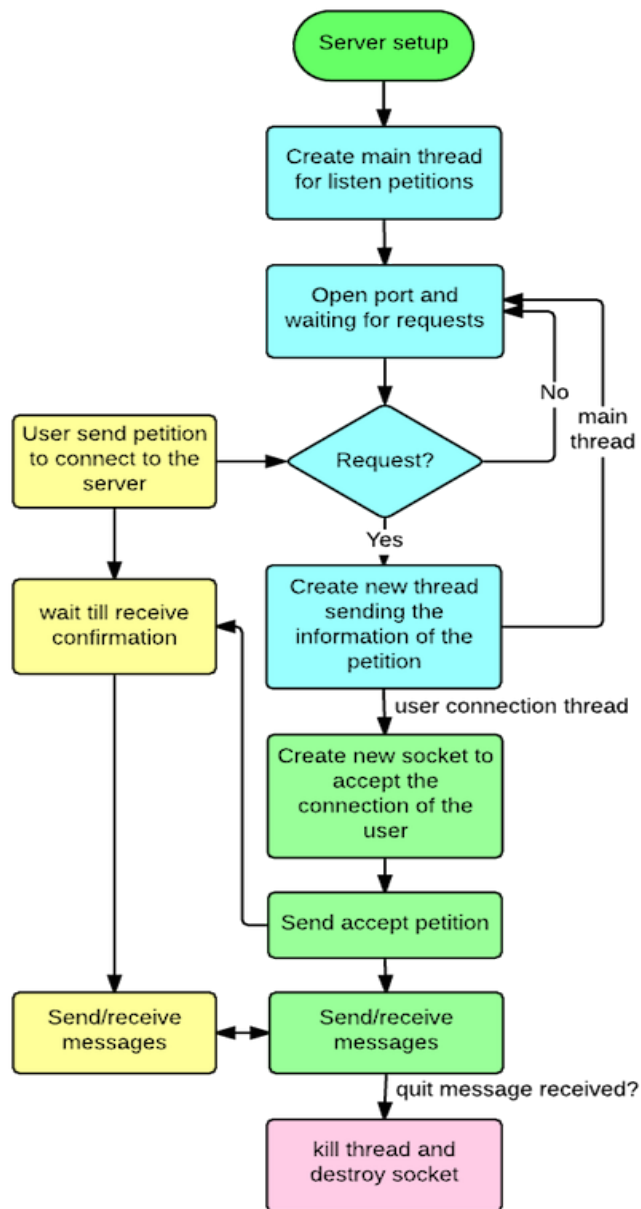# Computer Networks

# Report

**1)Project Analysis**

TCP chat system using socket programming

**AIM:**

- To make a fully functional TCP Chat server
- Multiple clients can connect, each with a unique username decided via prompt at connection time.
- Each user's message will be displayed along with the date, time, IP address.
- Choose multiple user limit equal to or above 10

**2) Design(Flow chart for your implemented project)**

The server setup creates a main thread localhost port opens and waits for the request from the clients. If the request gets accepted then the client sends the information. A user connection thread is made to create a new socket for data transmission. If the request or the petition is accepted then it connects to the server and sends messages. After the communication is done the thread is killed and the socket gets destroyed. The project establishes and helps to communicate with multiple client instances on single server localhost.

```mermaid
flowchart TD
    Start([Server setup])
    A[Create main thread for listen petitions]
    B[Open port and waiting for requests]
    C{Request?}
    D[User send petition to connect to the server]
    E[Create new thread sending the information of the petition]
    F[wait till receive confirmation]
    G[Create new socket to accept the connection of the user]
    H[Send accept petition]
    I[Send/receive messages]
    J[Send/receive messages]
    K[kill thread and destroy socket]
```

Server setup

Create main thread for listen petitions

Open port and waiting for requests

Request? — No — main thread

User send petition to connect to the server

Yes

Create new thread sending the information of the petition

wait till receive confirmation

user connection thread

Create new socket to accept the connection of the user

Send accept petition

Send/receive messages ⟷ Send/receive messages

quit message received?

kill thread and destroy socket

## 3) Requirements of your project(to run)

The project's code is written in Java. There are two files Server.java and Client.java.

Requirements of the project to run:
1. Users must have JDK installed along with java's latest version.
2. Use cmd to run the code for server.java file
   a. Then multiple clients can be created by running different instances of the client file
   b. When the client file is run user has to enter a unique username to be able to see the chats of multiple clients.

## 5) Steps to execute the complete project

Steps to run the project:
   1) Run the server.java file
      a. javac server.java
   2) In order to join with a new client run client.java file
      b. javac client.java
   3) The user has to keep running various instances of the client.java file in order to create multiple users.

## 6) Outputs for code

**\*\*We are only attaching the output code for three users but it can always be extended.\*\***

```java
import java.net.Socket;
import java.util.Scanner;
import java.io.*;
import java.net.*;


public class Client implements Runnable {

    Socket socketConnection;
    DataOutputStream outToServer;
    DataInputStream din;


    Client() throws UnknownHostException, IOException {

        socketConnection = new Socket("127.0.0.1", 8000);
        outToServer = new DataOutputStream(socketConnection.getOutputStream());
        din = new DataInputStream(socketConnection.getInputStream());

        Thread thread;
        thread = new Thread(this);
        thread.start();

        BufferedReader br = null;
        String ClientName = null;
        Scanner input = new Scanner(System.in);
        String SQL = "";
        try {
            System.out.print("Enter you name: ");
            ClientName = input.next();
            ClientName += ": ";
            //QUERY PASSING

            br = new BufferedReader(new InputStreamReader(System.in));
            while (!SQL.equalsIgnoreCase("exit")) {
                System.out.println();
                System.out.print(ClientName);

                SQL = br.readLine();
                outToServer.writeUTF(ClientName + SQL);
            }

        } catch (Exception e) {
            System.out.println(e);
        }
    }

    //Run | Debug
    public static void main(String[] arg) throws UnknownHostException, IOException {
        Client client = new Client();
    }

    public void run() {
        while (true) {
            try {
                System.out.println("\n" + din.readUTF());

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```java
1    import java.io.IOException;
2    import java.net.*;
3    import java.util.Vector;
4    import java.io.*;
5    import java.util.*;
6    import java.io.DataInputStream;
7    import java.net.ServerSocket;
8    import java.net.Socket;
9    import java.time.LocalDateTime;
10   import java.util.logging.Level;
11   import java.util.logging.Logger;
12   import java.text.*;
13   import java.util.Scanner;
14
15
16   public class Server {
17       static Vector < Socket > ClientSockets;
18       int clientCount = 0;
19       //int i = 0;
20
21
22       Server() throws IOException {
23           Date dNow = new Date();
24           System.out.println("MultiThreadServer started at " + String.format("%tc", dNow));
25           System.out.println();
26
27           ServerSocket server = new ServerSocket(8000);
28           ClientSockets = new Vector < Socket > ();
29
30
31           while (true) {
32               Socket client = server.accept();
33               AcceptClient acceptClient = new AcceptClient(client);
34               System.out.println("Connection from Socket " + "[addr = " + client.getLocalAddress() + ",port = " +
35                   client.getPort() + ",localport = " + client.getLocalPort() + "] at " +
36                   String.format("%tc", dNow));
37               System.out.println();
38               //System.out.println(clientCount);
39
40           }
41           //server.close();
42       }
43
     Run | Debug
44       public static void main(String[] args) throws IOException {
45           Server server = new Server();
46       }
47
48       class AcceptClient extends Thread {
49           Socket ClientSocket;
50           DataInputStream din;
51           DataOutputStream dout;
52
53           AcceptClient(Socket client) throws IOException {
54               ClientSocket = client;
55               din = new DataInputStream(ClientSocket.getInputStream());
56               dout = new DataOutputStream(ClientSocket.getOutputStream());
57               clientCount++;
58               ClientSockets.add(ClientSocket);
59               start();
60           }
61
62           public void run() {
63
64               try {
65                   while (true) {
66                       String msgFromClient = din.readUTF();
67                       System.out.println(msgFromClient);
68                       for (int i = 0; i < ClientSockets.size(); i++) {
69                           Socket pSocket = (Socket) ClientSockets.elementAt(i);
70                           DataOutputStream pOut = new DataOutputStream(pSocket.getOutputStream());
71                           pOut.writeUTF(msgFromClient);
72                           pOut.flush();
73                       }
74                   }
75
76               } catch (IOException e) {
77                   e.printStackTrace();
78               }
79
80           }
81       }
```

**Group Work:**

We all thoroughly went through the socket programming. We decided to have several meetings over how are we gonna make it as asked and how will we improve it. After having several meetings we decided to have Java as the primary language which we will code. Each member went through socket programming and made their own contribution to the project.