**ROLL NO. : 2019450046 BATCH: D DATE: 09/09/2021**

**NAME: Ayush Sah**

**EXPERIMENT NO: 01**

**EXPERIMENT TITLE:** Implement Chat application using socket programming

**Objective:**

Implement and test a Chat Application based on client-server distributed system

**Problem statement**

Suppose we are requested to create a distributed Chat application for communicating, that can be run on any client machine. It must do the following, any client can communicate with the server, any client can communicate with the other clients, the Server is able to send message to the Client.

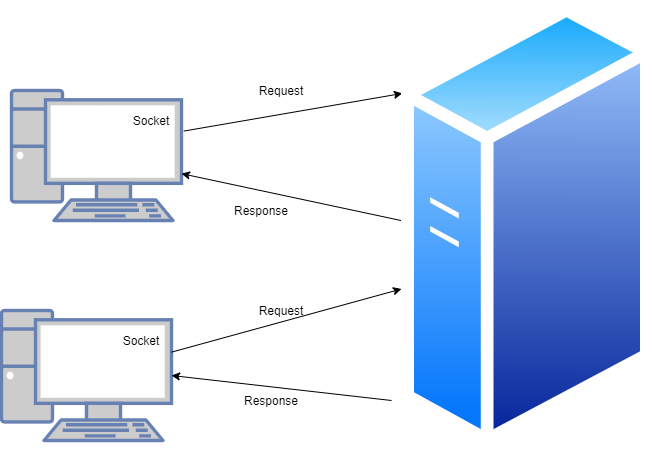
**Application analysis and design:**

To create a network of system to communicate we need to have followed components as part of the network.

1. **Server:** Server is a system which acts as a response mechanism to the client. A network server is a computer designed to act as central repository and help in providing various resources to other computers in the network. These other computers are generally clients who request for these resources.
2. **Client:** A client is a computer hardware device or software that accesses a service made available by a server.
3. **Network:** A connection of more than one client and one or more server is referred to as Network. A computer network is a group of computers interconnected using networks.
4. **Socket:** A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.
5. **Port:** Port is specific to an application where the message is designated to be sent. These numbers are a set of connection points paired along with a socket.

**Deliverables:**

**Conceptual architecture of the distributed system**

* A client communicates with another client by sending the message to server and then server passing the message to the destination client.
* The server acts as an intermediate between two clients. The server can perform operations and return the result to the client.****

**Source Code –**

**Server.java**

package Exp1.ChatApplication;  
  
import java.net.\*;  
import java.util.\*;  
import java.io.\*;  
  
public class Server  
{  
 private DataInputStream input = null;  
 static Vector *ClientSockets*;  
 static Vector *LoginNames*;  
  
 public Server() throws Exception  
 {  
 input = new DataInputStream(System.*in*);  
 ServerSocket soc=new ServerSocket(5219);  
 *ClientSockets*=new Vector();  
 *LoginNames*=new Vector();  
  
 while(true)  
 {  
 Socket CSoc=soc.accept();  
 AcceptClient obClient=new AcceptClient(CSoc);  
 }  
 }  
 public static void main(String args[]) throws Exception  
 {  
  
 Server ob=new Server();  
 }  
  
 class AcceptClient extends Thread  
 {  
 Socket ClientSocket;  
 DataInputStream din;  
 DataOutputStream dout;  
 AcceptClient (Socket CSoc) throws Exception  
 {  
 ClientSocket=CSoc;  
  
 din=new DataInputStream(ClientSocket.getInputStream());  
 dout=new DataOutputStream(ClientSocket.getOutputStream());  
  
 String LoginName=din.readUTF();  
  
 System.*out*.println("User Logged In :" + LoginName);  
 *LoginNames*.add(LoginName);  
 *ClientSockets*.add(ClientSocket);  
 start();  
 }  
  
 public void run()  
 {  
 while(true)  
 {  
  
 try  
 {  
 String msgFromClient=new String();  
 msgFromClient=din.readUTF();  
 StringTokenizer st=new StringTokenizer(msgFromClient);  
 String Sendto=st.nextToken();  
 String MsgType=st.nextToken();  
 int iCount=0;  
  
 if(MsgType.equals("LOGOUT"))  
 {  
 for(iCount=0;iCount<*LoginNames*.size();iCount++)  
 {  
 if(*LoginNames*.elementAt(iCount).equals(Sendto))  
 {  
 *LoginNames*.removeElementAt(iCount);  
 *ClientSockets*.removeElementAt(iCount);  
 System.*out*.println("User " + Sendto +" Logged Out ...");  
 break;  
 }  
 }  
 }  
 else  
 {  
 String msg="";  
 //String line="";  
 while(st.hasMoreTokens())  
 {  
 msg=msg+" " +st.nextToken();  
 System.*out*.println(msg);  
  
 }  
 for(iCount=0;iCount<*LoginNames*.size();iCount++)  
 {  
 if(*LoginNames*.elementAt(iCount).equals(Sendto))  
 {  
 Socket tSoc=(Socket)*ClientSockets*.elementAt(iCount);  
 DataOutputStream tdout=new DataOutputStream(tSoc.getOutputStream());  
 tdout.writeUTF(msg);  
 break;  
 }  
 }  
 String line="";  
 if(iCount==*LoginNames*.size())  
 {  
 line=input.readLine();  
 dout.writeUTF(line);  
 }  
  
 }  
 if(MsgType.equals("LOGOUT"))  
 {  
 break;  
 }  
  
 }  
 catch(Exception ex)  
 {  
 ex.printStackTrace();  
 }  
 }  
 }  
 }  
}

**Client.java**

package Exp1.ChatApplication;  
  
import java.awt.\*;  
import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.net.Socket;  
  
public class Client extends Frame implements Runnable {  
 Socket soc;  
 TextField tf;  
 TextArea ta;  
 Button btnSend, btnClose;  
 String sendTo;  
 String LoginName;  
 Thread t = null;  
 DataOutputStream dout;  
 DataInputStream din;  
  
 Client(String LoginName, String chatwith) throws Exception {  
 //input = new DataInputStream(System.in);  
 super(LoginName);  
 this.LoginName = LoginName;  
 sendTo = chatwith;  
 tf = new TextField(50);  
 ta = new TextArea(50, 50);  
 btnSend = new Button("Send");  
 btnClose = new Button("Close");  
 soc = new Socket("127.0.0.1", 5219);  
  
 din = new DataInputStream(soc.getInputStream());  
 dout = new DataOutputStream(soc.getOutputStream());  
 dout.writeUTF(LoginName);  
  
 t = new Thread(this);  
 t.start();  
  
 }  
  
 void setup() {  
 setSize(600, 400);  
 setLayout(new GridLayout(2, 1));  
  
 add(ta);  
 Panel p = new Panel();  
  
 p.add(tf);  
 p.add(btnSend);  
 p.add(btnClose);  
 add(p);  
 show();  
 }  
  
 public boolean action(Event e, Object o) {  
 if (e.arg.equals("Send")) {  
 try {  
 dout.writeUTF(sendTo + " " + "DATA" + " " + tf.getText().toString());  
 ta.append("\n" + LoginName + " Says:" + tf.getText().toString());  
 tf.setText("");  
 } catch (Exception ex) {  
 }  
 } else if (e.arg.equals("Close")) {  
 try {  
 dout.writeUTF(LoginName + " LOGOUT");  
 System.*exit*(1);  
 } catch (Exception ex) {  
 }  
  
 }  
  
 return super.action(e, o);  
 }  
  
 public static void main(String args[]) throws Exception {  
  
 Client Client1 = new Client("Ayush", "Sarvar");  
 Client1.setup();  
 }  
  
 public void run() {  
 while (true) {  
 try {  
  
 StringBuilder s = new StringBuilder(din.readUTF());  
 int c = 0;  
 StringBuilder stringBuilder = s;  
  
 ta.append("\n" + sendTo + " Says :" + stringBuilder);  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
 }  
}

**Client2.java**

package Exp1.ChatApplication;  
  
import java.awt.\*;  
import java.io.DataInputStream;  
import java.io.DataOutputStream;  
import java.net.Socket;  
  
public class Client2 extends Frame implements Runnable {  
 Socket soc;  
 TextField tf;  
 TextArea ta;  
 Button btnSend, btnClose;  
 String sendTo;  
 String LoginName;  
 Thread t = null;  
 DataOutputStream dout;  
 DataInputStream din;  
  
 Client2(String LoginName, String chatwith) throws Exception {  
 //input = new DataInputStream(System.in);  
 super(LoginName);  
 this.LoginName = LoginName;  
 sendTo = chatwith;  
 tf = new TextField(50);  
 ta = new TextArea(50, 50);  
 btnSend = new Button("Send");  
 btnClose = new Button("Close");  
 soc = new Socket("127.0.0.2", 5219);  
  
 din = new DataInputStream(soc.getInputStream());  
 dout = new DataOutputStream(soc.getOutputStream());  
 dout.writeUTF(LoginName);  
  
 t = new Thread(this);  
 t.start();  
  
 }  
  
 void setup() {  
 setSize(600, 400);  
 setLayout(new GridLayout(2, 1));  
  
 add(ta);  
 Panel p = new Panel();  
  
 p.add(tf);  
 p.add(btnSend);  
 p.add(btnClose);  
 add(p);  
 show();  
 }  
  
 public boolean action(Event e, Object o) {  
 if (e.arg.equals("Send")) {  
 try {  
 dout.writeUTF(sendTo + " " + "DATA" + " " + tf.getText().toString());  
 ta.append("\n" + LoginName + " Says:" + tf.getText().toString());  
 tf.setText("");  
 } catch (Exception ex) {  
 }  
 } else if (e.arg.equals("Close")) {  
 try {  
 dout.writeUTF(LoginName + " LOGOUT");  
 System.*exit*(1);  
 } catch (Exception ex) {  
 }  
  
 }  
  
 return super.action(e, o);  
 }  
  
 public static void main(String args[]) throws Exception {  
  
 Client Client1 = new Client("Sarvar", "Ayush");  
 Client1.setup();  
 }  
  
 public void run() {  
 while (true) {  
 try {  
 ta.append("\n" + sendTo + " Says :" + din.readUTF());  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
 }  
}

**Output –**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated **Text

Description automatically generated**

**Test Procedure for objective validation :.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Description** | **Action** | **Expected Output** | **Actual Output** | **Status** |
| Start Server | Run Server.java | “Server Started” will be printed on Server Terminal | “Server Started” is printed on Server Terminal | PASS |
| Start Client 1 | Run Client 1. | Client 1 terminal displays “Connected”. | Client 1 terminal displays “Connected”. | PASS |
| Start Client 2 | Run Client 2. | Client 2 terminal displays “Connected”. | Client 2 terminal displays “Connected”. | PASS |
| Send Messages between Clients | Clients send messages to each other. | Messages displayed on both clients’ terminals. | Messages displayed on both clients’ terminals. | PASS |
| Broadcasting | Server sends a broadcast message to clients. | Broadcast messages are sent to both the clients and displayed on their terminals. | Messages displayed on both clients’ terminals. | PASS |
| Logout | Click on close to close the chat on client side. | Chat application windows gets closed | Chat terminal is closed and user logged out message is sent to the server. | PASS |
| Reverse String | Client sends string to server | Server reverse the string and send to client | Reverse String is displayed | PASS |
| Get ASCII from Server | Client sends char to server | Server responds with ASCII value | Integer value is displayed | PASS |