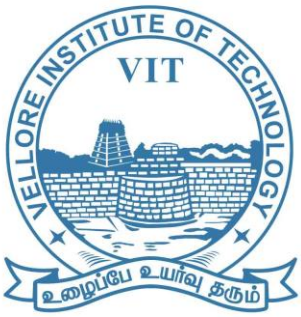


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Client – Server Application

PROJECT REPORT

Operating Systems – CSE2005

SLOT – F2

FACULTY – Lijo V.P

TEAM MEMBERS:

REG NO	NAME
18BCE2241	AMAAN ALI JAFRI
18BCE2213	SAURAV KUMARSINGH
18BCE0038	SYED AFZAL AHAMMED

CONTENTS

1. Acknowledgement

We would firstly like to acknowledge our professor Sir LIJO V.P for throughout helping us out in the project with his knowledge and experience.
We would also like to thank our batch mates for exchanging ideas with us which helped a lot in creative thinking and exploring in the topic.
We are thankful to the VIT administration for giving us this golden opportunity of trying out our own ideas on a semester level and getting a chance to execute it.
We thank the entire staff of VIT who have directly or indirectly contributed to the system for its proper functioning and giving students the freedom to analyze things and projecting their creative minds in the limelight.

BONAFIDE CERTIFICATE

This is to certify that the project "CLIENT SERVER APPLICATION" is the Bonafede work of "AMAAN ALI JAFRI (18BCE2241), SAURAV KUMAR SINGH (18BCE2213), SYED AFZAL AHAMMED (18BCE0038) in order to complete their J component for the course CSE2005 under my supervision.

PROF.LIJO V.P

Assistant Professor (Senior),

School of Computer Science and Engineering

VIT University, Vellore

632014.

3. Abstract:

Chat server is an online system created for the community of people to interact with one another on the Internet. This system provides solution to most of the shortcomings of the traditional system. The teachers, students and the firm are profited evenly by the suggested system. The system preserves a great amount of time and effort too, for both. A new software has been introduced for chatting, which is called Chat Server. This software plays a pivotal role in decreasing the interaction gap among the various people in a college, so it can be termed as very significant for the college. In our project model we have tried the implementation of a small chat client server model which sets up a server over a local host and enables various clients to connect to it and share messages. The users can connect using the client server which requests connections to the main server and after connection enables user to share messages. The project uses the concepts of multi-threading and the server uses a different thread for each client request. Thus, it enables data communication over a single client which was earlier not possible due to lack of multi-threading.

4. Introduction

The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the Wide Web

Client and server role:

The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

Servers are classified by the services they provide. For example, a web server serves web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a service.

Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer. Communication between servers, such as to synchronize data, is sometimes called inter-server or server-to-server communication.

Client and server communication

In general, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol. All client-server protocols operate in the application layer. The application layer protocol defines the basic patterns of the dialogue. To formalize the data exchange even further, the server may implement an application programming interface (API). The API is an abstraction layer for accessing a service. By restricting communication to a specific content format, it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange.

A server may receive requests from many distinct clients in a short period of time. A computer can only perform a limited number of tasks at any moment and relies on a scheduling system to prioritize incoming requests from clients to accommodate them. To prevent abuse and maximize availability, server software may limit the availability to clients. Denial of service attacks are designed to exploit a server's obligation to process requests by overloading it with excessive request rates.

5. LITERATURE SURVEY

The development & growth of distributed system & exponential growth of internet and widespread popularity of world wide web has created the possibility of education being imparted on much larger scale and has led to new avenues for distance education. Existing computer-based evaluation mechanisms, such as Web Based Testing, rely principally on the client-server model. Such mechanisms usually do not scale well and also do not fully support features like evaluation of subjective questions, delivery of dynamic content, and off-line examinations. These features are extremely desirable for distance evaluation and there is a need for alternate ways of designing such applications. This technique can be implemented in a distributed distance learning environment, which allows students or instructors to login from anywhere to a central server in an education center while still retaining the look-and-feel of personal setups. Many Universities, college, institution have started their online courses along with their regular in-house courses.

Sockets in JAVA:

A socket is the one endpoint of a two-way communication link between two programs running over the network. Running over the network means that the programs run on different computers, usually referred as the local and the remote computers. However, one can run the two programs on the same computer. Such communicating programs constitutes a client/server application. The server implements a dedicated logic, called service. The clients connect to the server to get served, for example, to obtain some data or to ask for the computation of some data. Different client/server applications implement different kind of services.

Network connection:

A network connection is initiated by a client program when it creates a socket for the communication with the server. To create the socket in Java, the client calls the Socket constructor and passes the server address and the the specific server port number to it. At this stage the server must be started on the machine having the specified address and listening for connections on its specific port number.

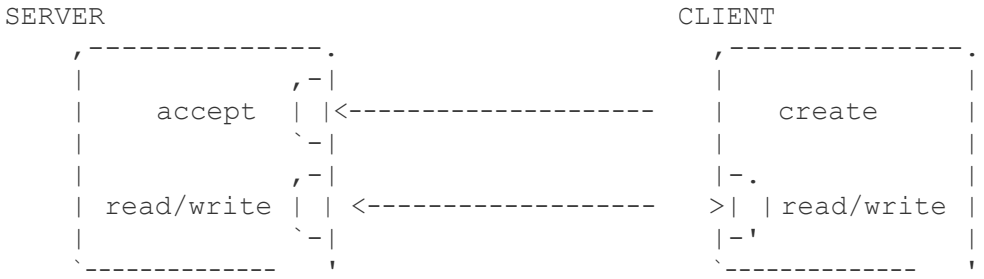
The server uses a specific port dedicated only to listening for connection requests from clients. It cannot use this specific port for data communication with the clients because the server must be able to accept the client connection at any instant. So, its specific port is dedicated only to listening for new connection requests. The server-side socket associated with specific port is called server socket. When a connection request arrives on this socket from the client side, the client and the server establish a connection. This connection is established as follows:

1. When the server receives a connection request on its specific server port, it creates a new socket for it and binds a port number to it.
2. It sends the new port number to the client to inform it that the connection is established.
3. The server goes on now by listening on two ports:
   a. it waits for new incoming connection requests on its specific port, and
   b. it reads and writes messages on established connection (on new port) with the accepted client.

The server communicates with the client by reading from and writing to the new port. If other connection requests arrive, the server accepts them in the similar way creating a new port for each new connection. Thus, at any instant, the server must be able to communicate simultaneously with many clients and to wait on the same time for incoming requests on its specific server port. The communication with each client is done via the sockets created for each communication.

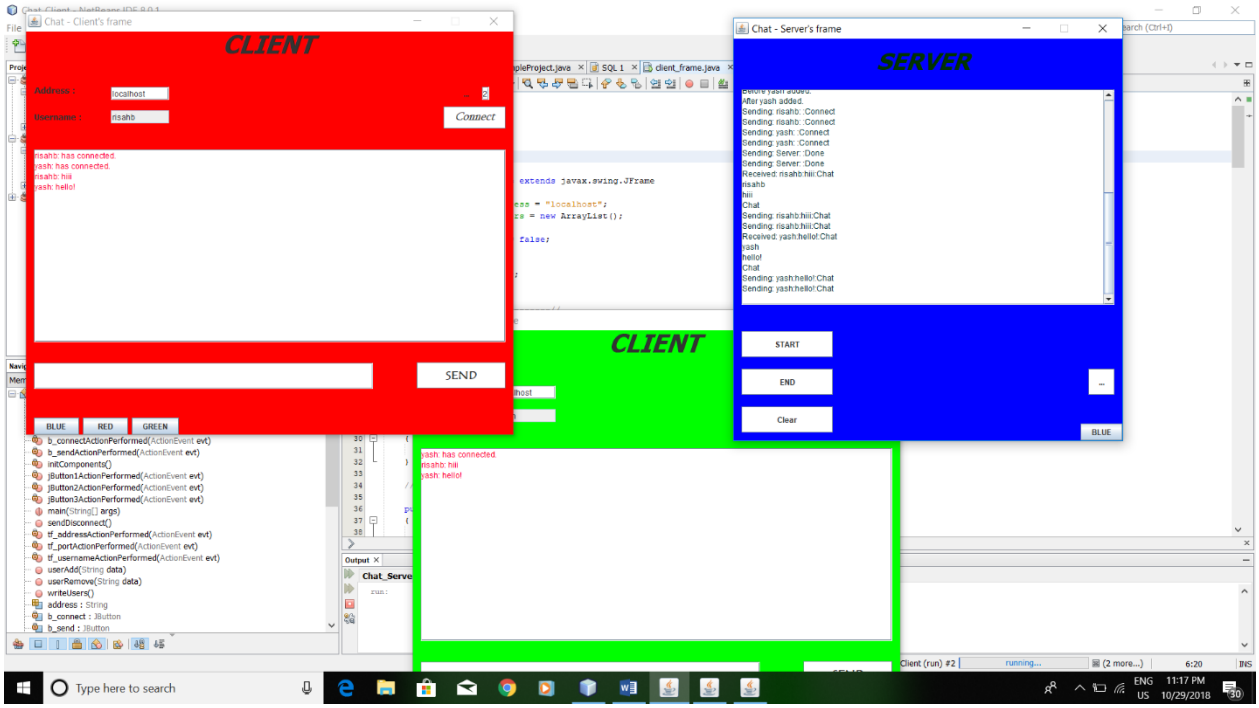
6. WORKFLOW DIAGRAM:

The client is implemented using two threads - one thread to interact with the server and the other with the standard input. Two threads are needed because a client must communicate with the server and, simultaneously, it must be ready to read messages from the standard input to be sent to the server.
The server is implemented using threads also. It uses a separate thread for each connection. It spawns a new client thread every time a new connection from a client is accepted.

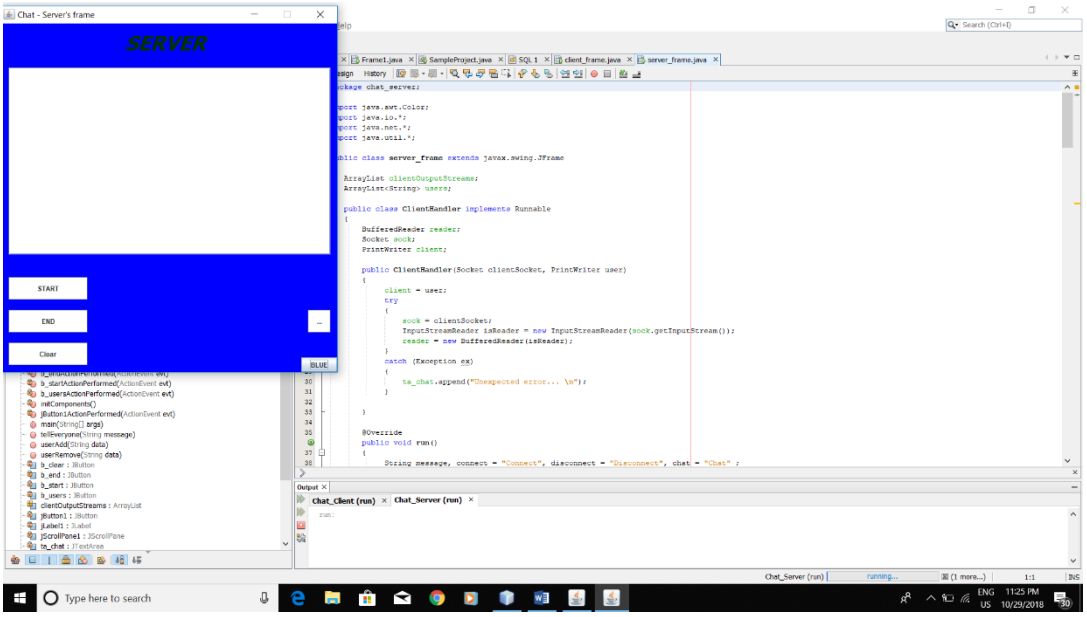


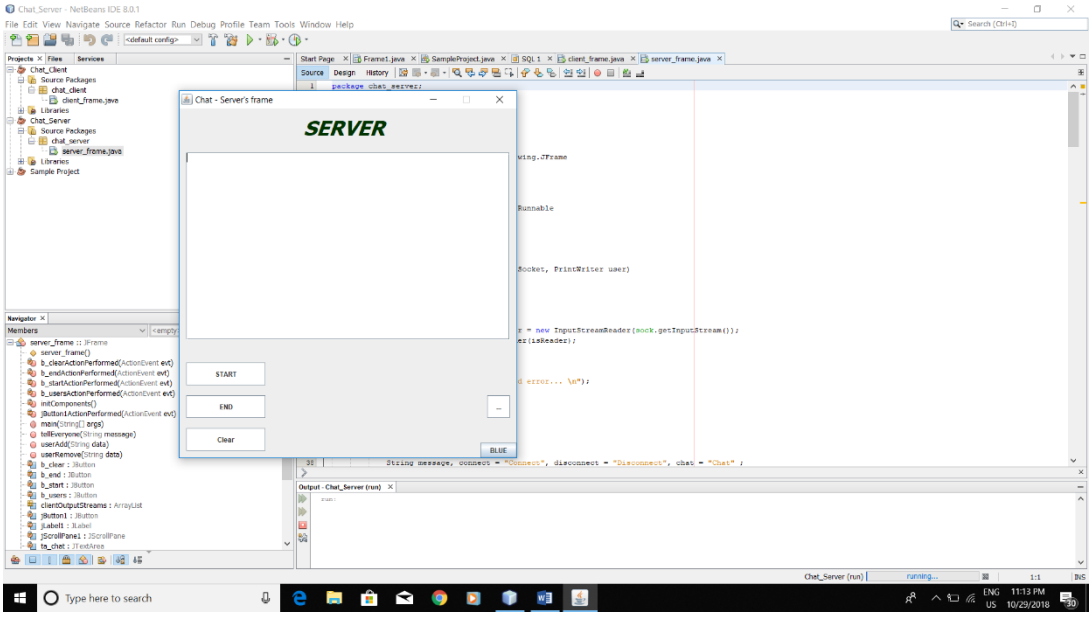
7.RESULTS AND SNAPSHOTS:

WORKING MODEL



SERVER:


Chat - Server's frame
SERVER
START
END
Clear
BLUE


Chat - Server's frame
SERVER
START
END
Clear
BLUE

Code:

```
package chat_server;

import java.io.*;
import java.net.*;
import java.util.*;

public class server_frame extends javax.swing.JFrame
{
  ArrayList clientOutputStreams;
  ArrayList<String> users;

  public class ClientHandler implements Runnable
  {
     BufferedReader reader;
     Socket sock;
     PrintWriter client;

     public ClientHandler(Socket clientSocket, PrintWriter user)
     {
        client = user;
        try
        {
           sock = clientSocket;
           InputStreamReader isReader = new InputStreamReader(sock.getInputStream());
           reader = new BufferedReader(isReader);
        }
        catch (Exception ex)
        {
           ta_chat.append("Unexpected error... \n");
        }

     }

     @Override
     public void run()
     {
        String message, connect = "Connect", disconnect = "Disconnect", chat = "Chat" ;
        String[] data;

        try
        {
           while ((message = reader.readLine()) != null)
           {
              ta_chat.append("Received: " + message + "\n");
              data = message.split(":");

              for (String token:data)
              {
                 ta_chat.append(token + "\n");
              }

              if (data[2].equals(connect))
```

```
                {
                    tellEveryone((data[0] + ":" + data[1] + ":" + chat));
                    userAdd(data[0]);
                }
                else if (data[2].equals(disconnect))
                {
                    tellEveryone((data[0] + ":has disconnected." + ":" + chat));
                    userRemove(data[0]);
                }
                else if (data[2].equals(chat))
                {
                    tellEveryone(message);
                }
                else
                {
                    ta_chat.append("No Conditions were met. \n");
                }
            }
        }
        catch (Exception ex)
        {
            ta_chat.append("Lost a connection. \n");
            ex.printStackTrace();
            clientOutputStreams.remove(client);
        }
            }
}

public server_frame()
{
    initComponents();
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
private void initComponents() {

    jScrollPane1 = new javax.swing.JScrollPane();
    ta_chat = new javax.swing.JTextArea();
    b_start = new javax.swing.JButton();
    b_end = new javax.swing.JButton();
    b_users = new javax.swing.JButton();
    b_clear = new javax.swing.JButton();
    lb_name = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("Chat - Server's frame");
    setName("server"); // NOI18N
    setResizable(false);

    ta_chat.setColumns(20);
    ta_chat.setRows(5);
    jScrollPane1.setViewportView(ta_chat);

    b_start.setText("START");
    b_start.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
```

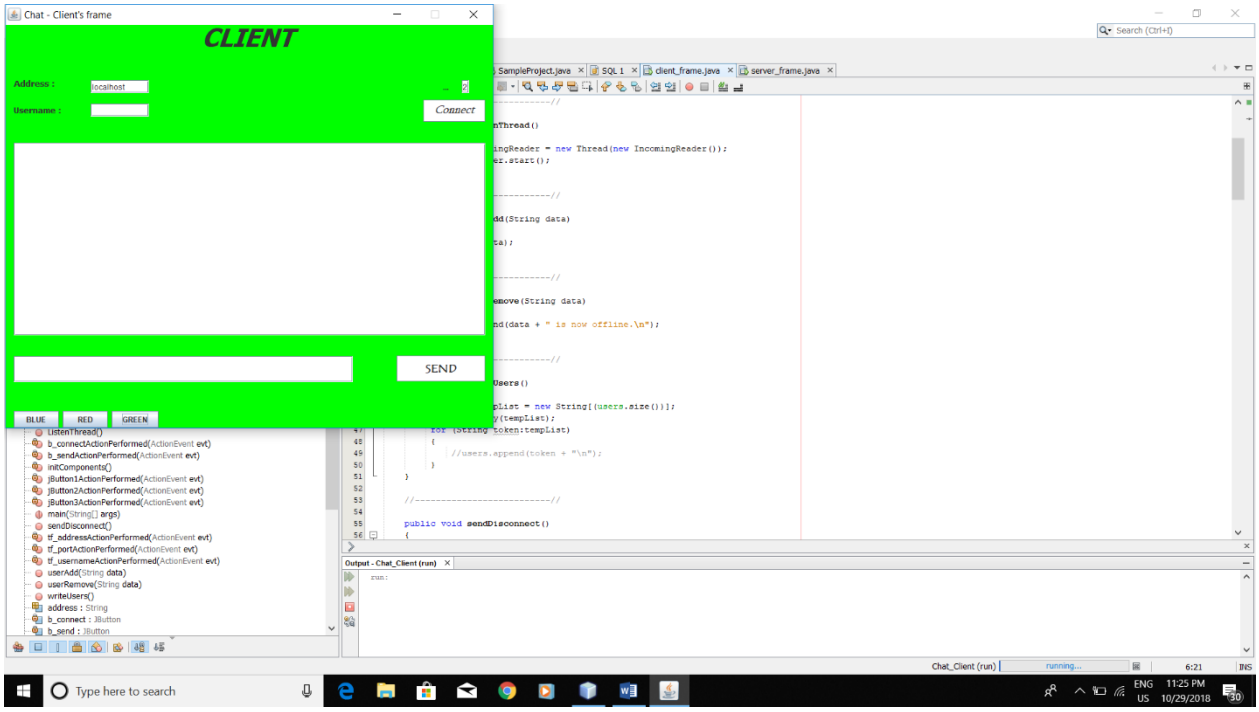
```
            b_startActionPerformed(evt);
        }
    });

    b_end.setText("END");
    b_end.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            b_endActionPerformed(evt);
        }
    });

    b_users.setText("Online Users");
    b_users.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            b_usersActionPerformed(evt);
        }
    });

    b_clear.setText("Clear");
    b_clear.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            b_clearActionPerformed(evt);
        }
    });

    lb_name.setText("Kaustubh's Chatroom");
    lb_name.setBorder(javax.swing.BorderFactory.createLineBorder(null));

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jScrollPane1)
                .addGroup(layout.createSequentialGroup()
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                        .addComponent(b_end, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(b_start, javax.swing.GroupLayout.DEFAULT_SIZE, 75, Short.MAX_VALUE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 281, Short.MAX_VALUE)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                        .addComponent(b_clear, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(b_users, javax.swing.GroupLayout.DEFAULT_SIZE, 103, Short.MAX_VALUE))))
            .addContainerGap())
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(lb_name)
            .addGap(209, 209, 209))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 340, Short.MAX_VALUE)
```

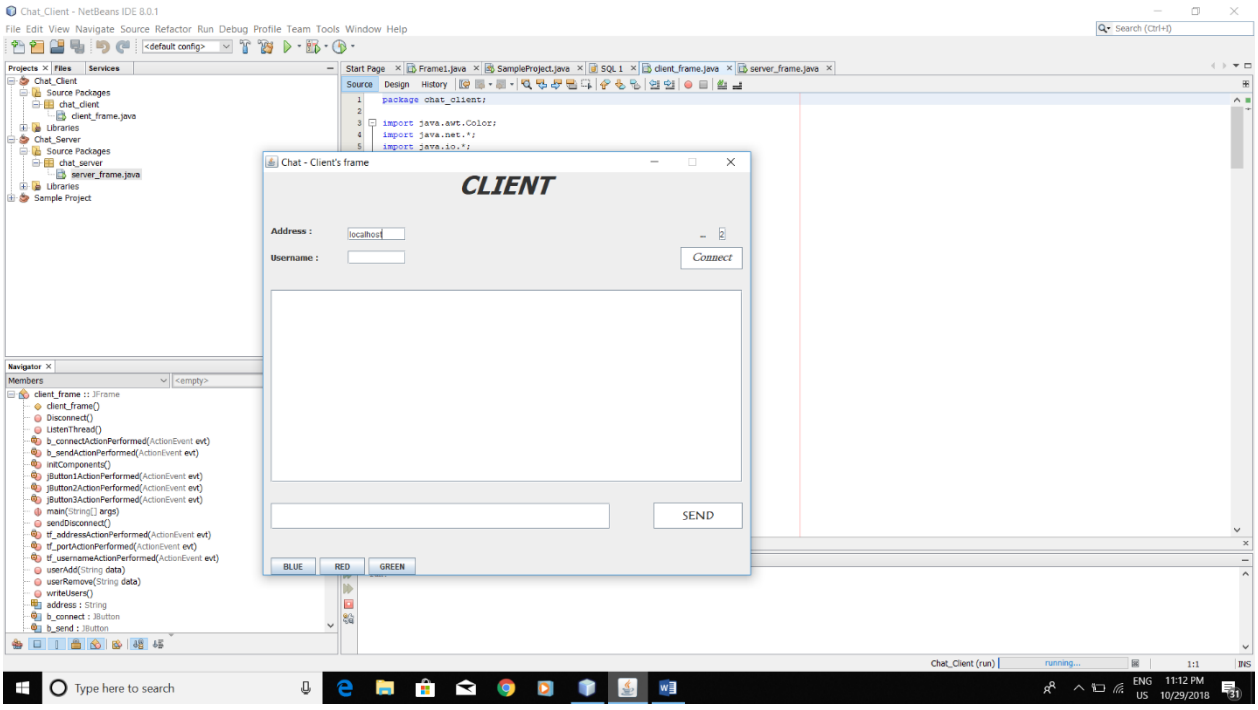
```
                .addGap(18, 18, 18)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(b_start)
                    .addComponent(b_users))
                .addGap(18, 18, 18)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(b_clear)
                    .addComponent(b_end))
                .addGap(4, 4, 4)
                .addComponent(lb_name))
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void b_endActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_b_endActionPerformed
        try
        {
            Thread.sleep(5000);             //5000 milliseconds is five second.
        }
        catch(InterruptedException ex) {Thread.currentThread().interrupt();}

        tellEveryone("Server:is stopping and all users will be disconnected.\n:Chat");
        ta_chat.append("Server stopping... \n");

        ta_chat.setText("");
    }//GEN-LAST:event_b_endActionPerformed

    private void b_startActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_b_startActionPerformed
        Thread starter = new Thread(new ServerStart());
        starter.start();

        ta_chat.append("Server started...\n");
    }//GEN-LAST:event_b_startActionPerformed

    private void b_usersActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_b_usersActionPerformed
        ta_chat.append("\n Online users : \n");
        for (String current_user : users)
        {
            ta_chat.append(current_user);
            ta_chat.append("\n");
        }

    }//GEN-LAST:event_b_usersActionPerformed

    private void b_clearActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_b_clearActionPerformed
        ta_chat.setText("");
    }//GEN-LAST:event_b_clearActionPerformed

    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable()
        {
```

```
        @Override
        public void run() {
            new server_frame().setVisible(true);
        }
    });
}

public class ServerStart implements Runnable
{
    @Override
    public void run()
    {
        clientOutputStreams = new ArrayList();
        users = new ArrayList();

        try
        {
            ServerSocket serverSock = new ServerSocket(2222);

            while (true)
            {
                                        Socket clientSock = serverSock.accept();
                                        PrintWriter writer = new PrintWriter(clientSock.getOutputStream());
                                        clientOutputStreams.add(writer);

                                        Thread listener = new Thread(new ClientHandler(clientSock, writer));
                                        listener.start();
                                        ta_chat.append("Got a connection. \n");
            }
        }
        catch (Exception ex)
        {
            ta_chat.append("Error making a connection. \n");
        }
    }
}

public void userAdd (String data)
{
    String message, add = ": :Connect", done = "Server: :Done", name = data;
    ta_chat.append("Before " + name + " added. \n");
    users.add(name);
    ta_chat.append("After " + name + " added. \n");
    String[] tempList = new String[(users.size())];
    users.toArray(tempList);

    for (String token:tempList)
    {
        message = (token + add);
        tellEveryone(message);
    }
    tellEveryone(done);
}

public void userRemove (String data)
{
    String message, add = ": :Connect", done = "Server: :Done", name = data;
```

```
            users.remove(name);
            String[] tempList = new String[(users.size())];
            users.toArray(tempList);

            for (String token:tempList)
            {
               message = (token + add);
               tellEveryone(message);
            }
            tellEveryone(done);
         }

         public void tellEveryone(String message)
         {
                  Iterator it = clientOutputStreams.iterator();

            while (it.hasNext())
            {
               try
               {
                  PrintWriter writer = (PrintWriter) it.next();
                              writer.println(message);
                              ta_chat.append("Sending: " + message + "\n");
                  writer.flush();
                  ta_chat.setCaretPosition(ta_chat.getDocument().getLength());

               }
               catch (Exception ex)
               {
                              ta_chat.append("Error telling everyone. \n");
               }
            }
         }

         // Variables declaration - do not modify//GEN-BEGIN:variables
         private javax.swing.JButton b_clear;
         private javax.swing.JButton b_end;
         private javax.swing.JButton b_start;
         private javax.swing.JButton b_users;
         private javax.swing.JScrollPane jScrollPane1;
         private javax.swing.JLabel lb_name;
         private javax.swing.JTextArea ta_chat;
         // End of variables declaration//GEN-END:variables
      }
```

CLIENT'S FRAME:





8. CONCLUSION:

In this project, we designed and implemented a small-scale client/server database application program.

Whatever changes are made in the client are also implemented in the server as both of them are connected to each other through the use of ports.

LIMITATIONS:

1) Congestion in Network: Too many requests from the clients may lead to congestion, which rarely takes place in P2P network. Overload can lead to breaking-down of servers. In peer-to-peer, the total bandwidth of the network increases as the number of peers increase.
2) Client-Server architecture is not as robust as a P2P and if the server fails, the whole network goes down. Also, if you are downloading a file from server and it gets abandoned due to some error, download stops altogether. However, if there would have been peers, they would have provided the broken parts of file.

3) Cost: It is very expensive to install and manage this type of computing.

4) You need professional IT people to maintain the servers and other technical details of network.

FUTURE SCOPE OF WORK:

The work covered in the thesis tries to solve various issues, which emerged as a result of literature survey. Still there are many unopened questions left and are of interest were identified and are mentioned below:

- **Web Security at Client Side**

In this research work we proposed PECA, but it still has shortcomings. For example, when web document is required to be save in the portable extended memory it reduces the security due to decentralization of data. During updates at client-side malicious codes may transfer to the

client machine. So, a web security framework is required at the client side to make PECA more secure and better performer. So, in near future, PECA and SWAM may be merged.

- **Server-Side Load Balancing**

Load balancing is a concept which is still under research. Everyday new frameworks, algorithms and models are being developed and existing models are updated. There is a vast scope for future enhancement. For example, the users are sending arbitrary data as a query on the web, and hence web-centric queries can be optimized at server level to reduce server load to improve the server performance. Further, implementation of our work is pending and hence an improvement may be recommended in the same.

9. REFERENCES:

1. Novell's Guide to Client-Server Application & Architecture - Jeffrey D. Schqnk, NovellPress.

2. Client Server Computing - Dawna Travis Dewire, McGraw Hill.

3. Developing Client Server Applications -W.H.Inman, BPB.

4.Guide to Client Server Databases - Joe Salemi, BPB