VIETNAM NATIONAL UNIVERSITY - HCM
Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering


## COMPUTER NETWORK (CO3003)

### Assignment 1

# NETWORK APPLICATION



| Name | Student's ID |
|---|---|
| Nguyễn Hữu Trung Nhân | 1752392 |



Ho Chi Minh City, May 2020 - Semester 192

# Contents

# 1 Phase 1

## 1.1 Functions of the application

### 1.1.1 "Login" function

"Login" function allows a user to get access to this chat application. A user just have to enter his/her name to use this chat application. If the name which was entered by the user is already exist on the server side, this "Login" function will display an error "This username is already exist" to inform the user that he/she needs to make a little bit change in his/her username, for instance, if a user's name is Clark, this Clark guy enter his name in the "Login" function to get access to the chat application. However, unfortunately, on the server side, there is a different person named Clark has already entered the chat application before. Therefore, in order to enter this chat application, this user should make a little bit change in his username (for example, using "Clark001", or "Clark99",... instead of just "Clark").

### 1.1.2 "ClientChat" function

"ClientChat" function allows a user to carry out a private chat with another user. As well as this user can have private chats with many different users who are already online. The connection between users will be implemented based on Peer-to-Peer architecture.

### 1.1.3 "File-Transfer" function

"File-Transfer" function allows a user to transfer as well as receive a file from other users using this chat application. The file's format and the file's size which are allowed to be transferred between users is defined before by the developer of this chat application in the source code. The connection between between users in this "File-Transfer" function part will be implemented based on Peer-to-Peer architecture.

## 1.2 Protocols

### 1.2.1 Protocol for "Login" function

When a user want to use this chat application, the very first thing is that he/she has to login. After "Login" function received the username, "Login" function will create a new port on this client side (the number of this port is greater or equal to 10000; This client port will be used to make Peer-to-Peer connection with other users) as well as create a new client socket (this client socket contains IP address of the server as well as the port number of the server). This client socket helps create connection to the server. After the connection with the server has been established, the client side will send to server a XML file which has a content as below:

<SESSION_REQ>
<PEER_NAME>
user-name
</PEER_NAME>
<PORT>
client-port
</PORT>
</SESSION_REQ>

On the server side, after the server has received the XML file, the server will analyses the data to check if the "user-name" is already existed or not and return a message (which is contained in a XML file) back to the client machine. After receiving message in the XML file from the server, "Login" function will close the socket and start to check the message. If the message in the XML file is "<SESSION_DENY />", the "Login" function will display an error on the Login User Interface to notice the user that the user-name is already used and he/she has to try another user-name. If the user has logged in successfully, "Login" function will change to the Main User Interface of this chat application.

### 1.2.2 Protocol for "ClientChat" function

The "ClientChat" function allows two client machines to chat with each other based on Peer-to-Peer architecture. When "client machine A" wants to chat with "client machine B", "client machine A" will try to connect to "client machine B", in this case, "client machine A" plays as a client and "client machine B" plays as a small server. As usual, "client machine A" will create a socket, this socket contains the IP and the port number of "client machine B". This socket helps "client machine A" connect with "client machine B". When the connection has been established, "client machine A" will send to "client machine B" a XML file which has a content that request the permission to connect as below:

<CHAT_REQ>
<PEER_NAME>
name-of-client-machine-A
</PEER_NAME>
</CHAT_REQ>

After the "client machine B" received the request in the XML file, the decision of "client machine B" will be send back to "client machine A" also in a XML file. If that response message in the XML file contains "<CHAT_DENY />, this means that "client machine B" do not want to chat with "client machine A", then the socket which contains the IP and the port number of "client machine B" will be closed in order to terminate the connection between "client machine A" and "client machine B". Otherwise, the connection is kept and chat windows for "client machine A" and "client machine B" will be opened so that to machines can chat with each other.

### 1.2.3 Protocol for "File-Transfer" function

The "File-Transfer" function allows two client machines to send files to each other based on Peer-to-Peer architecture. For instance, there are two client machines A and B. "Client machine A" wants to send a file to "client machine B". In this case, the very first thing to do is "client machine A" will form a connection to "client machine B". So "client machine A" will act as a client and "client machine B" will become a small server. As usual, "client machine A" will create a socket, this socket contains the IP and the port number of "client machine B". This socket helps "client machine A" connect with "client machine B". When the connection has been established, "client machine A" will send to "client machine B" a XML file which has a content that request the permission to connect as below:

<CHAT_REQ>
<PEER_NAME>
name-of-client-machine-A
</PEER_NAME>
</CHAT_REQ>

After the "client machine B" received the request in the XML file, the decision of "client machine B" will be send back to "client machine A" also in a XML file. If that response message in the XML file contains "<CHAT_DENY />, this means that "client machine B" do not want to chat with "client machine A", then the socket which contains the IP and the port number of "client machine B" will be closed in order to terminate the connection between "client machine A" and "client machine B". Otherwise, the connection is kept and chat windows for "client machine A" and "client machine B" will be opened. At this moment, both "client machine A" and "client machine B" can chat with each other as well as send files to each other. Consider an example in which "client machine A" sends a file to "client machine B". Firstly, "client machine A" will send a message to "client machine B" to require a permission to send a file. If "client machine B" do not want to receive that file, "client machine B" will send back the message in a XML file to "client machine A" which has the content "<FILE_REQ_NOACK />". Otherwise, if "client machine B" wants to receive that file, "client machine B" will send back the message in a XML file to "client machine A" which has the requirement to download that file as below:

<FILE_REQ_ACK>
client-port-number
</FILE_REQ_ACK>

When "client machine A" received the above XML file from "client machine B", "client machine A" will start to send that file following this below protocol:

<FILE_DATA_BEGIN />
data-of-the-file
<FILE_DATA_END />

"data-of-the-file" will be in the binary format because in the Internet Protocol Stack, at the Physical Layer, individual bit of data within a frame will be transferred from one machine to another machine. "<FILE_DATA_BEGIN />" will be sent from "client machine A" to "client machine B" to notify "client machine B" that file transferring process has been started and "<FILE_DATA_END />" will also be sent from "client machine A" to "client machine B" to notify "client machine B" that "client machine B" has fully received the file.

# 2 Phase 2

## 2.1 Specific functions of the application

### 2.1.1 Server Package

The Server Package contains two classes: ServerCore class and ServerGui class.

**ServerCore Class:** This class handles the back end of a server. "ServerCore Class" has all basics functions that a normal server built based on Client-Server architecture must have: create a new socket, waiting for a new connection to the server, accept the connection, read request, write reply, and close the connection. Besides those basic functions above, "ServerCore Class" can also manage a list of online client machine that are connecting to the server (Personally, I would like to call a client machine "a peer"). "ServerCore Class" will have a array list of "peer" type. A peer (a client machine) is defined as below:

```
private String namePeer = "";   //Name of a client machine
private String hostPeer = "";   //IP address of a client machine
private int portPeer = 0;       //Port number of a client machine

public void setPeer(String name, String host, int port) {
  namePeer = name;
  hostPeer = host;
  portPeer = port;
}
```

Besides having a array list of "peer" type to manage client machines that are connecting to the server, in "ServerCore Class", I also implemented a "waiting for a new connection" function named "WaitForConnect()" and this "WaitForConnect()" function is multi-threaded so that this server can handle the requests of many client machines at a same time. Below is how I implemented "WaitForConnect()" function:

```
public class WaitForConnect extends Thread {
  @Override
  public void run() {
    super.run();
    try {
      while (!isStop) {
        if (waitForConnection()) {
          if (isExit) {
            isExit = false;
          } else {
            obOutputClient = new ObjectOutputStream(connection.getOutputStream()
);
            obOutputClient.writeObject(sendSessionAccept());
            obOutputClient.flush();
            obOutputClient.close();
          }
        } else {
          obOutputClient = new ObjectOutputStream(connection.getOutputStream());
          obOutputClient.writeObject(Tags.SESSION_DENY_TAG);
          obOutputClient.flush();
          obOutputClient.close();
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

And the below function shows how "ServerCore Class" can create a new server socket and waiting for connections:

```
public ServerCore(int port) throws Exception {
  ServerSocket server = new ServerSocket(port);
  ArrayList<Peer> dataPeer = new ArrayList<Peer>(); //This array will manage the
   list of online clients
  (new WaitForConnect()).start();
}
```

To stop the server, in "ServerCore Class" there is "stopserver()" function as below:

```
public void stopserver() throws Exception {
  boolean isStop = true;
  ServerSocket server.close();
  Socket connection.close();
}
```

**ServerGui Class:** This class is the User Interface of the server. "ServerGui Class" provides an user interface that contains one button to turn on the server and one button to turn off the server. Also this user interface can show the number of online users. In order to create the user interface for the server, I used a java library called **javax.swing.***. If the server is off and button "Start" is pressed, "ServerGui Class" will create an object of "ServerCore Class" to form a new server socket:

```
    JButton btnStart = new JButton("START");
    btnStart.setBackground(UIManager.getColor("RadioButtonMenuItem.
    selectionBackground"));
    btnStart.setFont(new Font("Segoe UI", Font.PLAIN, 13));

    btnStart.setBounds(360, 36, 143, 43);
    frmServerMangement.getContentPane().add(btnStart);

    btnStart.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
        try {
          ServerCore server = new ServerCore(9999);
          JLabel lblStatus.setText("<html><font color='green'>RUNNING...</font></
    html>");
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    });
```

If the server is running and button "Stop is pressed, "ServerGui Class" will call "stopserver()" function in "ServerCore Class" in order to stop the server:

```
    JButton btnStop = new JButton("STOP");
    btnStop.setFont(new Font("Segoe UI", Font.PLAIN, 13));

    btnStop.setBounds(360, 108, 143, 43);
    frmServerMangement.getContentPane().add(btnStop);

    btnStop.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent arg0) {
        JLabel lblUserOnline.setText("0");
        try {
          ServerCore server.stopserver();
```

```
            JLabel lblStatus.setText("<html><font color='red'>OFF</font></html>");
        } catch (Exception e) {
            e.printStackTrace();
            lblStatus.setText("<html><font color='red'>OFF</font></html>");
        }
      }
    });
```

### 2.1.2 Client Package

The Client Package contains four classes: Client class, ClientServer class, MainGui class, ChatGui class.

**Client Class:** "Client Class" has "Client()" function to manage the information (IP address, client port number, name and the client message). In order to to update the list of online friends frequently, "Client()" function creates a thread to send request to the server. "Client()" function can also handle requests to connect from many other different users through "Request()" function. Below is the structure of "Client()" function:

```
public Client(String arg, int arg1, String name, String dataUser) throws
    Exception {
    IPserver = InetAddress.getByName(arg);
    nameUser = name;
    portClient = arg1;

    clientarray = Decode.getAllUser(dataUser);
    new Thread(new Runnable(){
      @Override
      public void run() {
        updateFriend();
      }
    }).start();

    server = new ClientServer(nameUser);
    (new Request()).start();
}
```

In order to have the ability to handle requests from multiple different users, "Request()" function has to be multi-threaded. Below is the basic idea how I implemented this function:

```
public void request() throws Exception {
    socketClient = new Socket();
    SocketAddress addressServer = new InetSocketAddress(IPserver, portServer);
    socketClient.connect(addressServer);
    String msg = Encode.sendRequest(nameUser);
    serverOutputStream = new ObjectOutputStream(socketClient.getOutputStream());
    serverOutputStream.writeObject(msg);
    serverOutputStream.flush();
    serverInputStream = new ObjectInputStream(socketClient.getInputStream());
    msg = (String) serverInputStream.readObject();
    serverInputStream.close();

    System.out.println("NguyenHuuTrungNhan" + msg);//To see msg Server returns,
    for debug
    clientarray = Decode.getAllUser(msg);
    new Thread(new Runnable() {
      @Override
      public void run() {
        updateFriend();
      }
```

```
    }).start();
  }

  public class Request extends Thread {
    @Override
    public void run() {
      super.run();
      while (!isStop) {
        try {
          Thread.sleep(timeOut);
          request();
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    }
  }
```

To connect with other online users, in "Client Class", I implemented function "intialNewChat()". This function creates a new socket to form the connection between the client and his/her friend. Then, a message requires the permission to establish the connection will be sent from the client to his/her friend's machine. If friend's machine accept, a new chat window will open for chatting. Otherwise, if friend's machine denies to connect, the socket will close the connection. The structure of function "intialNewChat" is presented as follow:

```
  public void intialNewChat(String IP, int host, String guest) throws Exception {
    final Socket connclient = new Socket(InetAddress.getByName(IP), host);
    ObjectOutputStream sendrequestChat = new ObjectOutputStream(connclient.
    getOutputStream());
    sendrequestChat.writeObject(Encode.sendRequestChat(nameUser));
    sendrequestChat.flush();
    ObjectInputStream receivedChat = new ObjectInputStream(connclient.
    getInputStream());
    String msg = (String) receivedChat.readObject();
    if (msg.equals(Tags.CHAT_DENY_TAG)) {
      MainGui.request("Your friend denied connect with you!", false);
      connclient.close();
      return;
    }
    new ChatGui(nameUser, guest, connclient, portClient);
  }
```

**ClientServer Class:** Because of the requirement that chatting feature between two clients has to be implemented based on Peer-to-Peer architecture, the idea is that, within chatting perspective, each client will plays two role: role client and role server. Therefore, this "ClientServer Class" was implemented in order to help the client machine performs its "server role". "ClientServer Class" creates as well as manages the ServerSocket of each user. Moreover, "ClientServer Class" will wait for any requests for permission to connect from other online friends using this chat application. Here is the basic structure for "ClientServer":

```
  public ClientServer(String name) throws Exception {
    username = name;
    port = Client.getPort();
    serverPeer = new ServerSocket(port);
    (new WaitPeerConnect()).start();
  }
```

"WaitPeerConnect()" can manage many requests to connect from many other clients. Therefore this "WaitPeerConnect()" is multi-threaded:

```
class WaitPeerConnect extends Thread {

  Socket connection;
  ObjectInputStream getRequest;

  @Override
  public void run() {
    super.run();
    while (!isStop) {
      try {
        connection = serverPeer.accept();
        getRequest = new ObjectInputStream(connection.getInputStream());
        String msg = (String) getRequest.readObject();
        String name = Decode.getNameRequestChat(msg);
        int res = MainGui.request("Account: " + name + " want to connect with
  you !", true);
        ObjectOutputStream send = new ObjectOutputStream(connection.
  getOutputStream());
        if (res == 1) {
          send.writeObject(Tags.CHAT_DENY_TAG);

        } else if (res == 0) {
          send.writeObject(Tags.CHAT_ACCEPT_TAG);
          new ChatGui(username, name, connection, port);
        }
        send.flush();
      } catch (Exception e) {
        break;
      }
    }
    try {
      serverPeer.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

**MainGui Class:** "MainGui Class" is the main user interface, this class displays a list of online friends. There will be two buttons displayed on the user interface. One button is the "Chat" button. After choosing a friend to chat, user pressed this button to chat will that friend in a new chatting window, if that friend refuse to chat with user, a new window will pop up to inform the user that his/her friend do not want to chat. The other button is the "Exit", user press this button to close the chat application. The graphic user interface in this class was implemented using a java library called **javax.swing.***. Below is the basic structure contained in the "MainGui Class":

```
private void initialize() {
  frameMainGui = new JFrame();
  frameMainGui.setTitle("Menu Chat");
  frameMainGui.setResizable(false);
  frameMainGui.setBounds(100, 100, 510, 522);
  frameMainGui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frameMainGui.getContentPane().setLayout(null);

  JLabel lblHello = new JLabel("Welcome");
  lblHello.setFont(new Font("Segoe UI", Font.PLAIN, 14));
  lblHello.setBounds(12, 27, 70, 16);
  frameMainGui.getContentPane().add(lblHello);

```

```
14
15      JLabel lblFriendsName = new JLabel("Name Friend: ");
16      lblFriendsName.setFont(new Font("Segoe UI", Font.PLAIN, 13));
17      lblFriendsName.setBounds(12, 378, 110, 16);
18      frameMainGui.getContentPane().add(lblFriendsName);
19
20      txtNameFriend = new JTextField("");
21      txtNameFriend.setFont(new Font("Segoe UI", Font.PLAIN, 13));
22      txtNameFriend.setColumns(10);
23      txtNameFriend.setBounds(100, 372, 384, 28);
24      frameMainGui.getContentPane().add(txtNameFriend);
25
26      btnChat = new JButton("Chat");
27      btnChat.setFont(new Font("Segoe UI", Font.PLAIN, 13));
28
29      btnChat.addActionListener(new ActionListener() {
30
31        public void actionPerformed(ActionEvent arg0) {
32          String name = txtNameFriend.getText();
33          if (name.equals("") || Client.clientarray == null) {
34            Tags.show(frameMainGui, "Invaild username", false);
35            return;
36          }
37          if (name.equals(nameUser)) {
38            Tags.show(frameMainGui, "This software doesn't support chat yourself
    function", false);
39            return;
40          }
41          int size = Client.clientarray.size();
42          for (int i = 0; i < size; i++) {
43            if (name.equals(Client.clientarray.get(i).getName())) {
44              try {
45                clientNode.intialNewChat(Client.clientarray.get(i).getHost(),Client.
    clientarray.get(i).getPort(), name);
46                return;
47              } catch (Exception e) {
48                e.printStackTrace();
49              }
50            }
51          }
52          Tags.show(frameMainGui, "Friend is not found. Please wait to update your
    list friend", false);
53        }
54      });
55      btnChat.setBounds(20, 426, 117, 36);
56      frameMainGui.getContentPane().add(btnChat);
57      btnExit = new JButton("Exit");
58      btnExit.setFont(new Font("Segoe UI", Font.PLAIN, 13));
59      btnExit.addActionListener(new ActionListener() {
60        public void actionPerformed(ActionEvent arg0) {
61          int result = Tags.show(frameMainGui, "Are you sure ?", true);
62          if (result == 0) {
63            try {
64              clientNode.exit();
65              frameMainGui.dispose();
66            } catch (Exception e) {
67              frameMainGui.dispose();
68            }
69          }
70        }
71      });
72      btnExit.setBounds(353, 426, 117, 36);
```

```
    frameMainGui.getContentPane().add(btnExit);

    lblActiveNow = new JLabel("List of Active Users");
    lblActiveNow.setForeground(new Color(100, 149, 237));
    lblActiveNow.setFont(new Font("Segoe UI", Font.PLAIN, 13));
    lblActiveNow.setBounds(10, 72, 156, 16);
    frameMainGui.getContentPane().add(lblActiveNow);

    listActive = new JList<>(model);
    listActive.setFont(new Font("Segoe UI", Font.PLAIN, 15));
    listActive.addMouseListener(new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent arg0) {
        String value = (String)listActive.getModel().getElementAt(listActive.
    locationToIndex(arg0.getPoint()));
        txtNameFriend.setText(value);
      }
    });
    listActive.setBounds(10, 99, 472, 251);
    frameMainGui.getContentPane().add(listActive);

    lblUsername = new JLabel(nameUser);
    lblUsername.setForeground(Color.RED);
    lblUsername.setFont(new Font("Segoe UI", Font.PLAIN, 15));
    lblUsername.setBounds(75, 21, 156, 28);
    frameMainGui.getContentPane().add(lblUsername);
  }
```

**ChatGui Class:** "ChatGui Class" is the chat window user interface. "ChatGui Class" helps display chat messages on the user interface. "ChatGui Class" allows user to type and send messages to other online friends who are also using this chat application as well as allows users to send file to each other. The chat window user interface in this "ChatGui Class" was implemented using java libray **jvax.swing.***. The functions used in "ChatGui Class" to create chat window is similar to the "MainGui Class" shown above. Typing messages feature is implemented in the lines of code as follow:

```
    txtMessage = new JTextField("");
    txtMessage.setBounds(10, 21, 479, 62);
    panelMessage.add(txtMessage);
    txtMessage.setColumns(10);

    txtMessage.addKeyListener(new KeyListener() {

      @Override
      public void keyTyped(KeyEvent arg0) {

      }

      @Override
      public void keyReleased(KeyEvent arg0) {

      }

      @Override
      public void keyPressed(KeyEvent arg0) {
        if (arg0.getKeyCode() == KeyEvent.VK_ENTER) {
          String msg = txtMessage.getText();
          if (isStop) {
            updateChat_send(txtMessage.getText().toString());
            txtMessage.setText("");
```

```
            return;
          }
          if (msg.equals("")) {
            txtMessage.setText("");
            txtMessage.setCaretPosition(0);
            return;
          }
          try {
            chat.sendMessage(Encode.sendMessage(msg));
            updateChat_send(msg);
            txtMessage.setText("");
            txtMessage.setCaretPosition(0);
          } catch (Exception e) {
            txtMessage.setText("");
            txtMessage.setCaretPosition(0);
          }
        }
      }
    });
```

### 2.1.3 Login Package

The Login Package contains Login class.

**Login Class:** "Login Class" will allow a new user to get access to the chat application only if the user-name which the new user entered has not been used by any online users who are using the chat application.

### 2.1.4 Data Package

The Data Package contains DataFile class and Peer class.

**DataFile Class:** "DataFile Class" is responsible for create a File object. The size of this file object is less than or equal to 1024000 Bytes (I intend to set the maximum set to around 1 MB only because if the file size is too large, it is going to take a long time to transfer that file).

**Peer Class:** "Peer Class" create a peer object. A peer object has three attributes, they are IP address, name and port number. This class is implemented as follow:

```
public class Peer {

  private String namePeer = "";
  private String hostPeer = "";
  private int portPeer = 0;
  public void setPeer(String name, String host, int port) {
    namePeer = name;
    hostPeer = host;
    portPeer = port;
  }

  public void setName(String name) {
    namePeer = name;
  }

  public void setHost(String host) {
    hostPeer = host;
  }

  public void setPort(int port) {
```

```
    portPeer = port;
  }

  public String getName() {
    return namePeer;
  }

  public String getHost() {
    return hostPeer;
  }

  public int getPort() {
    return portPeer;
  }
}
```

### 2.1.5 Tags Package

The Data Package contains Tags class, Encode class and Decode class.

**Tags Class:** "Tags Class" defines the protocol using in this chat application based on Extensible Markup Language format. Below are some protocols used by this chat application in "Tags Class":

```
  public static String SESSION_OPEN_TAG = "<SESSION_REQ>";
  public static String SESSION_CLOSE_TAG = "</SESSION_REQ>";
  public static String PEER_NAME_OPEN_TAG = "<PEER_NAME>";
  public static String PEER_NAME_CLOSE_TAG = "</PEER_NAME>";
  public static String PORT_OPEN_TAG = "<PORT>";
  public static String PORT_CLOSE_TAG = "</PORT>";
  public static String SESSION_KEEP_ALIVE_OPEN_TAG = "<SESSION_KEEP_ALIVE>";
  public static String SESSION_KEEP_ALIVE_CLOSE_TAG = "</SESSION_KEEP_ALIVE>";
  public static String STATUS_OPEN_TAG = "<STATUS>";
  public static String STATUS_CLOSE_TAG = "</STATUS>"
  public static String SESSION_DENY_TAG = "<SESSION_DENY />";
  public static String SESSION_ACCEPT_OPEN_TAG = "<SESSION_ACCEPT>";
  public static String SESSION_ACCEPT_CLOSE_TAG = "</SESSION_ACCEPT>";
  public static String CHAT_REQ_OPEN_TAG = "<CHAT_REQ>";
  public static String CHAT_REQ_CLOSE_TAG = "</CHAT_REQ>";
  public static String IP_OPEN_TAG = "<IP>";
  public static String IP_CLOSE_TAG = "</IP>";
  public static String CHAT_DENY_TAG = "<CHAT_DENY />";
  public static String CHAT_ACCEPT_TAG = "<CHAT_ACCEPT />";
  public static String CHAT_MSG_OPEN_TAG = "<CHAT_MSG>";
  public static String CHAT_MSG_CLOSE_TAG = "</CHAT_MSG>";
  public static String PEER_OPEN_TAG = "<PEER>";
  public static String PEER_CLOSE_TAG = "</PEER>";
  public static String FILE_REQ_OPEN_TAG = "<FILE_REQ>"
  public static String FILE_REQ_CLOSE_TAG = "</FILE_REQ>";
  public static String FILE_REQ_NOACK_TAG = "<FILE_REQ_NOACK />";
  public static String FILE_REQ_ACK_OPEN_TAG = "<FILE_REQ_ACK>";
  public static String FILE_REQ_ACK_CLOSE_TAG = "</FILE_REQ_ACK>";
  public static String FILE_DATA_BEGIN_TAG = "<FILE_DATA_BEGIN />";
  public static String FILE_DATA_OPEN_TAG = "<FILE_DATA>";
  public static String FILE_DATA_CLOSE_TAG = "</FILE_DATA>";
  public static String FILE_DATA_END_TAG = "<FILE_DATA_END />";
  public static String CHAT_CLOSE_TAG = "<CHAT_CLOSE />";
```

**Encode Class:** This class converts a normal message into XML format.

**Decode Class:** This class converts a XML format back into a normal message.

## 2.2 Application Designs

### 2.2.1 Class diagram

(Please zoom in to see more clearly)

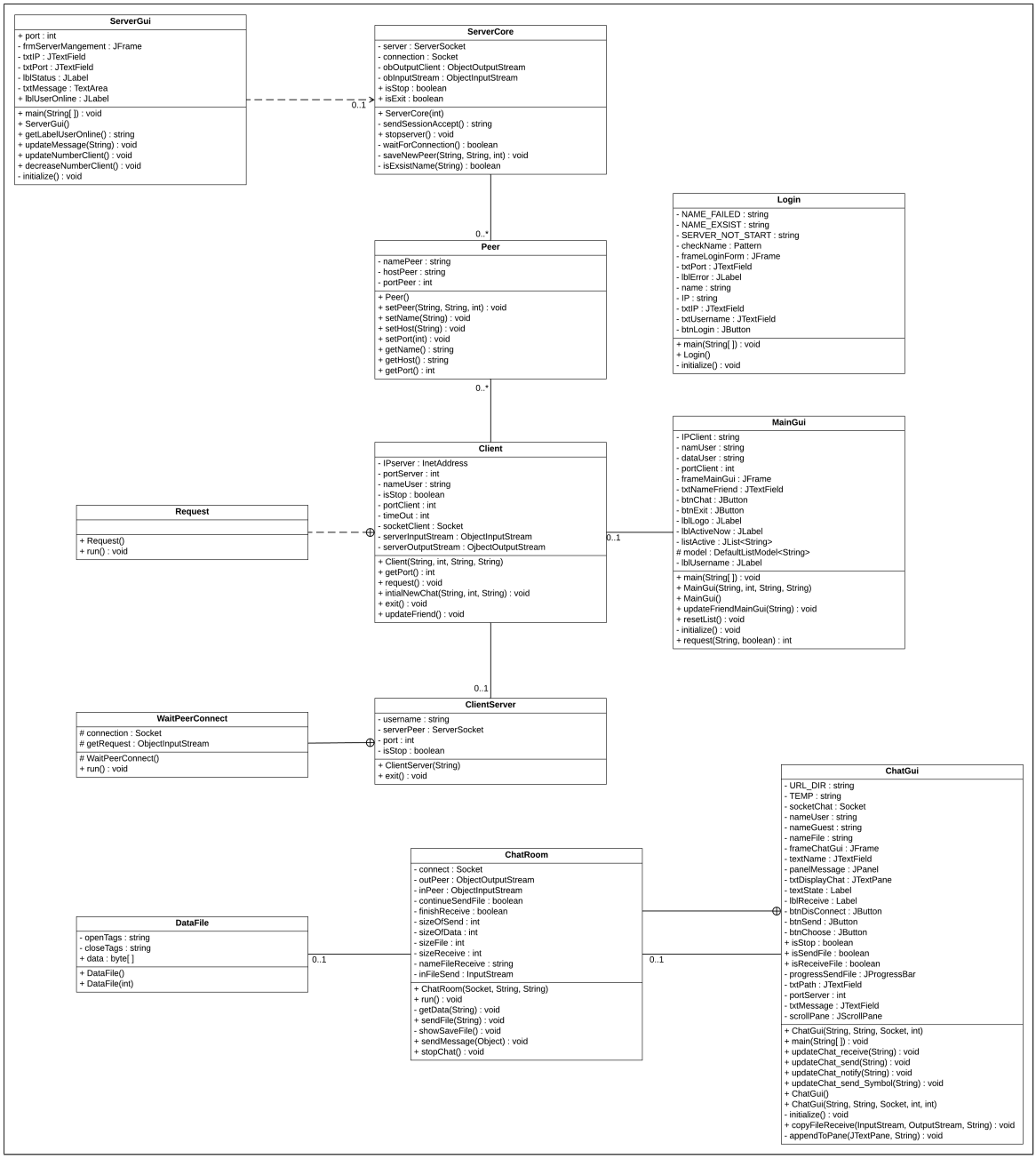


Figure 1: Class diagram for "Chat Application"

### 2.2.2 Activity diagram

(Please zoom in to see more clearly)

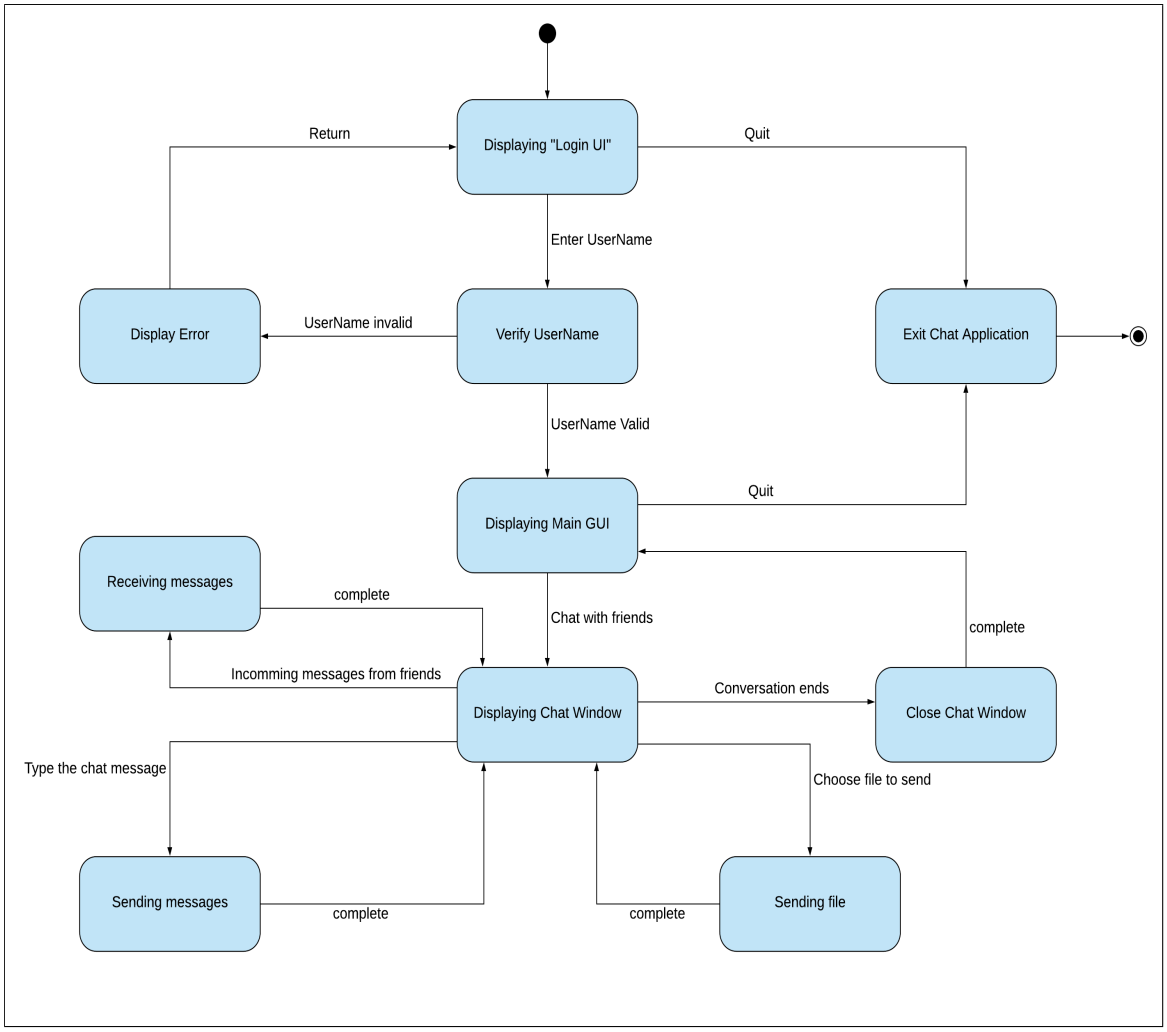


Figure 2: Activity diagram for "Chat Application"

## 2.3 Assessment of actual result

This Chat Application has been successfully implemented and satisfies the requirements. This Chat Application's architecture is the combination of client-server and P2P architectures. Client-server model is used for user management while P2P model is used for chat feature between users. This Chat Application can display the list of online users, allows a user to chat with different people at the same time as will as supports file transferring between different users. Below are some pictures of my result, please zoom in to see more clearly if the picture is not clear:

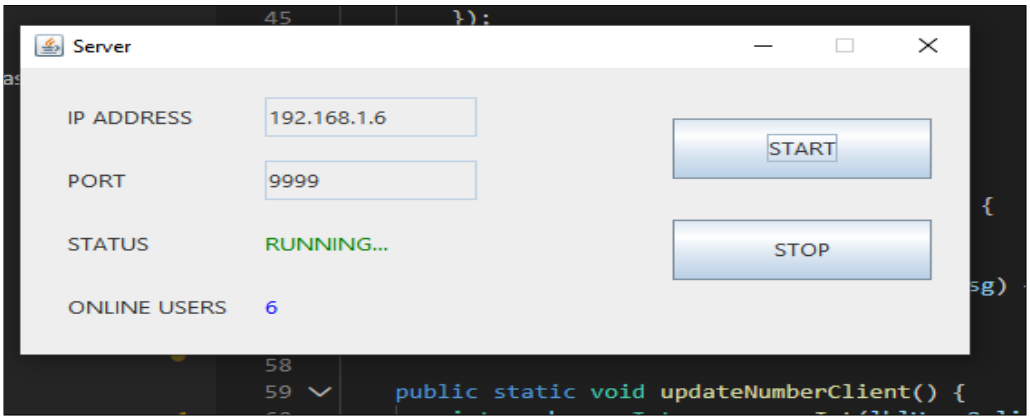


Figure 3

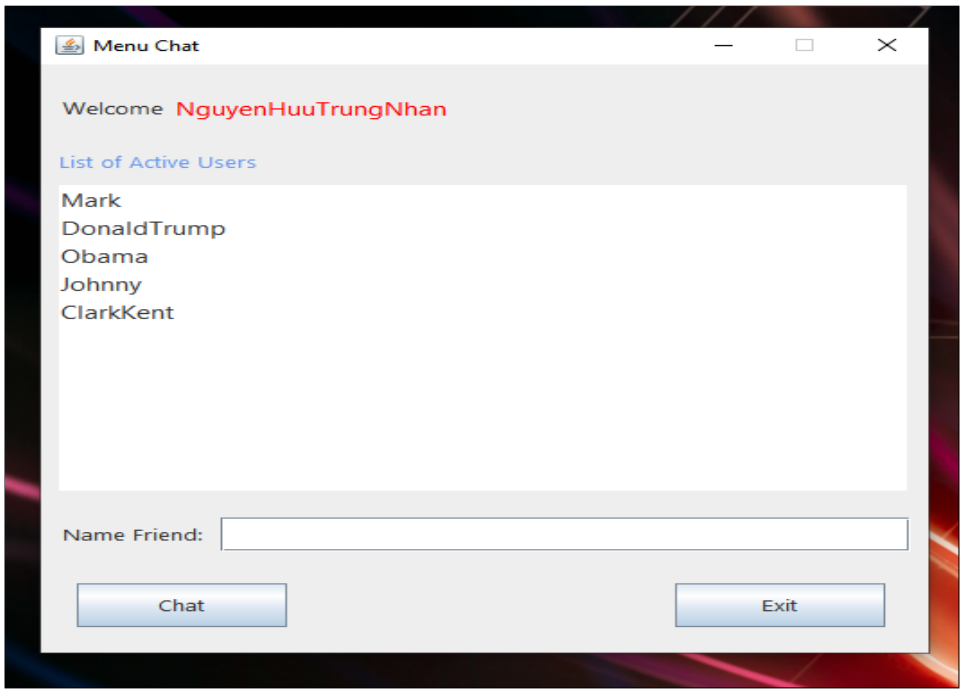


Figure 4

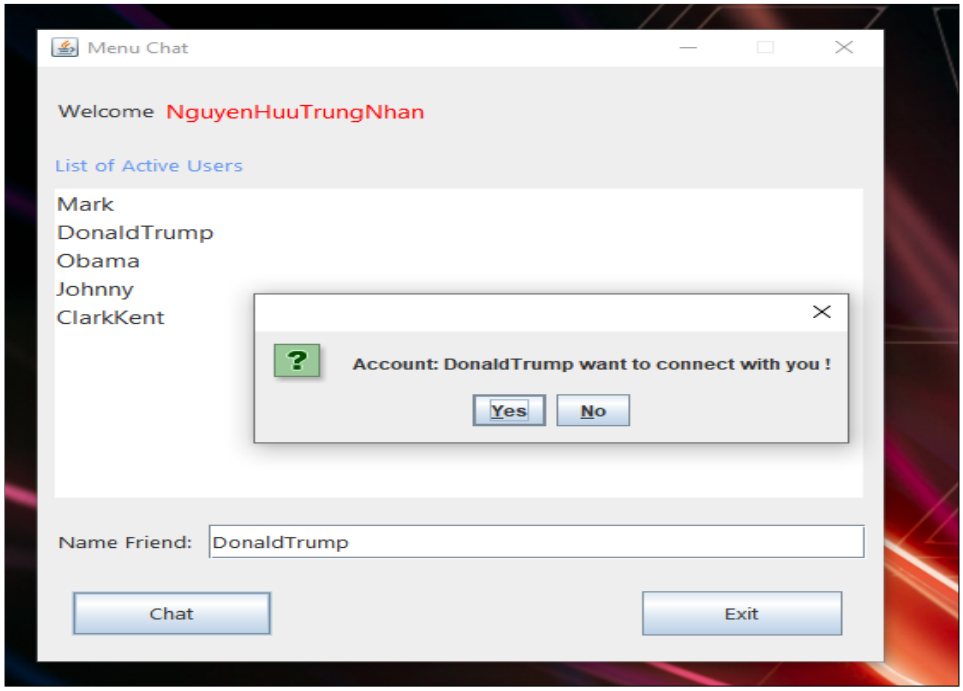


Figure 5

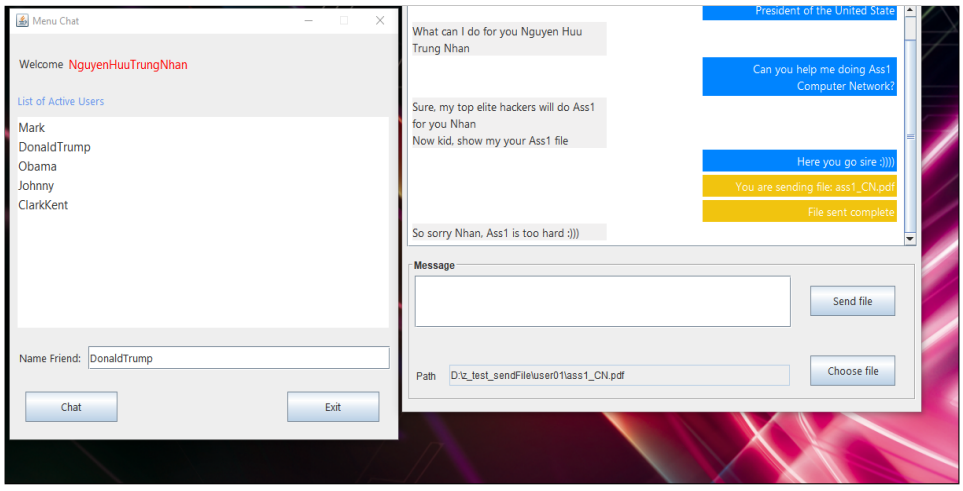


Figure 6

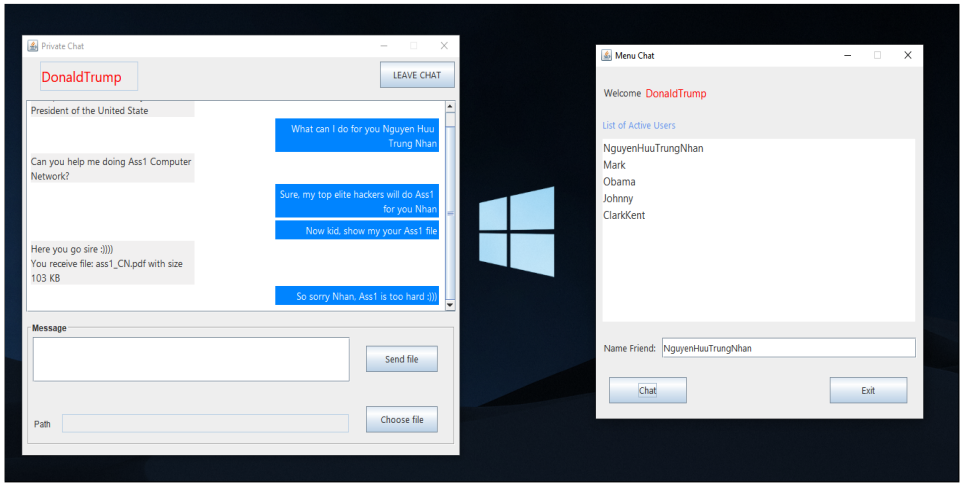


Figure 7

## 2.4 Manual document

Firstly, the server must be running (execute the application file FinalServer.jar)

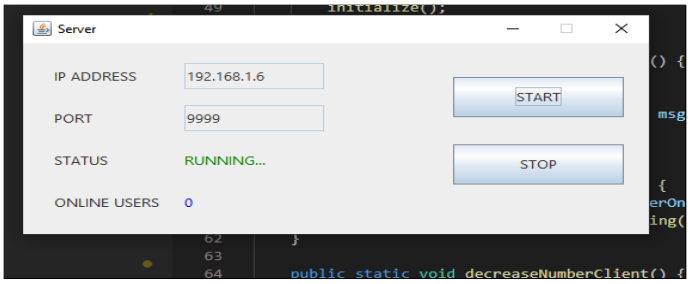


Figure 8

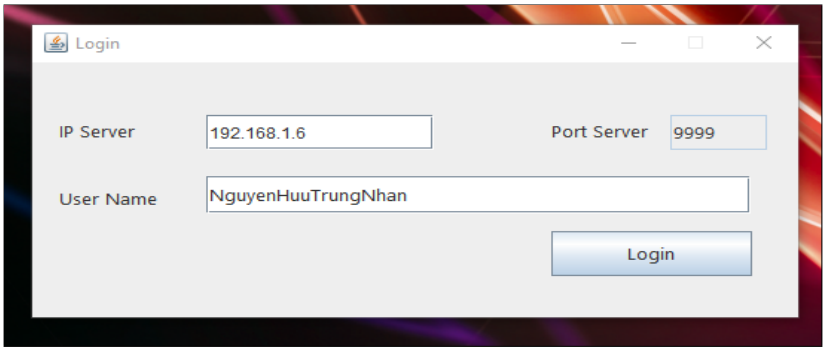


Figure 9

Secondly, (execute the application file FinalClient.jar) enter the username to login the chat

application. Please note that there must be no space in the username (Figure 9). After login to the chat application successfully, the main graphic user interface should look like Figure 10. There is a list of active users in the main graphic user interface. Then, click to choose any active user to start chatting. If the chosen user also agree to chat, there will be a chat room window pop up like the one in Figure 6. To send a file, click at button "Choose file" and select the file that is needed to be sent (Figure 11) and then click button "Send file" to send that file. When the file has been transferred successfully, there will a message "File sent complete" like in Figure 6. To leave the chat, click at button "LEAVE CHAT" in the chat window and to close the application, click button "Exit".

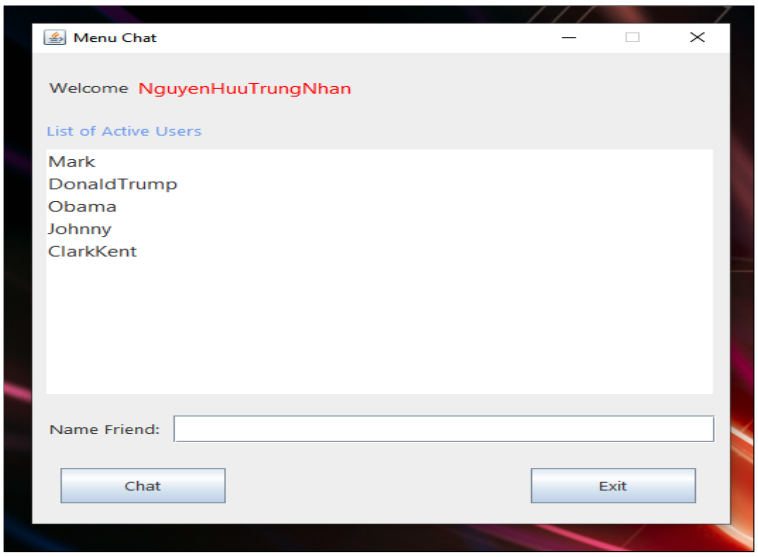


Figure 10

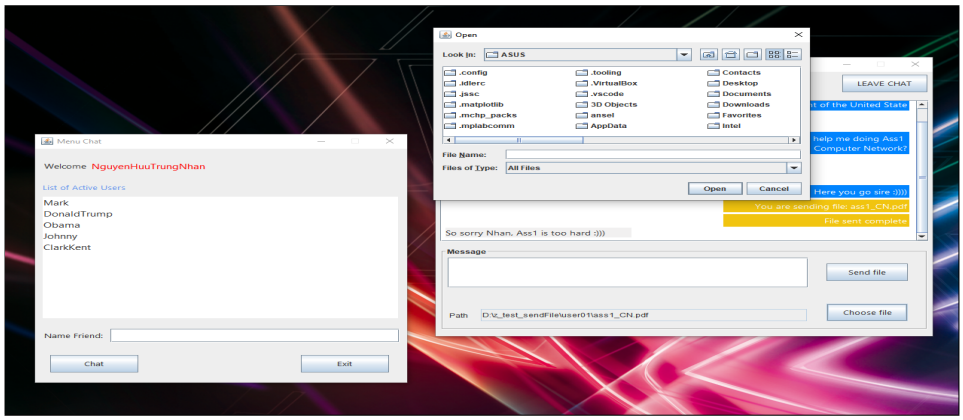


Figure 11

## 2.5 Source code

The source code is located in the following directory:
\ASS1_NguyenHuuTrungNhan_1752392\src

## 2.6 Application file

There are two application files: FinalServer.jar and FinalClient.jar. They are located in the following directory:
\ASS1_NguyenHuuTrungNhan_1752392\src