socket Programming


Group 2:

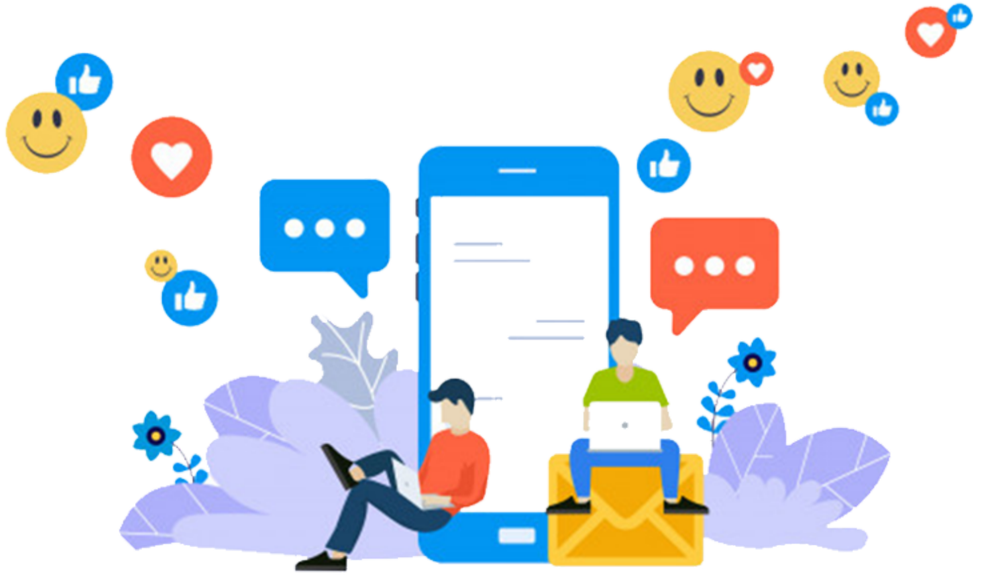
Izaaz Rahman Akbar (21/472855/PA/20348)
Matthew Tan (21/478240/PA/20736)
Rabbani Nur Kumoro (21/472599/PA/20310)
William Hilmy Susatyo (21/472585/PA/20380)

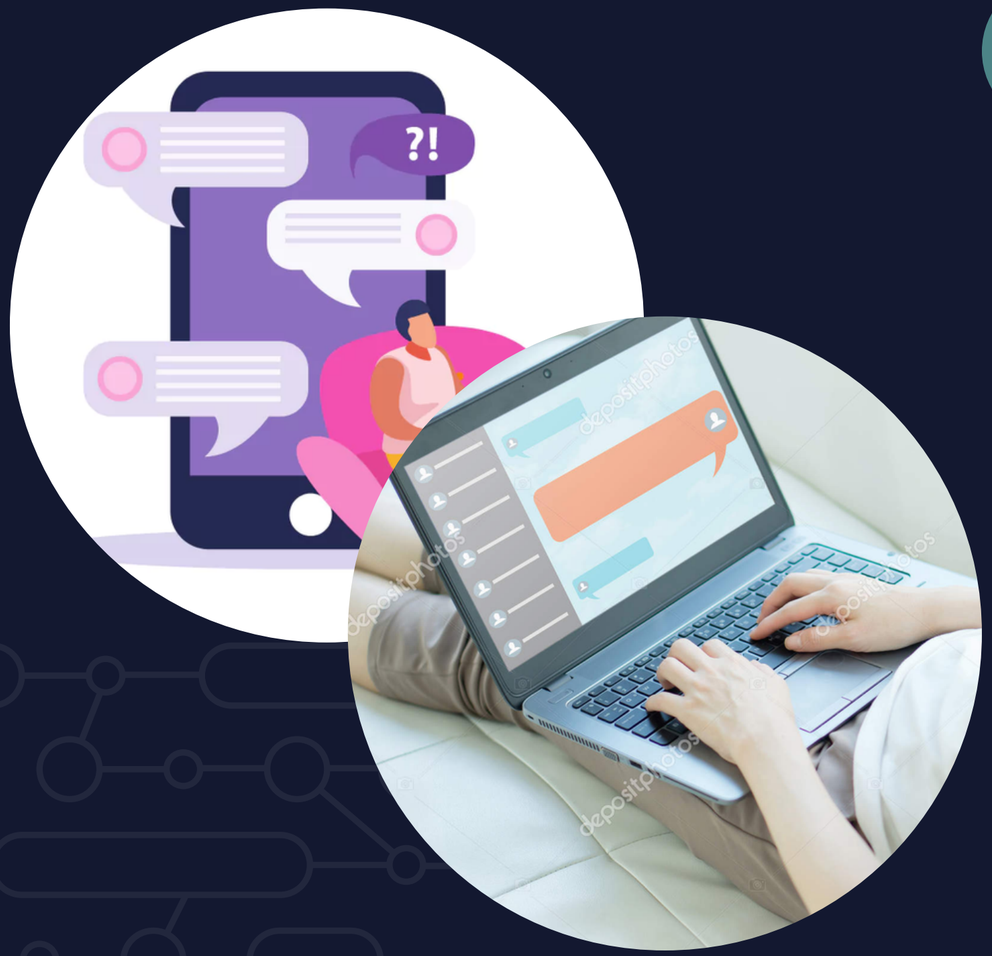

GitHub Link : **https://github.com/Easy1000/socket-programming**

# Introduction

Socket programming is a way for computers to communicate with each other over a network using a specific set of rules. Server-side socket programming refers to the use of sockets on a computer acting as a server in a client-server model.

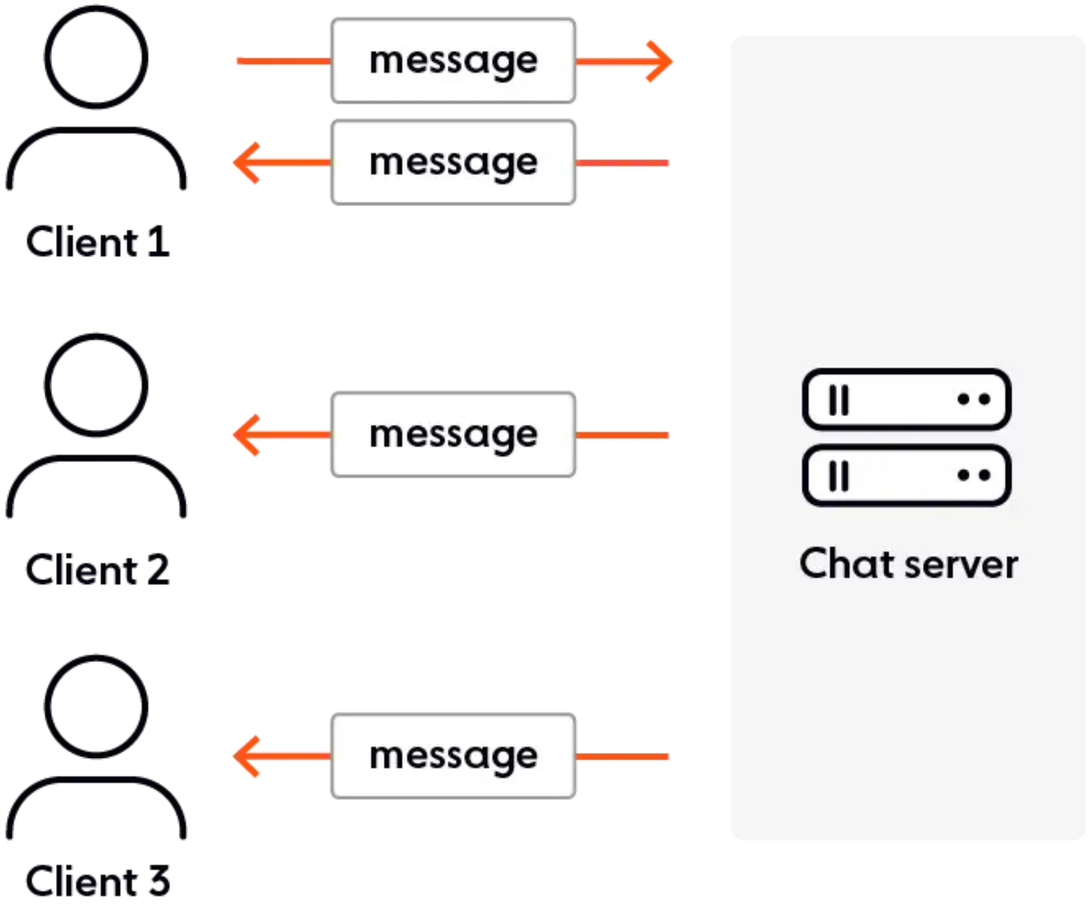
This project is an example of a multiple client chat server. We attempt to create server-side of socket programming by connecting multiple clients to a server so that the clients could communicate within each other. There will be sockets both on the server and the client's side.






# ARCHITECTURE

Sockets are used for two-way communication between programs on a network. In the Client-Server architecture, there are two sockets: one representing the client and one representing the server.




# PROTOCOL

In server-side socket programming, a protocol is a set of rules that govern the communication between the server and the client. Protocols define the format of the messages that are exchanged between the server and the client, as well as the actions that are taken based on the messages. in our case, the message type will be a string of characters. In addition, the client needs to specify another user's username anywhere in the message so that the message will be sent to that specific user.

Transmission Control Protocol (TCP): TCP is a reliable, connection-oriented protocol that ensures that all data is delivered to the destination. It is often used for applications that require reliable, error-free communication, such as file transfer and email.

# CLIENT.JAVA

The Client class has the following methods and fields:

- **Socket socket**
- **BufferedReader bufferedReader**
- **BufferedWriter bufferedWriter**
- **String username**
- **Client(Socket socket, String username)**
- **void sendMessage()**
- **void listenForMessage()**
- **void closeEverything(Socket socket, BufferedReader bufferedReader, BufferedWriter bufferedWriter)**
- **public static void main(String[] args)**

```
private BufferedReader bufferedReader;
private BufferedWriter bufferedWriter;
private String username;

public Client(Socket socket, String username) {
  try {
    this.socket = socket;
    this.bufferedWriter =
      new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
    this.bufferedReader =
      new BufferedReader(new InputStreamReader(socket.getInputStream()));
    this.username = username;
  } catch (IOException e) {
    closeEverything(socket, bufferedReader, bufferedWriter);
  }
}

public void sendMessage() {
  try {
    bufferedWriter.write(username);
    bufferedWriter.newLine();
    bufferedWriter.flush();

    Scanner scanner = new Scanner(System.in);
    while (socket.isConnected()) {
      String messageToSend = scanner.nextLine();
      bufferedWriter.write(messageToSend);
      bufferedWriter.newLine();
      bufferedWriter.flush();
    }
  } catch (IOException e) {
    closeEverything(socket, bufferedReader, bufferedWriter);
  }
}

public void listenForMessage() {
  new Thread(
    new Runnable() {
      // @override
      public void run() {
        String msgFromGroupChat;

        while (socket.isConnected()) {
          try {
            msgFromGroupChat = bufferedReader.readLine();
            System.out.println(msgFromGroupChat);
          } catch (IOException e) {
            closeEverything(socket, bufferedReader, bufferedWriter);
          }
        }
      }
    }
```

# SERVER.JAVA

The Server class has the following methods and fields:

- **ServerSocket serverSocket**
- **Server(ServerSocket serverSocket)**
- **void startServer()**
- **void closeServerSocket()**
- **public static void main(String[] args)**
- **java.io.IOException**
- **java.net.ServerSocket**
- **java.net.Socket. IOException**

```
public Server(ServerSocket serverSocket) {
  this.serverSocket = serverSocket;
}

public void startServer() {
  try {
    while (!serverSocket.isClosed()) {
      Socket socket = serverSocket.accept();
      System.out.println("A new client has connected");
      ClientHandler clientHandler = new ClientHandler(socket);
      Thread thread = new Thread(clientHandler);
      thread.start();
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}

public void closeServerSocket() {
  try {
    if (serverSocket != null) {
      serverSocket.close();
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}

public static void main(String[] args) {
  try {
    ServerSocket serverSocket = new ServerSocket(3000);
    Server server = new Server(serverSocket);
    server.startServer();
  } catch (IOException e) {
    e.printStackTrace();
```

# CLIENTHANDLER.JAVA

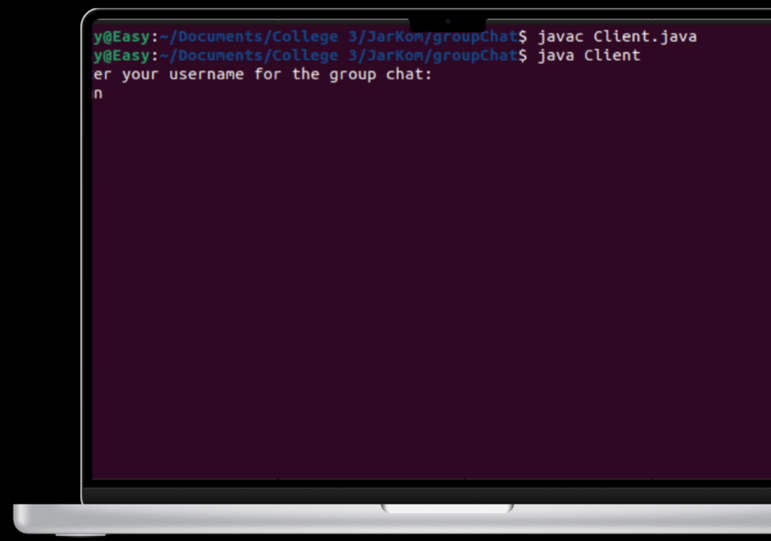
The ClientHandler class has the following methods and fields:

- **static ArrayList<ClientHandler> clientHandlers**
- **Socket socket**
- **BufferedReader bufferedReader**
- **BufferedWriter bufferedWriter**
- **ClientHandler(Socket socket)**
- **public void run()**
- **void directMessage(String messageToSend)**
- **void removeClientHandler()**
- **void closeEverything(Socket socket, BufferedReader bufferedreader, BufferedWriter bufferedWriter)**

```
private Socket socket;
private BufferedReader bufferedReader;
private BufferedWriter bufferedWriter;
private String clientUsername;

public ClientHandler(Socket socket) {
  try {
    this.socket = socket;
    this.bufferedWriter =
      new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
    this.bufferedReader =
      new BufferedReader(new InputStreamReader(socket.getInputStream()));
    this.clientUsername = bufferedReader.readLine();
    clientHandlers.add(this);
  } catch (IOException e) {
    closeEverything(socket, bufferedReader, bufferedWriter);
  }
}

//@override
public void run() {
  String messageFromClient;
  while (socket.isConnected()) {
    try {
      messageFromClient = bufferedReader.readLine();
      directMessage(messageFromClient);
    } catch (IOException e) {
      closeEverything(socket, bufferedReader, bufferedWriter);
      break;
    }
  }
}

public void directMessage(String messageToSend) {
  for (ClientHandler clientHandler : clientHandlers) {
    System.out.println(clientHandler.clientUsername);
    try {
      if (
        messageToSend.contains(clientHandler.clientUsername) &&
        !clientHandler.clientUsername.equals(this.clientUsername)
      ) {
        clientHandler.bufferedWriter.write(
          this.clientUsername + ": " + messageToSend
        );
        clientHandler.bufferedWriter.newLine();
        clientHandler.bufferedWriter.flush();
      }
    } catch (IOException e) {
      closeEverything(socket, bufferedReader, bufferedWriter);
    }
  }
}
```

# HOW TO USE IT?

- COMPILE AND RUN THE SERVER.JAVA FILE
- PUT THE LOCAL IP ADDRESS AND PORT OF THE SERVER IN THE MAIN METHOD OF CLIENT.JAVA AND COMPILE AND RUN THE CLIENT.JAVA FILE
- ENTER THE USERNAME FOR THE CLIENT
- LET OTHER USER CONNECT TO THE SERVER AND START THEIR CLIENT
- DIRECTLY MESSAGE OTHER USER BY SPECIFYING THE USERNAME ANYWHERE IN THE MESSAGE
- WELL DONE, YOU'RE ABLE TO MESSAGE A SPECIFIC USER USING THE PROGRAM




# REFERENCES


**01**

(Hanum S; Romie, O; Abdurahman, “Multi-network Transmission Using Socket Programming to Support Command and Control Systems”, AISC, Vol. 1291, p. 59-68. 2021

**02**

Pankaj. 2022. Java Socket Programming - Socket Server, Client example. [online] digitalocean.com. Available at: <https://www.digitalocean.com/community/tutorials/java-socket-programming-server-client> (Accessed: November 6, 2022).

**03**

Ashley, Apondi. 2021. Understanding Socket Programming in Java. [online] section.io. Available at: <https://www.section.io/engineering-education/socket-programming-in-java/>
(Accessed: November 24, 2022}


# DANK U WEL

VOOR HET KIJKEN VAN ONS PRESENTATIE