A Project Report

On

Chat Application

Course: CS 513 – Introduction to Local and Wide Area Networks

Submitted By: Neha Mahajan (627959983)

Submitted On: 2nd July 2015

I certify that this report is all my work, based on my personal study and knowledge. I have acknowledged all material and sources used in preparation of the project.

Neha Mahajan

ABSTRACT

This project report gives the detailed design and implementation of a chat application. The chat application is developed for windows operating system. Using this application, multiple users (clients) can communicate with each other using a single server. Functionalities like “whisper” and “broadcasting” have been implemented based on client server architecture. Attention to detail has been paid while developing the UI, including error handling to make the application robust. The application uses object oriented concepts and is developed in Java using Socket programming. The UI is developed in Java Swing. The report includes test cases used to verify the correctness of the application. The concluding section lists all the classes and the advanced features that can be implemented to enhance the application.

CONTENTS

1. Project Description

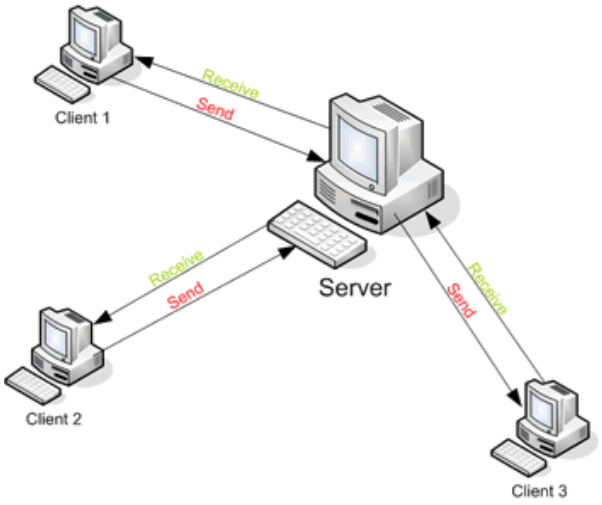
Chatting is a method which allows users to communicate with each other despite of geographical barriers. This chat application project is built on the chat method with multiple clients and single server. Project consists of two separate applications

1. Client:

   In order to communicate with other clients, it must establish a connection with the server. Each client must choose a unique username when connecting to server. Client can either whisper (send a private message to another client) or broadcast (send messages to all clients connected to server).

2. Server:

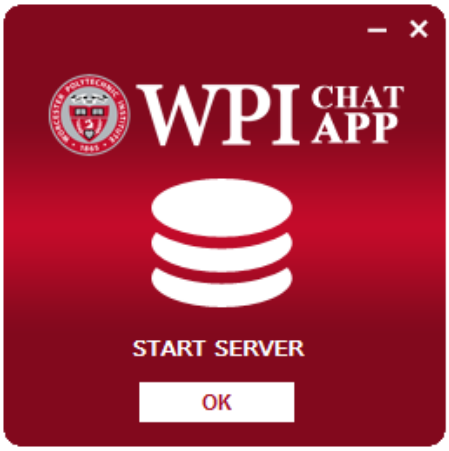
   Basic role of server is to accept connection from multiple clients, maintain list of all online users, update clients when another client connects/disconnects and allow simple text messages to be exchanged between the clients.

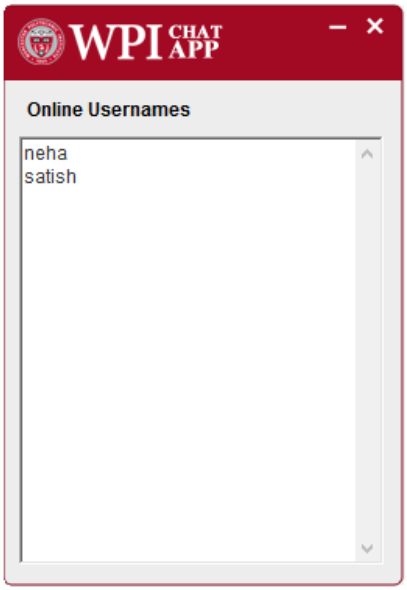


Client – Server Chat Application Model

1.1 Server Application

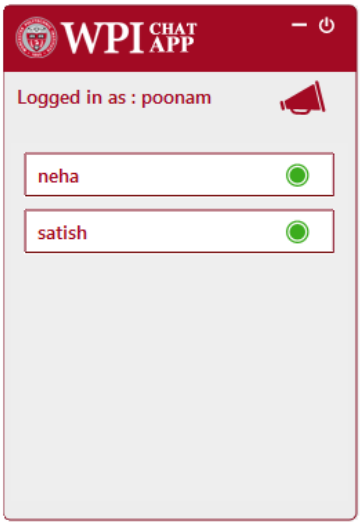
The server program begins by creating a ServerSocket object on a predefined port and waits for client connections. Once a client sends a connection request, server validates the client for a unique username by checking in the list of online users. If successful (username is unique) it sends list of online users to the newly connected client and updates all other clients with the new client username. If unsuccessful it sends "not accepted" message to client. The server handles multiple clients connect/disconnect requests and keep updating the clients. When a client sends a broadcast message to a server, it forwards the messages to all clients currently online. When a client wants to whisper, it sends the message to the server along with the recipient client name. Server filters the list of online clients and sends the message to the appropriate client. When server receives a disconnect request from a client it removes the client from the list of online users and updates other clients.





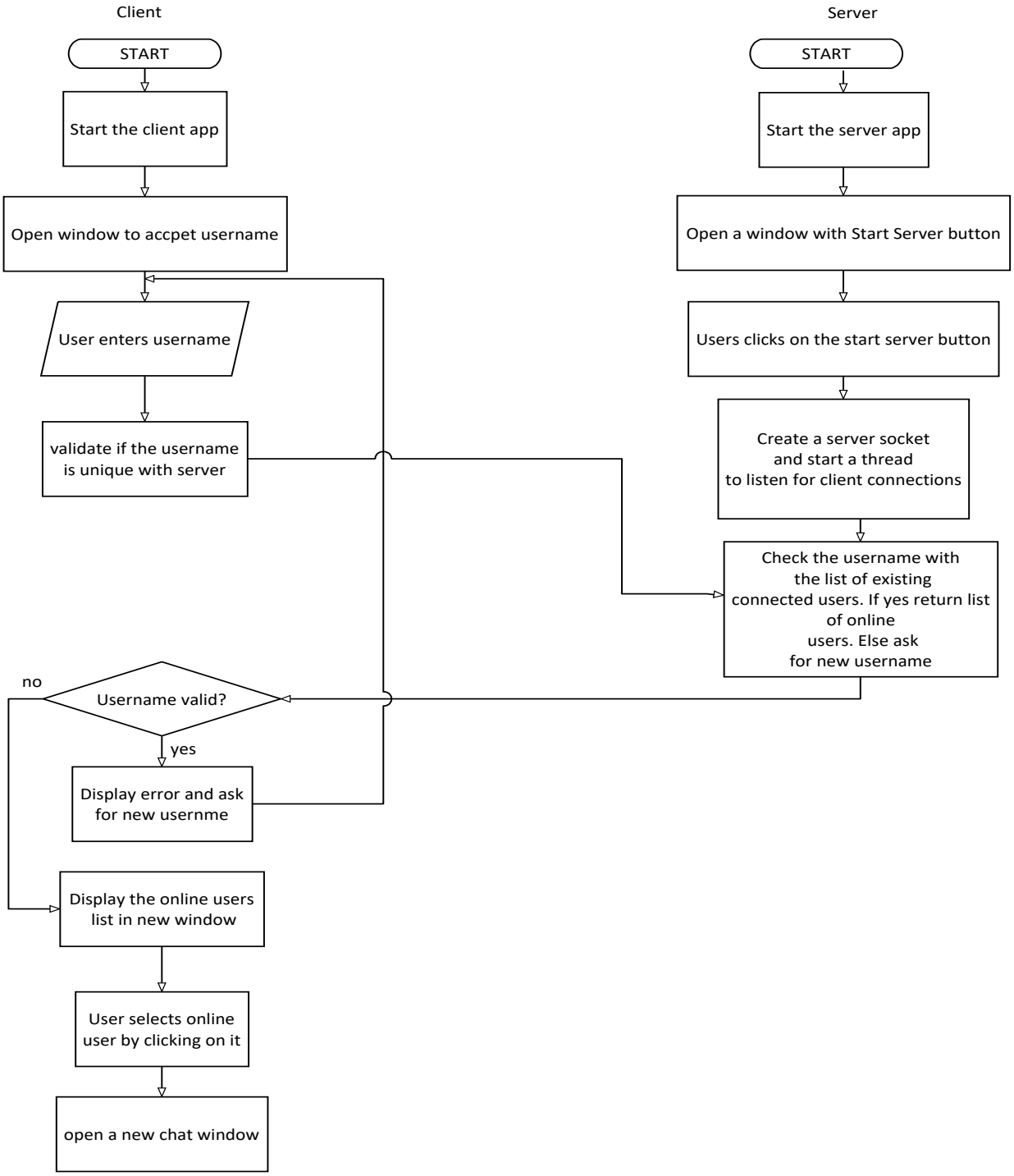
1.2 Client Application

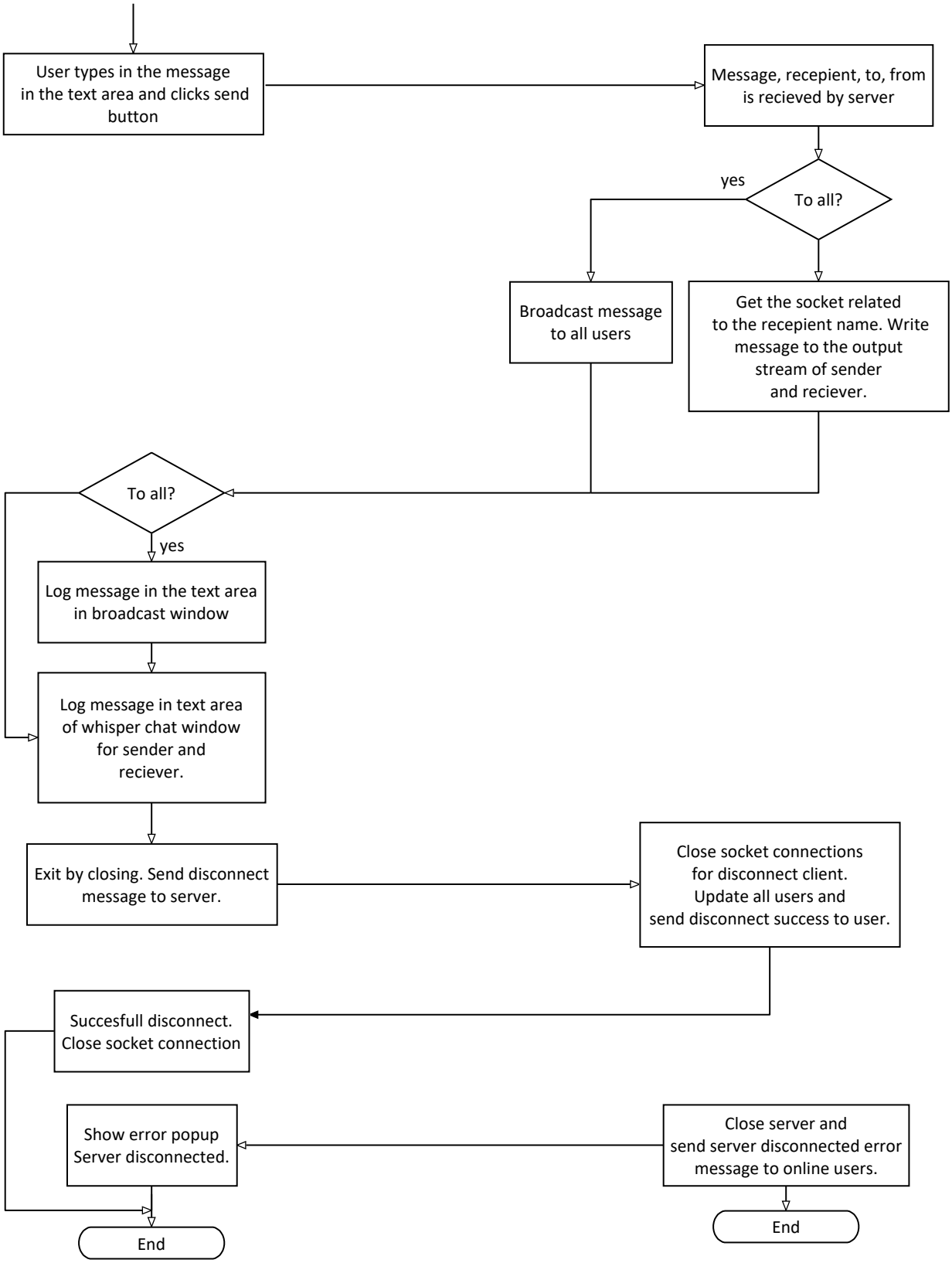
The client application begins by displaying a login page. User enters a username and sends connection request to the server by creating a new socket with hostname of the server (for this project it is localhost) with the specified port number (predefined for both client and server). If successful the main page of application is displayed with the list of online users. Users can either whisper or broadcast a message. When user clicks on a username from the list, a chat window opens where the user can type message and send. All the exchanged messages displayed in the scrollable area. If recipient client disconnects, the sender client cannot send messages anymore. User can chat with multiple other clients in separate unique chat windows. To broadcast a message user clicks on "broadcast icon" which opens a new chat window that can be used to send messages to all online clients. All messages which are broadcasted by any other online client will be displayed in the broadcast chat window.



2. Design and Implementation Details

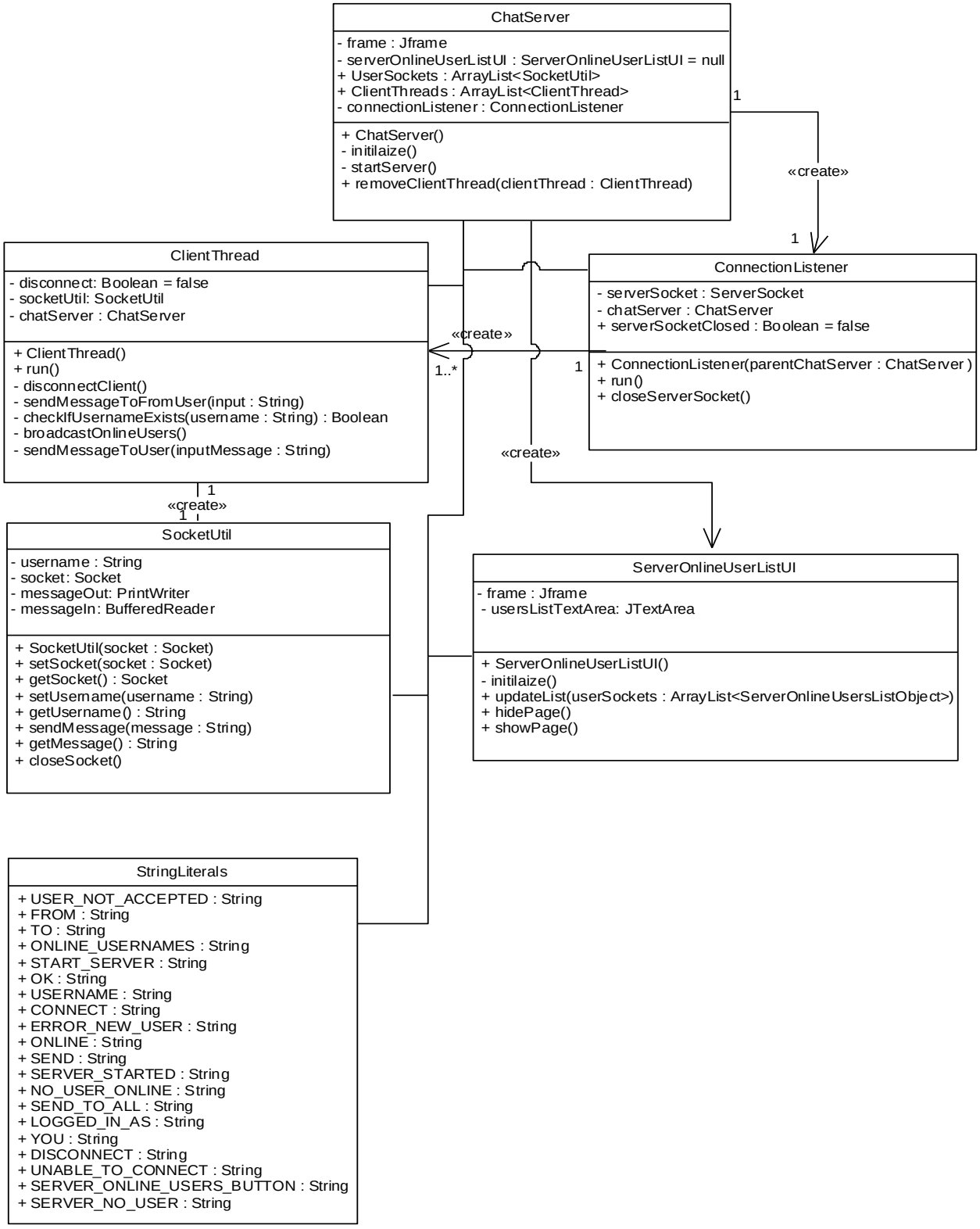
2.1 Flow Chart




User types in the message in the text area and clicks send button
Message, recepient, to, from is recieved by server
To all?
yes
Broadcast message to all users
Get the socket related to the recepient name. Write message to the output stream of sender and reciever.
To all?
yes
Log message in the text area in broadcast window
Log message in text area of whisper chat window for sender and reciever.
Exit by closing. Send disconnect message to server.
Close socket connections for disconnect client. Update all users and send disconnect success to user.
Succesfull disconnect. Close socket connection
Show error popup Server disconnected.
Close server and send server disconnected error message to online users.
End
End

2.2 Server Design and Implementation

2.2.1 Class Diagram



2.2.2 Create a Server Socket to listen for client connections

serverSocket = new ServerSocket(9001);

Java sockets use TCP Protocol for communication. The above line creates a new Server socket on port 9001. Any port number can be used which is not reserved for other services.

2.2.3 Handle client connections

Socket tempSocket = serverSocket.accept();

ClientThread tempClientThread = new ClientThread(tempSocket, this.chatServer);

tempClientThread.start();

Once a socket is created, it waits to accept client connection as shown above. When client creates a socket connection, it is received by the accept() method of Server Socket. For each new client a new socket is created and server allocates a separate thread for each new client connection. Server maintains list of sockets and threads in an ArrayList to send/receive multiple messages simultaneously. If a client provides a username, server looks in the ArrayList for duplicity and accordingly outputs error message or success by sending colon separated list of online users in a string variable. All online users are also updated by the new client username.

2.2.4 Handle Incoming/Outgoing Messages

this.messageOut = new PrintWriter(this.socket.getOutputStream(), true);

this.messageIn = new BufferedReader(new InputStreamReader(this.socket.getInputStream()));

When a client writes a message to the output stream, the server processes the message in the input stream for the client socket. Using the ArrayList server filters the sockets for recipient name and writes the message to the receiver's output stream. Depending on the type of message (whisper or broadcast) server decides to write messages to all output streams (broadcast) or only to receiver's output stream (whisper).

2.2.5 Handle Client Disconnect

When client disconnect/logs off, server receives disconnect message. Server closes the client socket, message input/output streams, removes the socket entry from the ArrayList and updates all other users. Once update is finished, the client thread is destroyed and its entry is removed from ArrayList.

2.2.6 Handle Server Disconnect

If server is closed / stopped, it sends error messages to all users, closes the server socket and the application terminates.

2.2.7 Server GUI Design

Java Swing is used to design the server UI. UI components like JButtons, TextArea, Labels, JFrame etc. are used for designing. Images are used wherever required.

2.2.8 Server Classes

Server application includes 4 classes. They are as follows,

1. ChatServer: Launch point of application and displays start window for server. Maintains ArrayList of client sockets and client threads.

2. ConnectionListener: Creates new ServerSocket and listens for client connection.

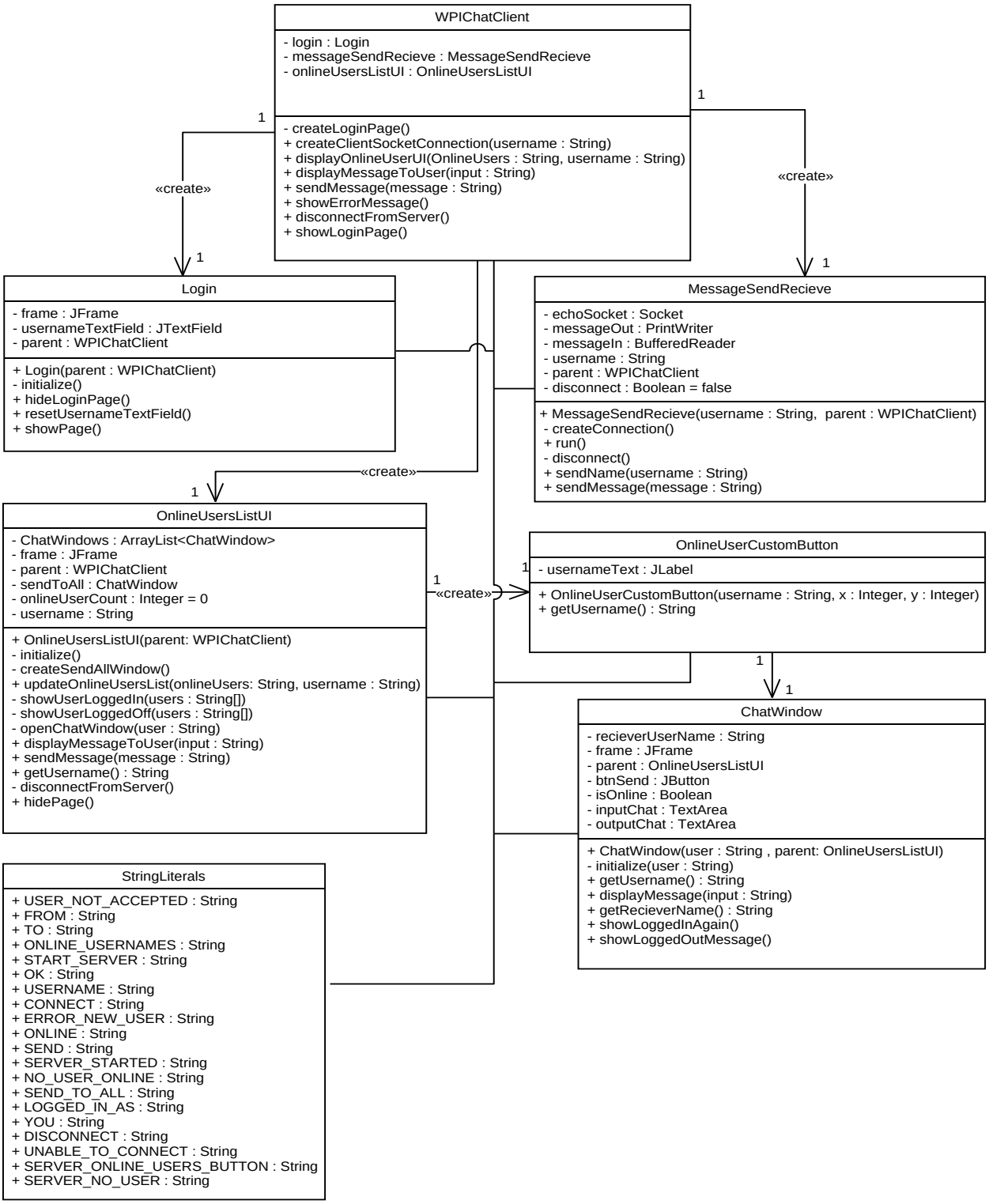
3. ClientThread: Handles the incoming and outgoing messages for single client socket.

4. SocketUtil: Creates input and output streams using client socket. Reads messages from input stream and writes messages to output stream.

5. ServerOnlineUserListUI: Opens new UI window to show list of online users.

2.3 Client Design and Implementation

2.3.1 Class Diagram



2.3.2 Create a Socket to connect to server

```
this.echoSocket = new Socket("localhost", 9001);

this.messageOut = new PrintWriter(this.echoSocket.getOutputStream(),
true);

this.messageIn = new BufferedReader(

        new InputStreamReader(this.echoSocket.getInputStream()));
```

Create a socket with hostname of the server (hostname = localhost) and port 9001. Create input and output streams.

2.3.3 Choose a unique username

Client enters a username and sends it to server for validation. If username is unique, client receives a colon separated list of usernames in a string variable else error message is shown in popup window.

2.3.4 Handle creation of chat windows

On the online user's list UI window, user can select to whisper or broadcast a message. The online users list is maintained in an Array of Strings. For each user a button is created and displayed in scrollable list. To chat with a user, the button with his/her name is clicked. On click of the button, a new chat window is opened. On click of the broadcast message icon, a separate chat window is opened.

2.3.5 Handle Incoming/Outgoing Messages

When user types a message in chat window and clicks send button, message is written to the output stream of socket. Message contains fields as From, To and Message.

For a broadcast message "To" field contains string value as "SendToAll".

E.g.: String message =
"From:XYZ:To:SendToAll:Loremipsumdolorsitamet,consecteturadipiscingelit
"

For a whisper message "To" filed contains string value as "<Recipient Name>"

E.g.: String message =
“From:XYZ:To:ABC:Loremipsumdolorsitamet,consecteturadipiscingelit”

When a message is received in the input stream, it is parsed and according to the value of “From” field it is either displayed in the broadcast chat window or whisper chat window of “From” username.

2.3.6 Handle Client Disconnect

When client clicks on disconnect/logoff button, it sends disconnect message to server. On successful disconnection, client closes the socket, input/output streams and the application terminates.

2.3.7 Handle Server Disconnect

If server is closed / stopped due to some reason, error is received by client and a pop up is shown to the user. On dismiss of the popup client application terminates.

2.3.8 Client GUI Design

Java Swing is used to design the server UI. UI components like JButtons, TextArea, Labels, JFrame etc. are used for designing. Images are used wherever required.

2.3.9 Client Classes

Client application includes 6 classes. They are as follows,

1. WPIChatClient: Launch point of application and displays Login window for client. On successful connection to server, displays the OnlineUsersListUI.

2. Login: Displays input text field to enter username.

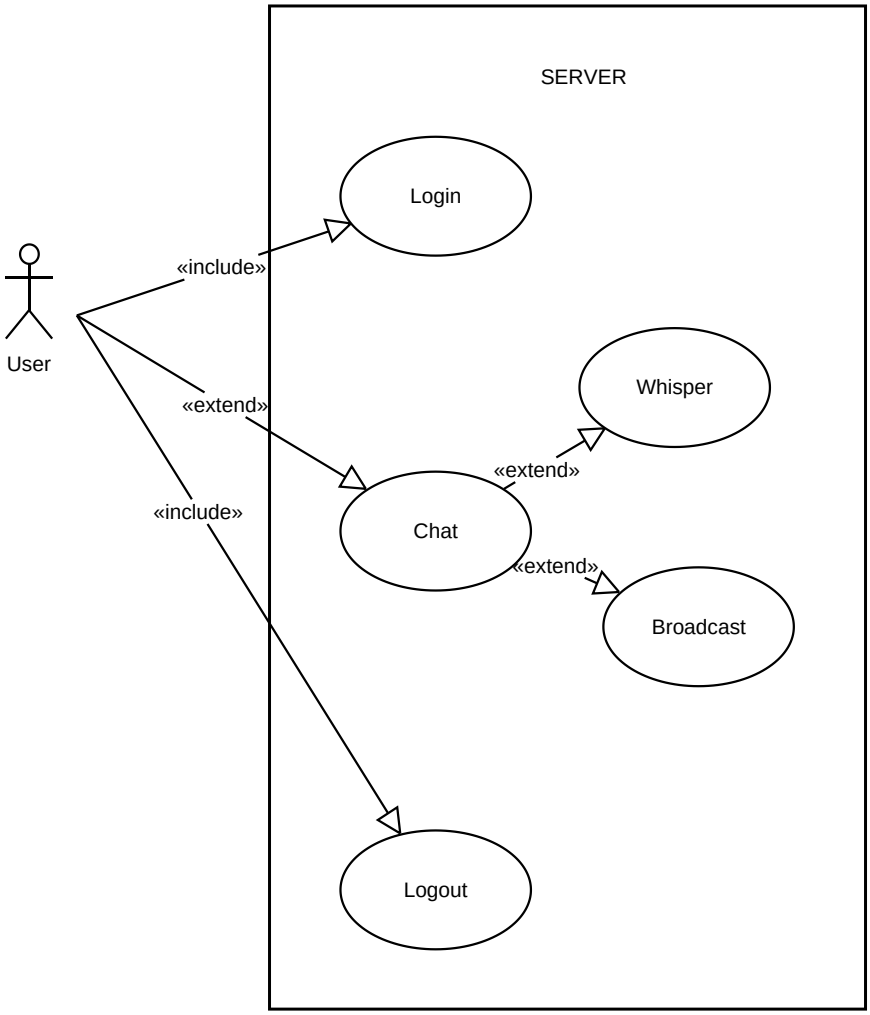
3. OnlineUsersListUI: Displays scrollable area with list of online users, logoff button and broadcast button.

4. OnlineUserCustomButton: Creates a custom UI for each online user to be displayed on the OnlineUsersListUI.

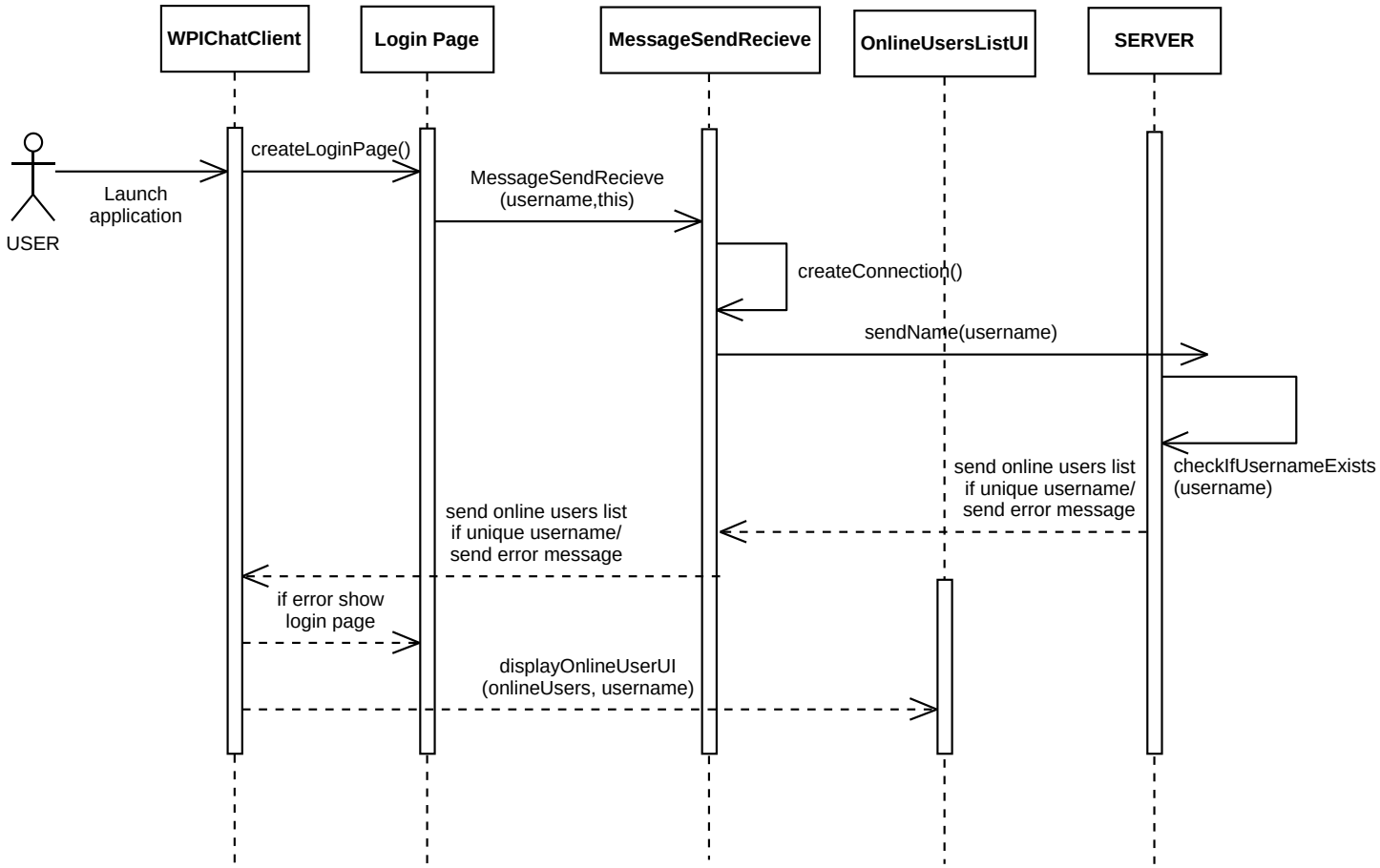
5. ChatWindow: Opens new chat window where user can send and read message logs in scrollable area.

6. MessageSendRecieve: Manages the connection to the server, creating output/input stream and write to output stream/read from input stream.

2.4 Use Case Diagram



2.5 Sequence Diagram



3. Testing and Evaluation

3.1 Unit Testing (Manual Testing)

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. **Unit testing** is often automated but it can also be done manually.

Following table lists down the test cases to check functionality and correctness of each module of client. For test cases 7- 11 the login page was bypassed locally to test functionality for online user's list UI page and chat windows. Hence the test data was hardcoded to achieve required results.

Note: Test data used for testing is dummy data.

Test Case Id	Pre-conditions	Test Description	Test Data	Expected Result
1	Client application must be running.	Validate User Interface for login page.	None	Close, minimize and connect button are visible. Username input text field is visible and editable.
2	Client application must be running.	Functionality of close button for login page.	None	Application should be terminated.
3	Client application must be running.	Functionality of minimize button for login page.	None	Application should be minimized.
4	Client application must be running. Server must be running.	If username length is 8 or less than 8 characters.	"nehamaha"	Username should be accepted to be sent to server.
5	Client application must be running.	If username length greater than 8 characters.	"satishnarkhede"	Display popup for maximum number of characters allowed.
6	Client application must be running.	If username is empty.	""	Display popup for invalid username.
7	Client application must be running. Online user's list UI page must be visible.	Functionality on click of broadcast icon.	onlineUserCount >= 1	Open broadcast chat window.
8	Client application must be running.	If online users are none, check functionality of	onlineUserCount = 0	Display popup "No user is online".

	Online user's list UI page must be visible.	broadcast icon.		
9	Client application must be running. Online user's list UI page must be visible.	UI for no user is online.	onlineUserCount = 0	Display message of no user online in the online user's list UI page.
10	Client application must be running. Online user's list UI page must be visible.	List of online users UI.	onlineUsers = ["ABC", "XYZ", "PQR"]	Custom buttons for each user must be displayed as a list in scrollable box.
11	Client application must be running. Chat window must be visible.	Check if message is visible in scrollable text area.	inputMessages = "Lorem ipsum"	Message must be displayed in the text area.
12	Client application must be running. Chat window must be visible.	Functionality of close button for chat window.	none	Chat window should be closed.
13	Client application must be running. Chat window must be visible.	Functionality of minimize button for chat window.	none	Chat window should be minimized.

Following table lists down the test cases to check functionality and correctness of each module of server.

Test Case Id	Pre-conditions	Test Description	Test Data	Expected Result
1	Server application is running.	Check close button functionality on server page.	none	Application should be terminated with successful closing of server socket.
2	Server application is running.	Checks minimize button functionality on server page.	none	Application should be minimized.
3	Server application is running.	On click of view online user buttons on server page open list view of online clients.	onlineUsers = ["ABC", "XYZ", "PQR"]	List of all online user's should be displayed.

3.2 Integration Testing

Integration testing (sometimes called **integration** and **testing**, abbreviated I&T) is the phase in software **testing** in which individual software modules are combined and tested as a group. It occurs after unit **testing** and before validation **testing.**

Integration testing was done with real client and server application. Few tests cases are shown in the below table. Detailed test cases and results along with screenshots are available in the appendix section.

Test Case Id	Pre-conditions	Test Description	Expected Result
1	Client and server application must be running.	Check if client could establish a connection to server and choose a unique username.	If username is unique server accepts the client.
2	Client and server application must be running.	Check if separate threads are created for each client on server side.	Threads are created and maintained on server side.
3	Client and server application must be running.	Check if message send by client is received by server.	Message is received from client.
4	Client and server application must be running.	Check if message send by server is read by client.	Client receives message from server.
5	Client application and server must be running.	Check if client can disconnect from server.	Client is successfully disconnected.
6	Client application and server must be running.	Check if server shutdown client is updated.	Client receives error message.

4. Limitations and Enhancements

4.1 Limitations

There are few limitations like

1. Online users are maintained for a single server session.

2. Only text communication is possible between users.

3. Data is transferred as a String between client and server.

4.2 Enhancements

There is always a room for improvements in any software, however good and efficient it may be. Currently as given in limitation we just deal with text communication. In future the software can be extended to include services like

- File transfer: Allowing users to transfer files.
- Image share: Allowing users to share images in chat window. This may also include use of emoticons.
- Maintain a database on server side and provide password for users.
- Display list of both online and offline users.
- Use different data types for communication between client and server.
- Create separate chat groups.
- Voice and video chatting.
- Save offline messages.

5. Summary

The “Chat application” allows multiple users to exchange information from different locations. This project helped in understanding the client-server model, how sockets work and what problems can be faced when handling multiple connections.

The report gives the detailed design of the chat application and how each module was tested for its correctness and functionality. The report also lists down various enhancements that can be done. The project helped in understanding why testing the software is important part of development and if not done on time, how it can affect the entire functionality of the project.

6. Appendices

6.1 References

1. https://docs.oracle.com/javase/tutorial/networking/sockets/

2. https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

3. http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

4. https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

6.2 Test Cases and Test Results

- Test Case 1

  Test Description:

  To verify login page user interface. Check the alignment of UI components. Username Label, username input text field, close/minimize/connect buttons are displayed.

  Steps:

  Launch the client application.

  Expected Result:

  Username text field should be visible and editable.
  Connect/close/minimize buttons should be visible.

  Actual Result:

  Login page is displayed with all UI components.

  Status: PASS

- Test Case 2

  Test Description:

  Username is greater than 8 characters.

  Steps:

  Launch the client application.

  Enter username with more than 8 characters.

  Click on connect button.

  Expected Result:

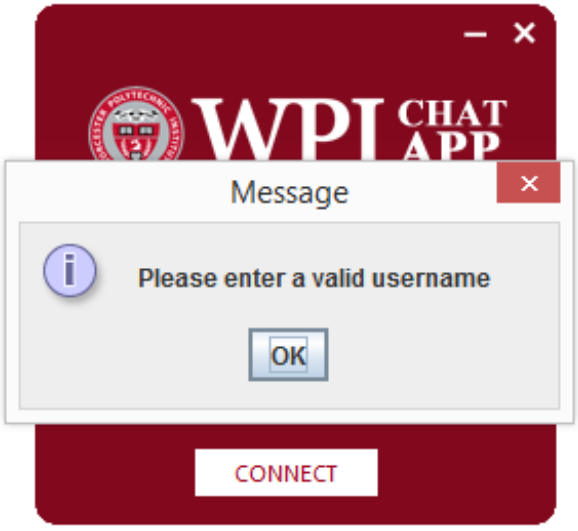
  Display error popup for maximum number of characters for username.

  Actual Result:

  Popup is shown with error message.

  Status: PASS

- Test Case 3
  Test Description:
  Username is empty
  Steps:
  Launch the client application.
  Do not enter username.
  Click on connect button.
  Expected Result:
  Display error popup for invalid username.
  Actual Result:
  Popup is shown with error message.
  Status: PASS



- Test Case 4

  Test Description:

  Username is not unique.

  Steps:

  Launch the server application and click on start server button.
  Launch the client application.
  Enter username as “Neha”.
  Click on connect button.
  Launch another client application.
  Enter username again as “Neha”.
  Click on connect button.

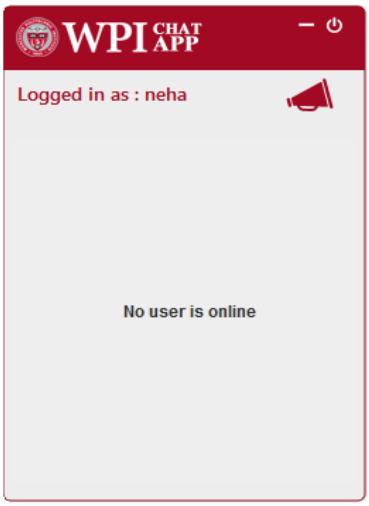
  Expected Result:

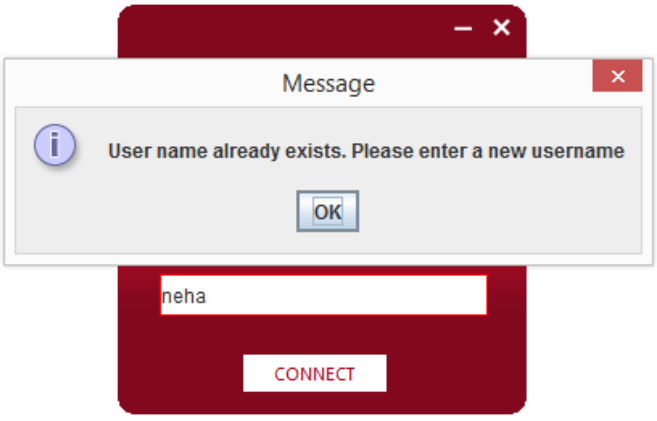
  If username not unique display error message popup and ask for new username.

  Actual Result:

  Popup is shown with error message.

  Status: PASS





- Test Case 5

  Test Description:

  Username is unique.

  Steps:

  Launch the server application and click on start server button.

  Launch the client application.

  Enter unique username as “Neha”. Make sure there is no other client whose username is “Neha”.

  Click on connect button.

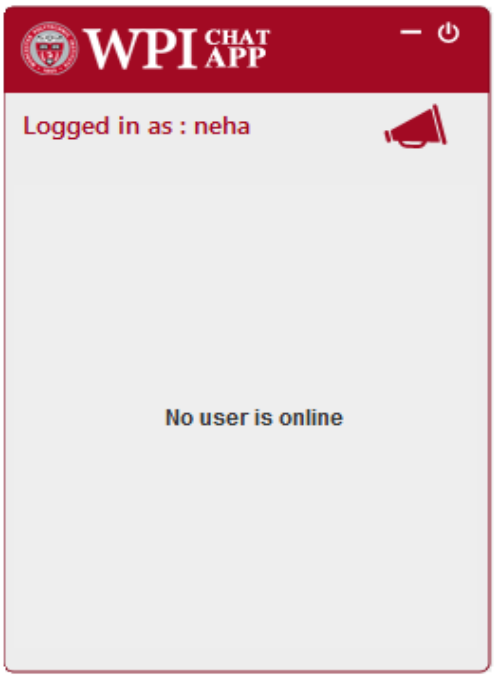
  Expected Result:

  User must be navigated to online user’s list UI page.

  Actual Result:

  Online user’s list UI page is displayed.

  Status: PASS



- Test Case 6

  Test Description:

  Check if clients are updated when new user joins.

  Steps:

  Launch the server application and click on start server button.
  Launch the client application.
  Enter unique username as “Neha”.
  Click on connect button.
  Launch the client application.
  Enter unique username as “Poonam”.
  Click on connect button.

  Expected Result:

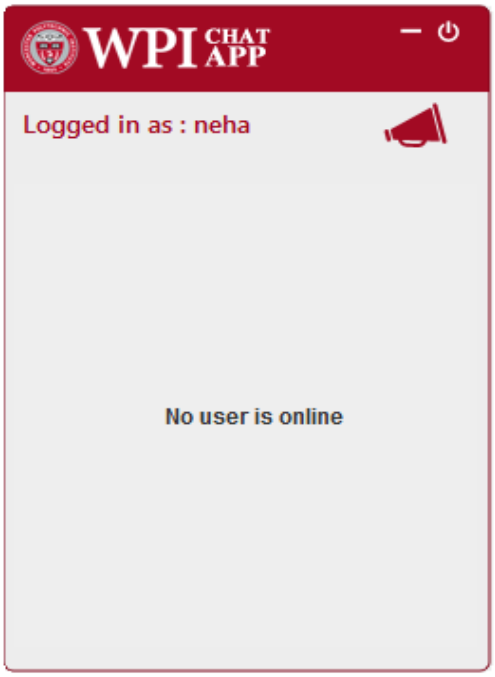
  New user must be displayed to already connected users.
  New user must be navigated to online user’s list UI page.
  List of online users must be visible on online user’s list UI page of new user.

  Actual Result:

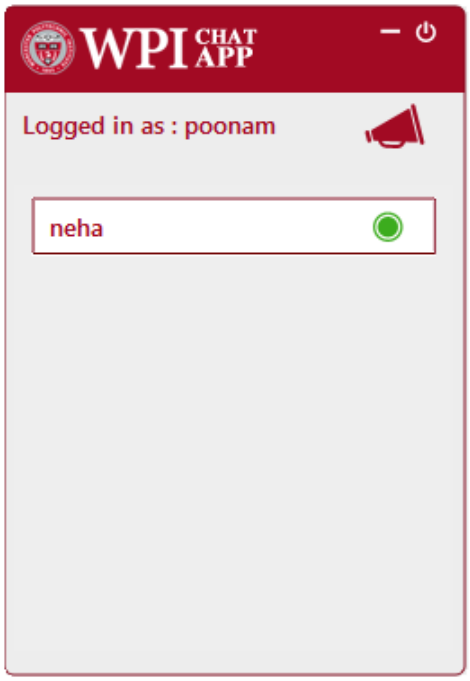
  Online user’s list UI page is displayed.

  Status: PASS

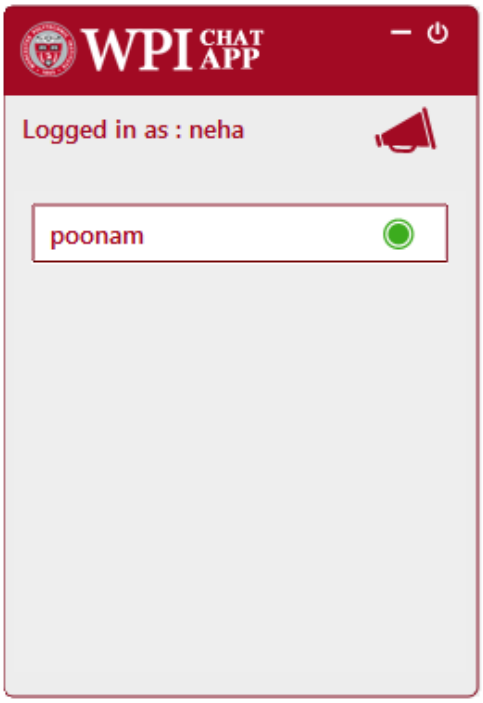
User “Neha Logs in”



User “Poonam Logs in”



User “Neha” is visible in Online Users List UI page of user “Poonam”



User “Poonam” is visible in Online Users List UI page of user “Neha”

- Test Case 7

  Test Description:

  Message send by one client is received by another client (whisper).

  Steps:

  Launch the server application and click on start server button.
  Launch the client application.
  Enter unique username as “Neha”.
  Click on connect button.
  Launch the client application.
  Enter unique username as “Poonam”.
  Click on connect button.
  Online user’s list UI page is visible. Click on “Neha” to open a chat window.
  Type a message in output text field and click on send button.

  Expected Result:

  Message send must be displayed in the input text area of both sender and receiver.
  If chat window on receiver side is not open, chat window opens and message is displayed.

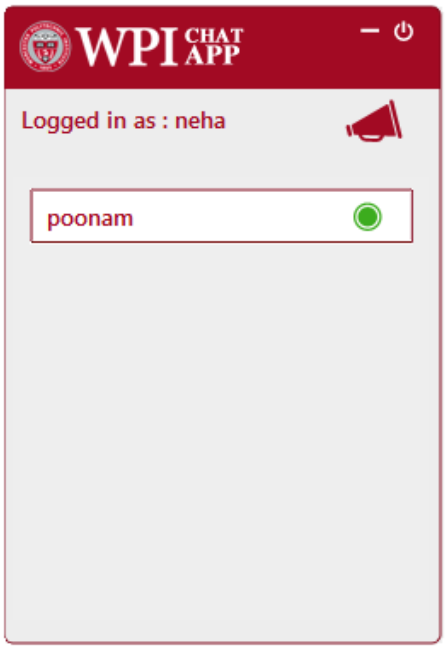
  Actual Result:

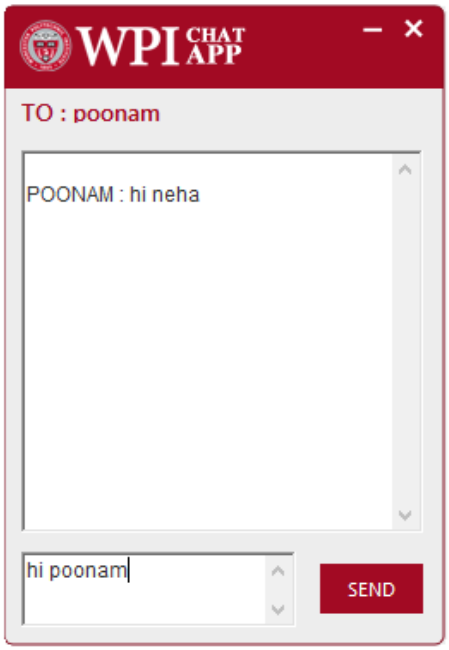
  Message is displayed.

  Status: PASS

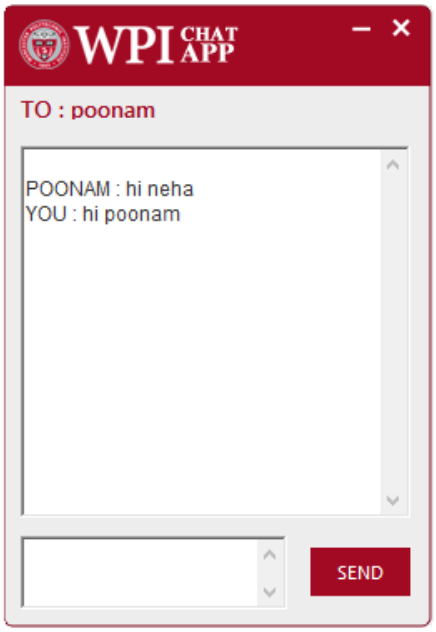
“Poonam” is logged in. Online Users list UI page contains “Neha” as online user.

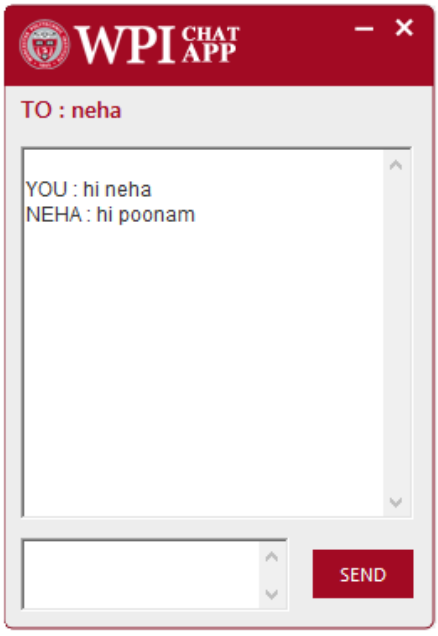
Chat window for “Neha” opens. Message is typed and displayed in the input text are.





Message typed by “Poonam” is displayed in the output text of chat window for “Neha”. “Neha” type’s message back and clicks on send button.





Messages typed by “Neha”, is displayed in chat window for both “Neha” and “Poonam”.

- Test Case 8

  Test Description:

  Message send by one client is received by all other clients (broadcast).

  Steps:

  Launch the server application and click on start server button.
  Launch the client application.
  Enter unique username as “Neha”.
  Click on connect button.
  Launch the client application.
  Enter unique username as “Poonam”.
  Click on connect button.
  Launch the client application.
  Enter unique username as “Satish”.
  Click on connect button.
  Let anyone open broadcast chat window by clicking on broadcast icon on online user’s list UI page.
  Type a message in output text field of broadcast chat window and click on send button.

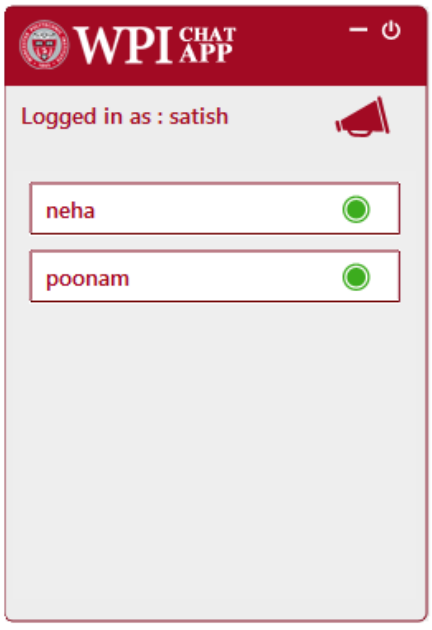
  Expected Result:

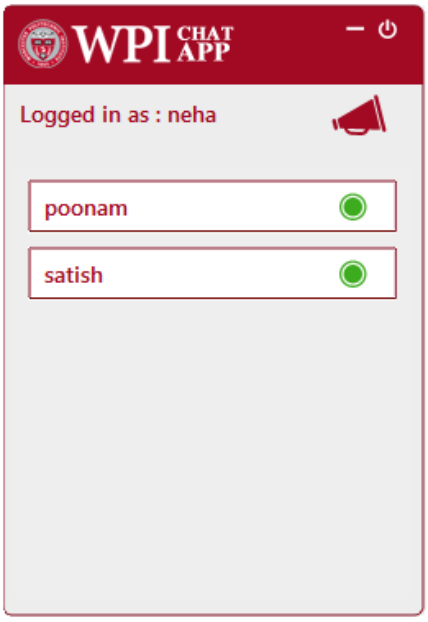
  Message send by anyone is seen in the broadcast chat window for all online users.

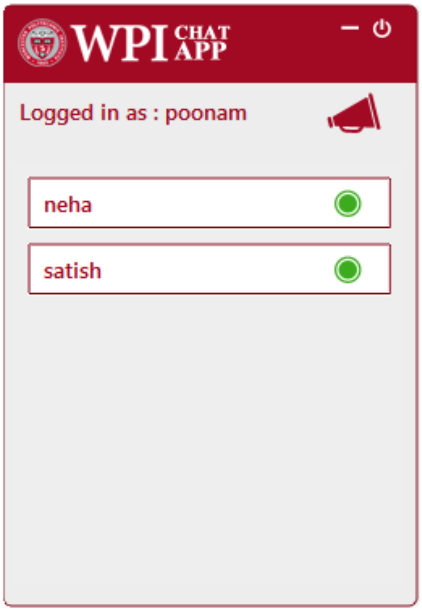
  Actual Result:

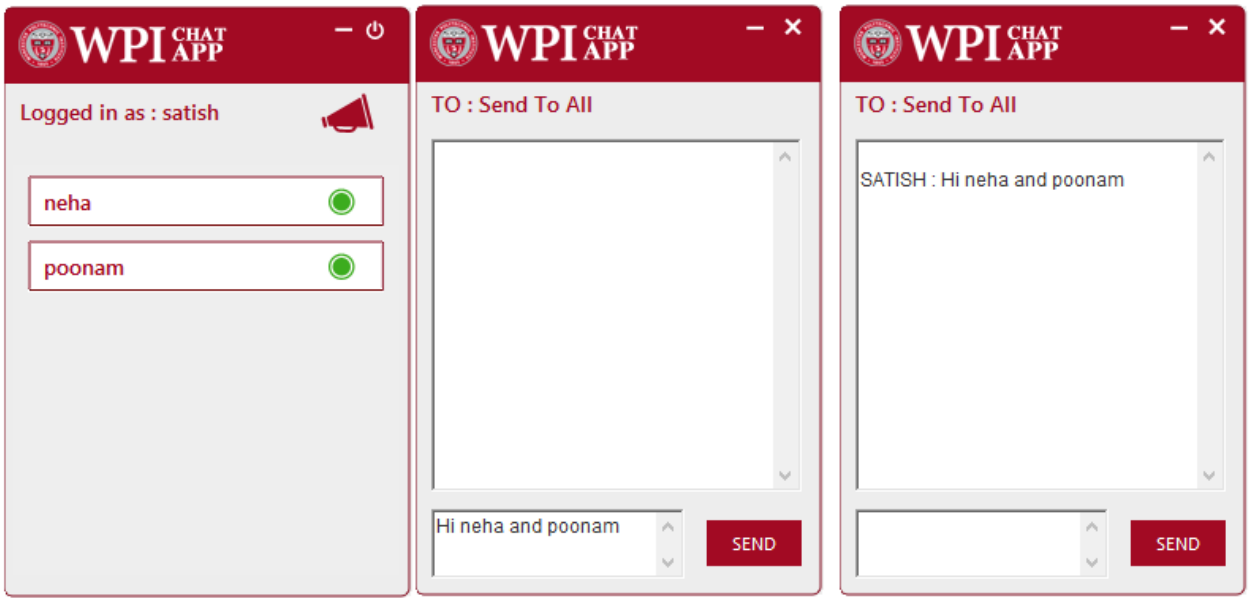
  Message is displayed in broadcast window for all online users.

  Status: PASS

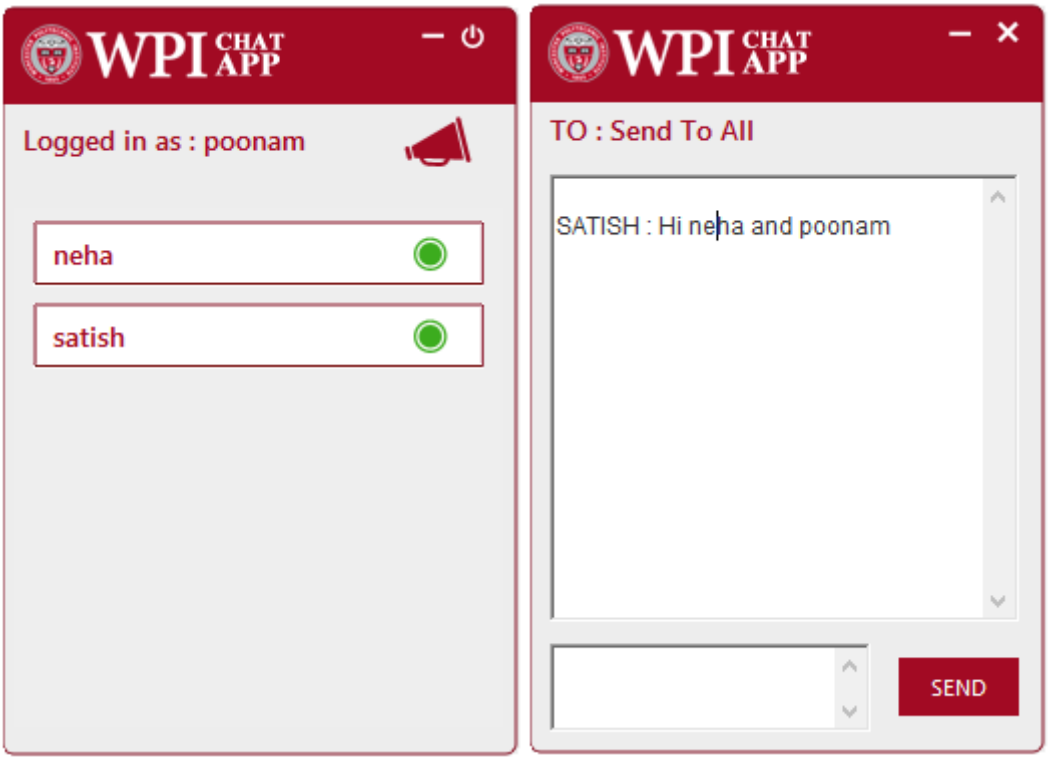




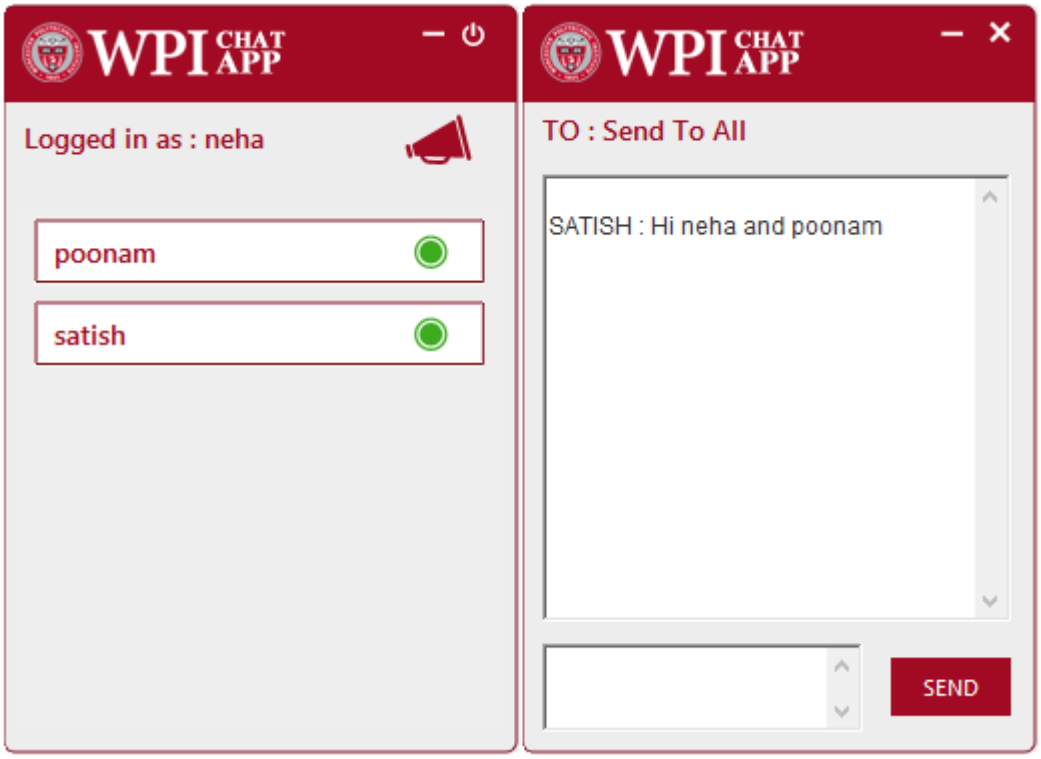




Logged in as “Satish”. Broadcast chat window is opened. Message is typed. Message displayed in the output text area.



Message displayed in broadcast chat window for “Poonam”



Message displayed in broadcast chat window for “Neha”

- Test Case 9

  Test Description:

  User clicks send button without typing a message.

  Steps:

  Launch the server application and click on start server button.
  Launch the client application.
  Enter unique username as “Neha”.
  Click on connect button.
  Launch the client application.
  Enter unique username as “Poonam”.
  Click on connect button.
  Click on “Poonam” button in “Neha” online user list UI page.
  Click on send button without typing anything in the input text area.

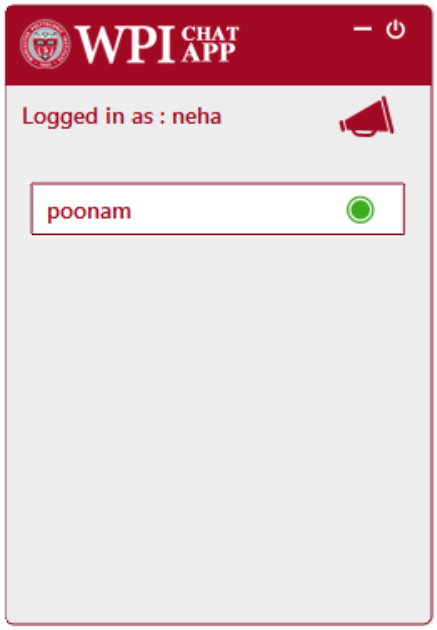
  Expected Result:

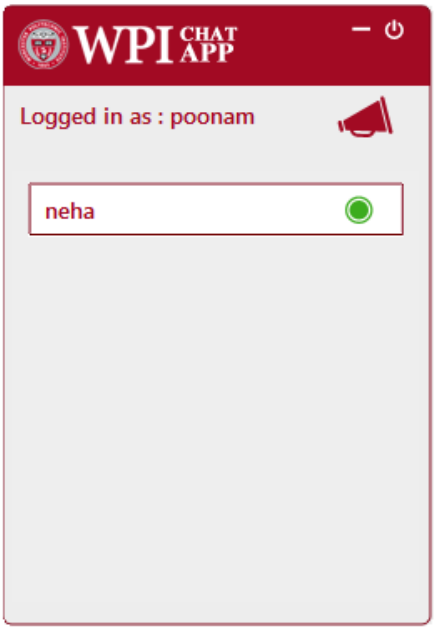
  Error message is shown in popup.

  Actual Result:

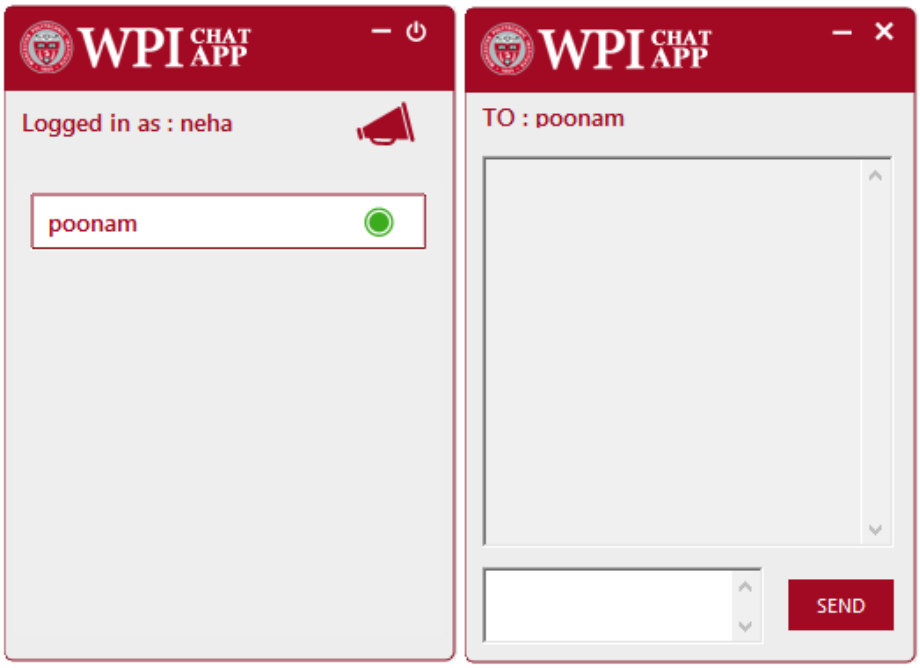
  Error message is displayed.

  Status: PASS

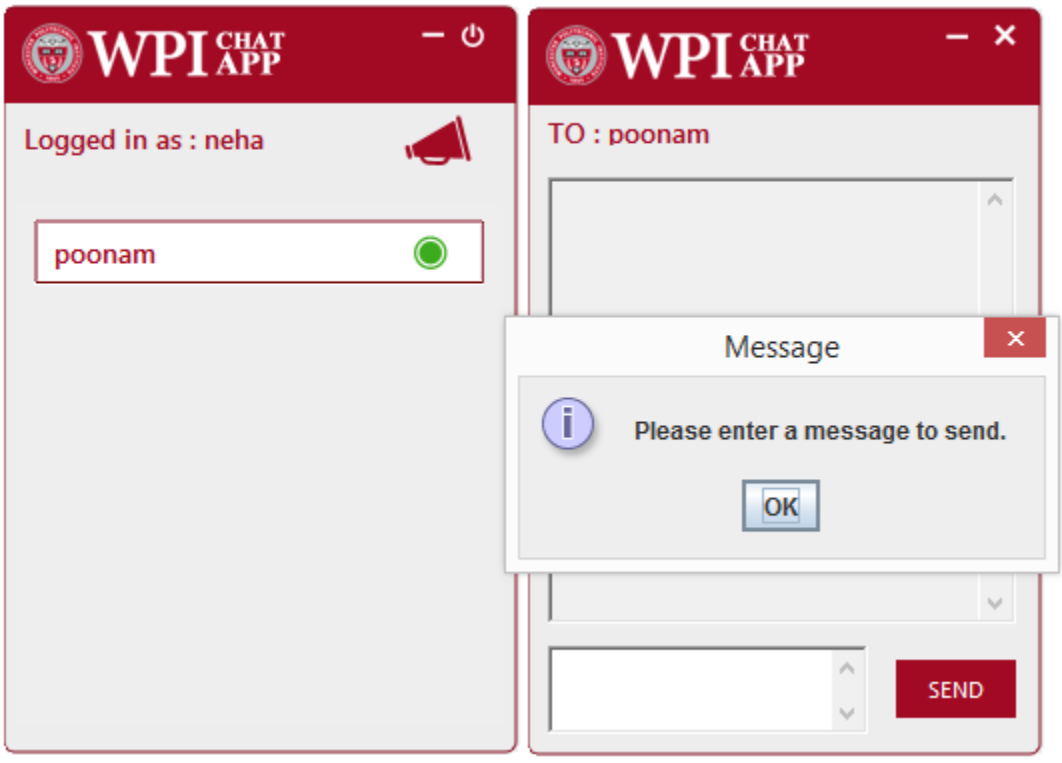




Both users are online



On click on “Poonam” chat window is opened. Now click on send button.



Error message is displayed in popup.

- Test Case 10

  Test Description:

  If user logs off, and chat window is open for that user in another user application the offline message cannot be sent.

  Steps:

  Launch the server application and click on start server button.
  Launch the client application.
  Enter unique username as “Neha”.
  Click on connect button.
  Launch the client application.
  Enter unique username as “Poonam”.
  Click on connect button.
  Open chat window for “Poonam” from “Neha’s” chat application.
  Exchange few messages.
  Now disconnect “Poonam” by clicking on log off button in the online user’s list UI page.

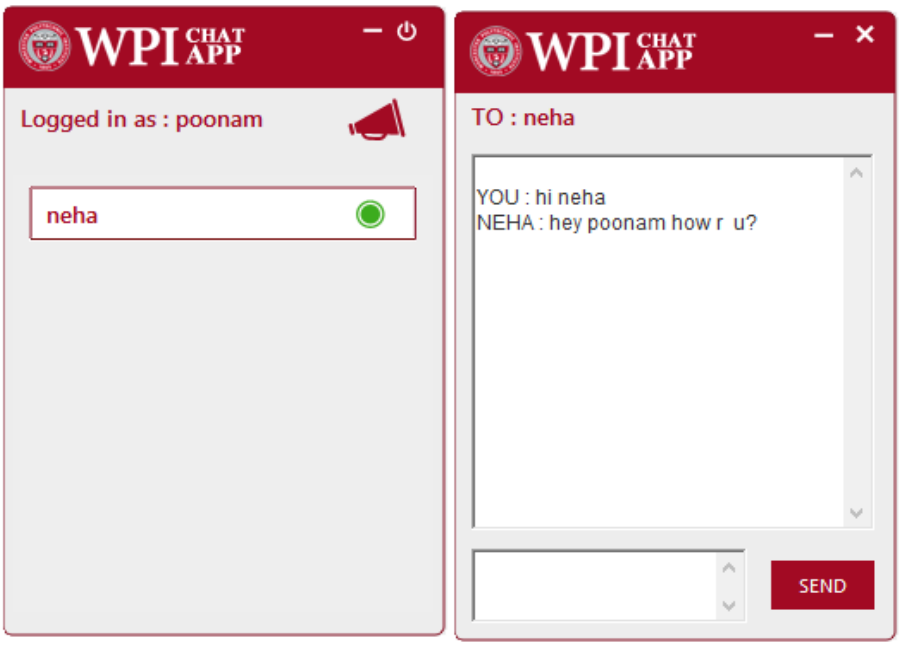
  Expected Result:

  Send button should not be clickable and user is offline message must be displayed in the output text area for “Neha’s” application.
  “Poonam” must be removed from the online user’s list for “Neha” application.

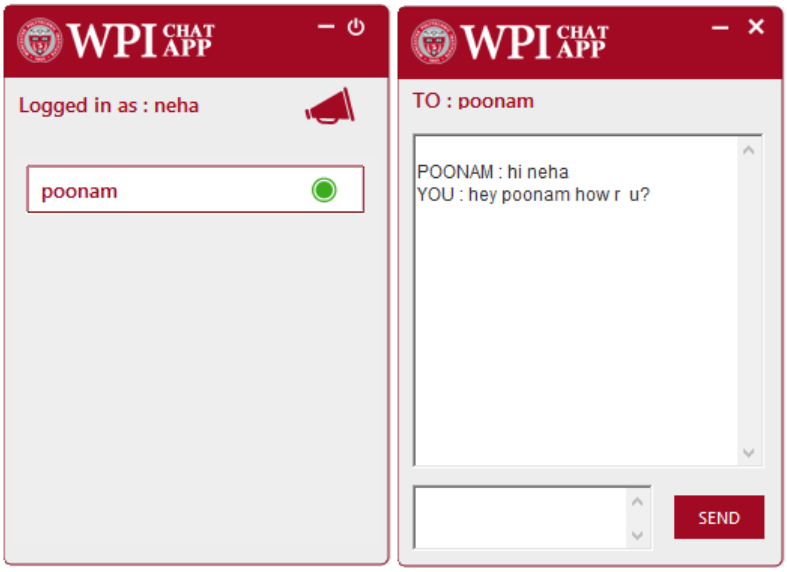
  Actual Result:

  Send button is disabled and offline message is displayed

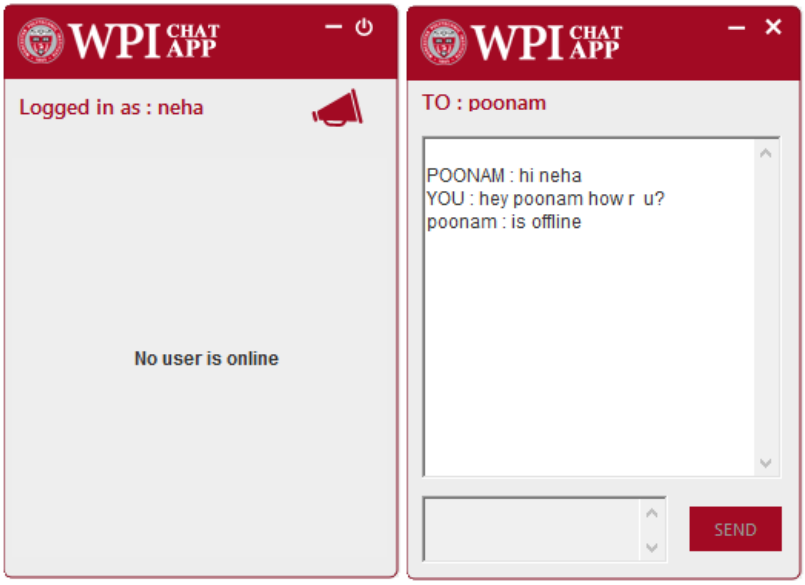
  Status: PASS



Chat application of “Poonam”



Chat application of "Neha"



"Poonam" has logged off. "Neha" is updated and "Poonam" is removed from online user's list. Send button is disabled and offline status is shown.

- Test Case 11

  Test Description:

  Client disconnects.

  Steps:

  Launch the server application and click on start server button.
  Launch the client application.
  Enter unique username as “Neha”.
  Click on connect button.
  Click on logoff button.

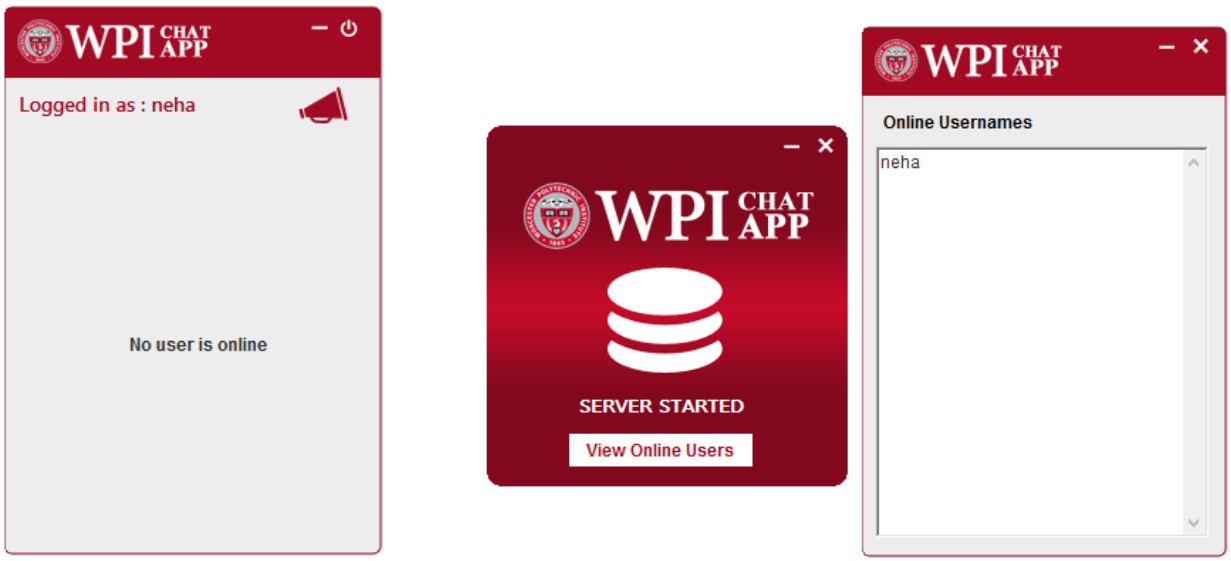
  Expected Result:

  Client must be removed from server list and login page for the client is displayed.

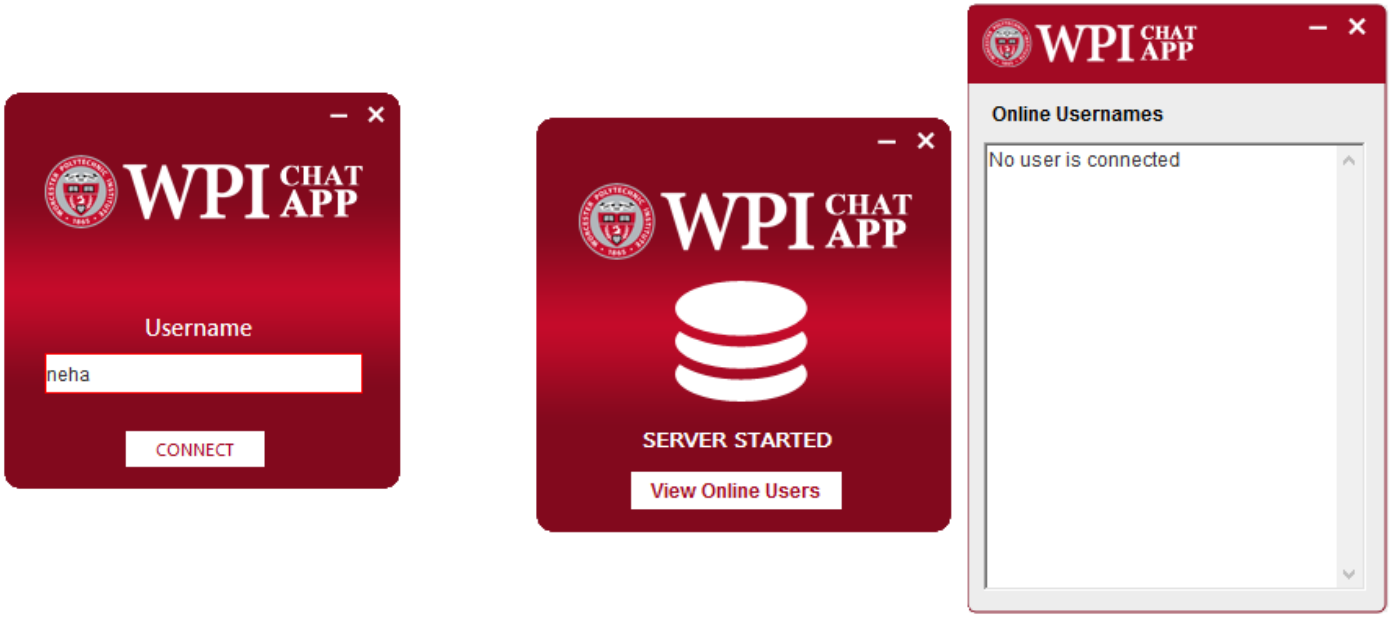
  Actual Result:

  Login page is shown. Client removed from server list.

  Status: PASS



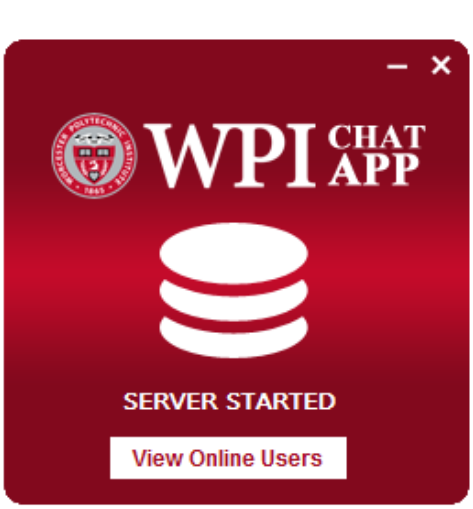
User “Neha is online”. On server “Neha” is shown in the online usernames.

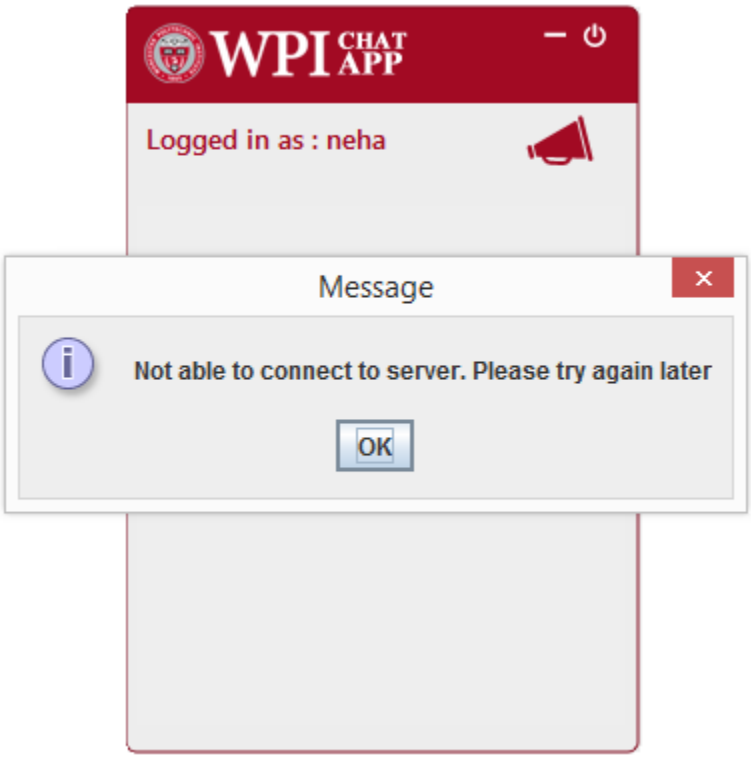


Login page for “Neha” is shown. Server online list is updated.

- Test Case 12
  Test Description:
  Server disconnects.
  Steps:
  Launch the server application and click on start server button.
  Launch the client application.
  Enter unique username as “Neha”.
  Click on connect button.
  Close the server by clicking the close button.
  Expected Result:
  Error message in a popup must be displayed to client. On click of ok button client application must terminate.
  Actual Result:
  Error message is shown in the popup. Clicking on ok buttons terminates the client application.

  Status: PASS


WPI CHAT APP
SERVER STARTED
View Online Users


WPI CHAT APP
Logged in as : neha
Message
Not able to connect to server. Please try again later
OK

6.3.1 ChatServer.java

To load the entire project just import the project in the eclipse IDE and run the ChatServer.java and WPIChatClient.java file. I have attached the zip file of the project on my wpi. Images are used in the project and hence all folders must be at the same level as given in the zip file.

```
	import java.awt.EventQueue;

/**
 * Main launch point of the server application.
 * @author Neha Mahajan.
 */
public class ChatServer {

	private JFrame frame;
	private JLabel lblStartServer;
	private JButton btnOk,listUsersBtn;

	// ArrayList to contain client sockets is created.
	public ArrayList<SocketUtil> UserSockets = new ArrayList<SocketUtil>();

	//ArrayList to contain client threads is created.
	public ArrayList<ClientThread> ClientThreads = new
ArrayList<ClientThread>();
	private int pX, pY;

	// Contains the reference to online users list UI object.
	public ServerOnlineUserListUI serverOnlineUserListUI = null;

	// Contains the reference to ConnectionListener object.
	private ConnectionListener connectionListener;
	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					ChatServer window = new ChatServer();
					window.frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Constructor.
	 * Calls method to draw UI components.
	 * @param none
	 * @return none
	 */
	public ChatServer() {
```

```
			initialize();
		}

		/**
		 * Creates the main JFrame and adds all UI components to it.
		 * Contains function calls for close,minimize,start server and list
online users button.
		 * Drag functionality for JFrame is implemented in the function.
		 * @param none
		 * @return none
		 */
		private void initialize() {
			frame = new JFrame();
			frame.addWindowListener(new WindowAdapter() {
				@Override
				public void windowClosing(WindowEvent we) {
					try {
						if(connectionListener != null){
							connectionListener.serverSocketClosed =
true;
							connectionListener.closeServerSocket();
						}
						System.exit(0);
					} catch (IOException e) {
					}
				}
			});
			frame.setBounds(550, 220, 254, 254);
			//frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			frame.getContentPane().setLayout(null);
			frame.setUndecorated(true);

			frame.addComponentListener(new ComponentAdapter() {
			@Override
			 public void componentResized(ComponentEvent e) {
			     frame.setShape(new RoundRectangle2D.Double(0, 0,
frame.getWidth(), frame.getHeight(), 20, 20));
			 }
		});

			lblStartServer = new
JLabel(StringLiterals.START_SERVER,SwingConstants.CENTER);
			lblStartServer.setFont(new Font("Segoe UI Symbol", Font.BOLD,
13));
			lblStartServer.setForeground(Color.WHITE);
			lblStartServer.setBounds(42, 190, 163, 14);
			frame.getContentPane().add(lblStartServer);

			btnOk = new JButton(StringLiterals.OK);
			btnOk.setBorder(null);
			btnOk.setBackground(Color.WHITE);
			btnOk.setForeground(new Color(162, 10, 35));
			btnOk.setFocusPainted(false);
			btnOk.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent ae) {
					btnOk.setVisible(false);
					listUsersBtn.setVisible(true);
```

```
		lblStartServer.setText(StringLiterals.SERVER_STARTED);
                        startServer();
                }
            });
            btnOk.setBounds(78, 217, 89, 23);
            frame.getContentPane().add(btnOk);

            listUsersBtn = new
JButton(StringLiterals.SERVER_ONLINE_USERS_BUTTON);
            listUsersBtn.setVisible(false);
            listUsersBtn.setBorder(null);
            listUsersBtn.setBackground(Color.WHITE);
            listUsersBtn.setForeground(new Color(162, 10, 35));
            listUsersBtn.setFocusPainted(false);
            listUsersBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ae) {
                        if(serverOnlineUserListUI == null){
                              serverOnlineUserListUI = new
ServerOnlineUserListUI();
                        }
                        serverOnlineUserListUI.updateList(UserSockets);
                        serverOnlineUserListUI.showPage();
                  }
            });
            listUsersBtn.setBounds(58, 217, 129, 23);
            frame.getContentPane().add(listUsersBtn);

            JButton minimizeBtn = new JButton();
            minimizeBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent arg0) {
                        frame.setState(Frame.ICONIFIED);
                  }
            });
            minimizeBtn.setBounds(205, 5, 20, 20);
            minimizeBtn.setBorder(null);
            minimizeBtn.setBorderPainted(false);
            minimizeBtn.setOpaque(false);
            minimizeBtn.setContentAreaFilled(false);
            ImageIcon minimizeImage = new ImageIcon("images\\minimize.png");
            minimizeBtn.setIcon(minimizeImage);
            frame.getContentPane().add(minimizeBtn);

            JButton closeBtn = new JButton();
            closeBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent arg0) {
                        try {
                              if(connectionListener != null){
                                    connectionListener.serverSocketClosed =
true;
                                    connectionListener.closeServerSocket();
                              }
                        } catch (IOException e) {
                        }
                        WindowEvent closingEvent = new WindowEvent(frame,
WindowEvent.WINDOW_CLOSING);
```

```
	Toolkit.getDefaultToolkit().getSystemEventQueue().postEvent(closingEven
t);
			}
		});
		closeBtn.setBounds(233, 8, 11, 12);
		closeBtn.setBorder(null);
		closeBtn.setBorderPainted(false);
		closeBtn.setOpaque(false);
		closeBtn.setContentAreaFilled(false);
		ImageIcon closeImage = new ImageIcon("images\\close.png");
		closeBtn.setIcon(closeImage);
		frame.getContentPane().add(closeBtn);

		JPanel panel = new JPanel();
		panel.addMouseMotionListener(new MouseMotionAdapter() {
			@Override
			public void mouseDragged(MouseEvent mouseEvent) {

	frame.setLocation(frame.getLocation().x+mouseEvent.getX()-
pX,frame.getLocation().y+mouseEvent.getY()-pY);
			}
		});
		panel.addMouseListener(new MouseAdapter() {
			@Override
			public void mousePressed(MouseEvent mouseEvent) {
				pX=mouseEvent.getX();
			    pY=mouseEvent.getY();
			}
		});
		panel.setBounds(0, 0, 193, 30);
		panel.setOpaque(false);

		frame.getContentPane().add(panel);

		JLabel server_logo = new JLabel();
		server_logo.setBounds(85, 100, 81, 74);
		frame.getContentPane().add(server_logo);
		ImageIcon serverImage = new ImageIcon("images\\server_logo.png");
		server_logo.setIcon(serverImage);

		JLabel wpi_logo = new JLabel();
		wpi_logo.setBounds(26, 40, 204, 47);
		frame.getContentPane().add(wpi_logo);
		ImageIcon wpiLogoImage = new ImageIcon("images\\wpi_logo.png");
		wpi_logo.setIcon(wpiLogoImage);

		JLabel lblNewLabel = new JLabel();
		lblNewLabel.setBounds(0, 0, 254, 254);
		frame.getContentPane().add(lblNewLabel);
		ImageIcon bgImage = new ImageIcon("images\\bg.png");
		lblNewLabel.setIcon(bgImage);
	}

	/**
	 * The function creates an object of ConnectionListener class.
	 * ConnectionListener class creates a new server socket.
```

```
	 * Stores the reference of object in variable connectionListener.
	 * @param none
	 * @return none
	 */
	private void startServer(){
		connectionListener = new ConnectionListener(this);
	}

	/**
	 * This function is called when a client disconnects from server.
	 * It removes the disconnected client thread from the ArrayList.
	 * @param clientThread contains the reference to disconnected client
thread.
	 * @return none
	 */
	public void removeClientThread(ClientThread clientThread) {
		this.ClientThreads.remove(clientThread);
	}
}
```

6.3.2 ConnectionListener.java

```
	import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;


/**
*
* @author Neha Mahajan
*
*/
public class ConnectionListener extends Thread{


	// Contains reference to object of ServerSocket.
	private ServerSocket serverSocket;


	// Contains reference to object of ChatServer class.
```

```
private ChatServer chatServer;
public Boolean serverSocketClosed = false;

/**
 * Constructor.
 * Initializes the ChatServer object.
 * @param parentChatServer contains reference to the parent class.
 * @return none
 */
public ConnectionListener(ChatServer parentChatServer){
        this.chatServer = parentChatServer;
        this.start();
}

/**
 * Overrides the run method of thread class.
 * Creates a new server socket and wait for client connections.
 * On new client connection creates new client thread and adds thread to the ArrayList.
 * Calls run method of newly created client thread.
 * Handles the exception when server socket is closed or
 * when there is an exception with input/output streams.
 * @param none.
 * @return none.
 * @exception Exception, IOException
 */
public void run(){
```

```
		try
		{
			serverSocket = new ServerSocket(9001);
				while(true){
					if(serverSocketClosed == true)
						break;

					Socket tempSocket = serverSocket.accept();
					ClientThread tempClientThread = new
ClientThread(tempSocket, this.chatServer);
					tempClientThread.start();
					chatServer.ClientThreads.add(tempClientThread);
				}
		}catch (Exception e){
		}finally{
			try {
				if(!serverSocket.isClosed())
					serverSocket.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}

	/**
	 * The function is called to close a server socket.
```

```
        * Calls the close method of server socket.
        * @param none.
        * @return none.
        */
        public void closeServerSocket() throws IOException{
                this.serverSocket.close();
        }
}
```

6.3.3 ClientThread.java

```
import java.io.IOException;
import java.net.Socket;


/**
 *
 * @author Neha Mahajan
 *
 */
public class ClientThread extends Thread{


        // Contains the reference to client socket object.
   private SocketUtil SocketUtil;
        private boolean disconnect = false;


        // Contains reference to object of ChatServer class.
        private ChatServer chatServer;
```

```
/**
 * Constructor.
 * Initializes the ChatServer object.
 * Creates a new SocketUtil object with reference to socket object created for the client
 * by the ConnectionListener class.
 * @param   socket contains reference to socket object.
 *                          parentChatServer contains reference to the parent class.
 * @return none
 */
public ClientThread(Socket socket, ChatServer parentChatServer) throws Exception{
        this.chatServer = parentChatServer;
        SocketUtil = new SocketUtil(socket);
}

/**
 * Overrides the run method of thread class.
 * Handles messages received in socket input stream.
 * Adds socket to ArrayList of Sockets.
 * Calls functions like broadcast and send message.
 * Handles client disconnect.
 * @param none
 * @return none
 */
public void run(){
        try{
                while(true){
```

```
                String input = SocketUtil.getMessage();
                String[] outputMessage = input.split(":");

                switch(outputMessage[0]){
                        case StringLiterals.USERNAME:
                                if(checkIfUsernameExists(outputMessage[1]) ==
true){
        SocketUtil.sendMessage(StringLiterals.USER_NOT_ACCEPTED + ":");
                                } else {
                                        SocketUtil.setUsername(outputMessage[1]);
                                        chatServer.UserSockets.add(SocketUtil);
                                        broadcastOnlineUsers();
                                }
                        break;

                        case StringLiterals.FROM:
                                sendMessageToUser(input);
                        break;

                        case StringLiterals.DISCONNECT:
                                this.disconnect  = true;
                        break;
                }
                if(this.disconnect)
                        break;
        }
```

```
                if(this.disconnect)
                        disconnectClient();
        }catch (IOException e){
                e.printStackTrace();
        }
}

/**
 * Closes the client socket.
 * Removes socket entry from ArrayList.
 * Updates all other online users.
 * Initiates client thread removal.
 * @param none
 * @return none
 */
private void disconnectClient() {
        SocketUtil.closeSocket();
        chatServer.UserSockets.remove(SocketUtil);
        try {
                broadcastOnlineUsers();
                this.chatServer.removeClientThread(this);
        } catch (IOException e) {
                e.printStackTrace();
        }
}
```

```
/**
 * Sends the message back to the sender to be displayed in the chat window.
 * @param input contains the messages written by the client.
 * @return none
 */
private void sendMessageToFromUser(String input) {
        SocketUtil.sendMessage(input);
}

/**
 * The function validates the username to check if it is unique.
 * It checks in the ArrayList of sockets and if username does not exists returns false.
 * If ArrayList contains username then return true.
 * @param username contains the username(String) provided by the client.
 * @return boolean
 */
private Boolean checkIfUsernameExists(String username){
        SocketUtil tempUser;

        for(int i = 0; i < chatServer.UserSockets.size(); i++){
                tempUser = chatServer.UserSockets.get(i);
                if(tempUser.getUsername().equals(username))
                        return true;
        }

        return false;
```

```
}

/**
 * When a new client connects and username is accepted this function is called to
 * inform all the other online users about the newly connected clients.
 * The function access all the user sockets from the ArrayList and writes
 * the list of online users as string to the output streams.
 * It also calls method of online user list UI to update the list.
 * @param none
 * @return none
 */
private void broadcastOnlineUsers() throws IOException{
        if(chatServer.serverOnlineUserListUI != null){
                chatServer.serverOnlineUserListUI.updateList(chatServer.UserSockets);
        }

        String tempString = StringLiterals.ONLINE_USERNAMES + ":";
        SocketUtil tempUser;

        for(int i = 0; i < chatServer.UserSockets.size(); i++){
                tempUser = chatServer.UserSockets.get(i);
                tempString += tempUser.getUsername();
                if(i < chatServer.UserSockets.size() - 1)
                        tempString += ":";
        }
```

```
            for(int i = 0; i < chatServer.UserSockets.size(); i++){
                    tempUser = chatServer.UserSockets.get(i);
                    tempUser.sendMessage(tempString);
            }
    }

    /**
     * Handles the forwarding of the messages.
     * Sends message to all users including the sender if it is broadcast message.
     * Sends message to sender and receiver only if a whisper message.
     * @param inputMessage contains the message(String) from the sender.
     * @return none
     */
    private void sendMessageToUser(String inputMesage) throws IOException{
            String[] messageArray = inputMesage.split(":");
            SocketUtil tempToUser;

            if(messageArray[3].equals(StringLiterals.SEND_TO_ALL)){
                    for(int i = 0; i < chatServer.UserSockets.size(); i++){
                            tempToUser = chatServer.UserSockets.get(i);
                            tempToUser.sendMessage(inputMesage);
                    }
            } else {
                    sendMessageToFromUser(inputMesage);
                    for(int i = 0; i < chatServer.UserSockets.size(); i++){
                            tempToUser = chatServer.UserSockets.get(i);
```

```
                                if(tempToUser.getUsername().equals(messageArray[3])){
                                        tempToUser.sendMessage(inputMesage);
                                        break;
                                }
                        }
                }
        }
}
```

6.3.4 SocketUtil.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

/**
*
 * @author Neha Mahajan
*
*/
public class SocketUtil {
        private String username ="";

        // Contains reference to socket object
        private Socket socket;
```

```
        // Contains reference to output stream of socket
        private PrintWriter messageOut;

        // Contains reference to input stream of socket
    private BufferedReader messageIn;

    /**
         * Constructor.
         * @param   socket contains reference to socket object.
         * @return none
         */
        public SocketUtil(Socket socket) throws IOException {
                setSocket(socket);
        }

        /**
         * Creates the output and input stream for the socket.
         * Stores the reference of the socket object.
         * @param   socket contains reference to socket object.
         * @return none
         */
        public void setSocket(Socket socket) throws IOException {
                this.socket = socket;
                this.messageOut = new PrintWriter(this.socket.getOutputStream(), true);
                this.messageIn = new BufferedReader(new
InputStreamReader(this.socket.getInputStream()));
```

```
}

/**
 * Returns socket object.
 * @param none.
 * @return socket.
 */
public Socket getSocket(){
        return this.socket;
}

/**
 * Initializes the username.
 * @param username contains the username(String) provided by the client.
 * @return none.
 */
public void setUsername(String username) {
        this.username = username;
}

/**
 * Returns username.
 * @param none.
 * @return username.
 */
public String getUsername(){
```

```
        return this.username;
}

/**
 * Writes the message to output stream.
 * @param message String.
 * @return none.
 */
public void sendMessage(String message){
        this.messageOut.println(message);
        this.messageOut.flush();
}

/**
 * Writes the message to output stream.
 * @param message String.
 * @return message returns message(String) received in the input stream.
 * @exception IOException
 */
public String getMessage() throws IOException{
        return this.messageIn.readLine();
}

/**
 * Closes the client socket.
 * @param none.
```

```
	* @return none.
	* @exception IOException
	*/
	public void closeSocket(){
		try {
			this.socket.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

6.3.5 ServerOnlineUserListUI.java

```
import java.awt.Frame;

/**
 *
 * @author Neha Mahajan
 *
 */
public class ServerOnlineUserListUI {

	private JFrame frame;
	private TextArea usersListTextArea;
	private int pX, pY;
	/**
	 * Create the application.
	 */
	public ServerOnlineUserListUI() {
		initialize();
	}

	/**
	 * Creates the JFrame and adds all UI components to it.
	 * Contains function call for minimize button.
	 * Drag functionality for JFrame is implemented in the function.
	 * @param none
	 * @return none
	 */
	private void initialize() {
		frame = new JFrame();
		frame.setBounds(550, 220, 258, 374);
		frame.setUndecorated(true);
```

```
            frame.setShape(new RoundRectangle2D.Double(0, 0,
frame.getWidth(), frame.getHeight(), 10, 10));
            frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
            frame.getContentPane().setLayout(null);

            JButton closeBtn = new JButton();
            closeBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent arg0) {
                        hidePage();
                  }
            });

            JButton minimizeBtn = new JButton();
            minimizeBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent arg0) {
                        frame.setState(Frame.ICONIFIED);
                  }
            });
            minimizeBtn.setBounds(205, 5, 20, 20);
            minimizeBtn.setBorder(null);
            minimizeBtn.setBorderPainted(false);
            minimizeBtn.setOpaque(false);
            minimizeBtn.setContentAreaFilled(false);
            ImageIcon minimizeImage = new ImageIcon("images\\minimize.png");
            minimizeBtn.setIcon(minimizeImage);
            frame.getContentPane().add(minimizeBtn);

            closeBtn.setBounds(233, 8, 11, 12);
            closeBtn.setBorder(null);
            closeBtn.setBorderPainted(false);
            closeBtn.setOpaque(false);
            closeBtn.setContentAreaFilled(false);
            ImageIcon closeImage = new ImageIcon("images\\close.png");
            closeBtn.setIcon(closeImage);
            frame.getContentPane().add(closeBtn);

            JPanel panel = new JPanel();
            panel.addMouseMotionListener(new MouseMotionAdapter() {
                  @Override
                  public void mouseDragged(MouseEvent mouseEvent) {

      frame.setLocation(frame.getLocation().x+mouseEvent.getX()-
pX,frame.getLocation().y+mouseEvent.getY()-pY);
                  }
            });
            panel.addMouseListener(new MouseAdapter() {
                  @Override
                  public void mousePressed(MouseEvent mouseEvent) {
                        pX=mouseEvent.getX();
                    pY=mouseEvent.getY();
                  }
            });
            panel.setBounds(0, 0, 200, 40);
            panel.setOpaque(false);
            frame.getContentPane().add(panel);

            JLabel lblNewLabel = new JLabel(StringLiterals.ONLINE_USERNAMES);
```

```
            lblNewLabel.setForeground(Color.BLACK);
            lblNewLabel.setBounds(15, 60, 120, 14);
            frame.getContentPane().add(lblNewLabel);

            usersListTextArea = new TextArea("",2,50,
TextArea.SCROLLBARS_VERTICAL_ONLY);
            usersListTextArea.setEditable(false);
            usersListTextArea.setBounds(10, 85, 234, 275);
            frame.getContentPane().add(usersListTextArea);

            JLabel listUI = new JLabel();
            ImageIcon listUIbgImage = new
ImageIcon("images\\online_user_bg.png");
            listUI.setIcon(listUIbgImage);
            listUI.setBounds(0, 0, 258, 374);
            frame.getContentPane().add(listUI);

            frame.setVisible(true);
      }

      /**
       * Displays the updated list of online users.
       * @param userSockets contains the reference to ArrayList of user
sockets created in ChatServer class.
       * @return none
       */
      public void updateList(ArrayList<SocketUtil> userSockets){
            if(userSockets.size() == 0){
                  usersListTextArea.setText(StringLiterals.SERVER_NO_USER);
            }
            else{
                  String temp = "";
                  for(int i = 0; i < userSockets.size(); i++){
                        SocketUtil tempToUser = userSockets.get(i);
                        temp += tempToUser.getUsername() + "\n";
                  }

                  usersListTextArea.setText(temp);
            }
      }

      /**
       * Hides the JFrame.
       * @param none.
       * @return none
       */
      public void hidePage() {
            this.frame.setVisible(false);
      }

      /**
       * Displays the JFrame.
       * @param none.
       * @return none
       */
      public void showPage() {
            this.frame.setVisible(true);
```

```
	}
}
```

6.3.6 StringLiterals.java

```
/**
 * Contains list of all static Strings used in both client and server
classes.
 * @author Neha Mahajan
 *
 */
public class StringLiterals {
	public static final String USER_NOT_ACCEPTED = "User Not Accepted";
	public static final String FROM = "From";
	public static final String TO = "To";
	public static final String ONLINE_USERNAMES = "Online Usernames";
	public static final String START_SERVER = "START SERVER";
	public static final String OK = "OK";
	public static final String USERNAME = "Username";
	public static final String CONNECT = "CONNECT";
	public static final String ERROR_NEW_USER = "User name already exists.
Please enter a new username";
	public static final String ONLINE = "Online";
	public static final String SEND = "SEND";
	public static final String SERVER_STARTED = "SERVER STARTED";
	public static final String NO_USER_ONLINE = "Oops no user is online to
send message";
	public static final String SEND_TO_ALL = "Send To All";
	public static final String LOGGED_IN_AS = "Logged in as : ";
	public static final String YOU = "YOU";
	public static final String DISCONNECT = "DISCONNECT";
	public static final String UNABLE_TO_CONNECT = "Not able to connect to
server. Please try again later";
	public static final String SERVER_ONLINE_USERS_BUTTON = "View Online
Users";
	public static final String SERVER_NO_USER = "No user is connected";
	public static final String ENTER_VALID_USERNAME = "Please enter a valid
username";
	public static final String USERNAME_LENGTH = "Please enter maximum of 8
characters";
	public static final String NO_USER_ONLINE_LABEL = "No user is online";
	public static final String MESSAGE_EMPTY = "Please enter a message to
send.";
}
```

6.3.7 WPIChatClient.java

```
import java.io.IOException;

import javax.swing.JOptionPane;
```

```
/**
 * Main launch point of the client application.
 * @author Neha Mahajan
 *
 */
public class WPIChatClient {

	// Contains reference to object of Login Page.
	private Login login;

	// Contains reference to object of MessageSendRecieve class.
	private MessageSendRecieve messageSendRecieve;

	// Contains reference to object of OnlineUsersListUI class.
	private OnlineUsersListUI onlineUsersListUI;

	public static void main(String[] args) {
		WPIChatClient wpiChatClient = new WPIChatClient();
		wpiChatClient.createLoginPage();
	}

	/**
	 * Create a new login page and store its reference.
	 * @param none
	 * @return none
	 */
```

```
private void createLoginPage(){
        this.login = new Login(this);
}

/**
 * Create a new object of MessageSendRecieve class and store its reference.
 * Calls method of MessageSendRecieve to send username to server.
 * @param username contains the username(String) entered by the client.
 * @return none
 */
public void createClientSocketConnection(String username){
        if(this.messageSendRecieve == null)
                this.messageSendRecieve = new MessageSendRecieve(username, this);
        else {
                this.messageSendRecieve.sendName(username);
        }
}

/**
 * Hides the login page after successful login.
 * Create a new object of OnlineUsersListUI class and store its reference.
 * Displays the OnlineUsersListUI page.
 * Calls method of OnlineUsersListUI to display list of online users.
 * @param username contains the username(String) validated by the server.
 *                 OnlineUsers contains the list of online users(String) returned by the server.
 * @return none
```

```
 */
public void displayOnlineUserUI(String OnlineUsers, String username){
        login.hideLoginPage();

        if(this.onlineUsersListUI == null){
                this.onlineUsersListUI = new OnlineUsersListUI(this);
        }
        this.onlineUsersListUI.updateOnlineUsersList(OnlineUsers,username);
}

/**
 * Calls method of OnlineUsersListUI to display message in the chat window.
 * @param input contains the input message(String) received from the server(sender).
 * @return none
 */
public void displayMessageToUser(String input) {
        this.onlineUsersListUI.displayMessageToUser(input);
}

/**
 * Sends message to server.
 * @param message contains the message(String) typed by the client in the chat window.
 * @return none
 */
public void sendMessage(String message){
        this.messageSendRecieve.sendMessage(message);
```

```
	}

	/**
	 * Displays error message in a popup if username is not unique.
	 * @param none.
	 * @return none
	 */
	public void showErrorMessage() {
		JOptionPane.showMessageDialog(null, StringLiterals.ERROR_NEW_USER);
		login.resetUsernameTextField();
	}

	/**
	 * Sends disconnect message to server.
	 * @param none.
	 * @return none
	 */
	public void disconnectFromServer() throws IOException {
		this.messageSendRecieve.sendMessage(StringLiterals.DISCONNECT + ":");
	}

	/**
	 * Displays the login page after successful disconnect from server.
	 * Removes the references for both MessageSendRecieve and OnlineUsersListUI class objects.
	 * @param none.
```

```
        * @return none
        */
        public void showLoginPage() {
                this.messageSendRecieve = null;
                this.onlineUsersListUI.hidePage();
                this.onlineUsersListUI = null;
                this.login.showPage();
        }
}
```

6.3.8 Login.java

```
import javax.swing.BorderFactory;

/**
 *
 * @author Neha Mahajan
 *
 */
public class Login{

	private JFrame frame;
	private JTextField usernameTextField;

	// Contains the reference to the object of WPIChatClient class.
	private WPIChatClient parent;
	private JLabel bgLabel;
	private int pX, pY;

	/**
	 * Constructor.
	 * @param parent contains reference to WPIChatClient object.
	 * @return none
	 */
	public Login(WPIChatClient parent) {
		this.parent = parent;
		initialize();
	}

	/**
	 * Creates the main JFrame and adds all UI components to it.
	 * Contains function calls for close,minimize and connect button.
	 * Drag functionality for JFrame is implemented in the function.
	 * @param none
	 * @return none
	 */
	private void initialize() {
```

```
            frame = new JFrame();
            frame.setBounds(550, 220, 254, 254);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.getContentPane().setLayout(null);
            frame.setUndecorated(true);
            frame.setShape(new RoundRectangle2D.Double(0, 0,
frame.getWidth(), frame.getHeight(), 20, 20));

            JLabel lblUsername = new JLabel(StringLiterals.USERNAME);
            lblUsername.setFont(new Font("Segoe UI Semibold", Font.BOLD,
15));
            lblUsername.setForeground(Color.WHITE);
            lblUsername.setBounds(90, 142, 74, 14);
            frame.getContentPane().add(lblUsername);

            usernameTextField = new JTextField();
            usernameTextField.setBounds(26, 167, 204, 26);
            Border border = BorderFactory.createLineBorder(Color.RED, 1);
            usernameTextField.setBorder(border);

            frame.getContentPane().add(usernameTextField);
            usernameTextField.setColumns(10);

            JLabel wpi_logo = new JLabel();
            wpi_logo.setBounds(26, 40, 204, 47);
            frame.getContentPane().add(wpi_logo);
            ImageIcon wpiLogoImage = new ImageIcon("images\\wpi_logo.png");
            wpi_logo.setIcon(wpiLogoImage);

            JButton btnConnect = new JButton(StringLiterals.CONNECT);
            btnConnect.setFont(new Font("Segoe UI Symbol", Font.PLAIN, 11));
            btnConnect.setBorder(null);
            btnConnect.setBackground(Color.WHITE);
            btnConnect.setForeground(new Color(162, 10, 35));
            btnConnect.setFocusPainted(false);
            btnConnect.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        if(usernameTextField.getText().equals(""))
                              JOptionPane.showMessageDialog(null,
StringLiterals.ENTER_VALID_USERNAME);
                        else if(usernameTextField.getText().length() > 8){
                              JOptionPane.showMessageDialog(null,
StringLiterals.USERNAME_LENGTH);
                              usernameTextField.setText("");
                        }
                        else

      parent.createClientSocketConnection(usernameTextField.getText());
                  }
            });
            btnConnect.setBounds(78, 217, 89, 23);
            frame.getContentPane().add(btnConnect);

            JButton minimizeBtn = new JButton();
            minimizeBtn.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent arg0) {
                        frame.setState(Frame.ICONIFIED);
```

```
			}
		});
		minimizeBtn.setBounds(205, 5, 20, 20);
		minimizeBtn.setBorder(null);
		minimizeBtn.setBorderPainted(false);
		minimizeBtn.setOpaque(false);
		minimizeBtn.setContentAreaFilled(false);
		ImageIcon minimizeImage = new ImageIcon("images\\minimize.png");
		minimizeBtn.setIcon(minimizeImage);
		frame.getContentPane().add(minimizeBtn);

		JButton closeBtn = new JButton();
		closeBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				WindowEvent closingEvent = new WindowEvent(frame,
WindowEvent.WINDOW_CLOSING);

	Toolkit.getDefaultToolkit().getSystemEventQueue().postEvent(closingEven
t);
			}
		});
		closeBtn.setBounds(233, 8, 11, 12);
		closeBtn.setBorder(null);
		closeBtn.setBorderPainted(false);
		closeBtn.setOpaque(false);
		closeBtn.setContentAreaFilled(false);
		ImageIcon closeImage = new ImageIcon("images\\close.png");
		closeBtn.setIcon(closeImage);
		frame.getContentPane().add(closeBtn);

		JPanel panel = new JPanel();
		panel.addMouseMotionListener(new MouseMotionAdapter() {
			@Override
			public void mouseDragged(MouseEvent mouseEvent) {

	frame.setLocation(frame.getLocation().x+mouseEvent.getX()-
pX,frame.getLocation().y+mouseEvent.getY()-pY);
			}
		});
		panel.addMouseListener(new MouseAdapter() {
			@Override
			public void mousePressed(MouseEvent mouseEvent) {
				pX=mouseEvent.getX();
			pY=mouseEvent.getY();
			}
		});
		panel.setBounds(0, 0, 193, 30);
		panel.setOpaque(false);

		frame.getContentPane().add(panel);

		bgLabel = new JLabel();
		bgLabel.setBounds(0, 0, 254, 254);
		frame.getContentPane().add(bgLabel);
		ImageIcon bgImage = new ImageIcon("images\\bg.png");
		bgLabel.setIcon(bgImage);
		frame.getContentPane().add(bgLabel);
```

```
		frame.setVisible(true);
	}

	/**
	 * Hides the JFrame of the Login Page.
	 * @param none
	 * @return none
	 */
	public void hideLoginPage(){
		frame.setVisible(false);
	}

	/**
	 * Reset the username text field.
	 * @param none
	 * @return none
	 */
	public void resetUsernameTextField() {
		usernameTextField.setText("");
	}

	/**
	 * Displays the JFrame of the Login Page.
	 * @param none
	 * @return none
	 */
	public void showPage() {
		frame.setVisible(true);
	}
}
```

6.3.9 MessageSendRecieve.java

```
import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.net.UnknownHostException;


import javax.swing.JOptionPane;


/**
```

```
 *
 * @author Neha Mahajan
 *
 */
public class MessageSendRecieve extends Thread{
        // Contains reference to socket object
        private Socket echoSocket;

        // Contains reference to output stream of socket
        private PrintWriter messageOut;

        // Contains reference to input stream of socket
  private BufferedReader messageIn;
  private String username;

  // Contains the reference to the object of WPIChatClient class.
  private WPIChatClient parent;
  private Boolean disconnect = false;

  /**
        * Constructor.
        * @param parent contains reference to WPIChatClient object.
        *                      username contains the username selected by the user.
        * @return none
        */
        public MessageSendRecieve(String username, WPIChatClient parent){
```

```
                this.username = username;
                this.parent = parent;
                this.createConnection();
        }

        /**
         * Creates the socket, output and input stream for the socket.
         * Stores the reference of the socket object.
         * Stores the reference to output/input streams.
         * @param none.
         * @return none.
         * @exception UnknownHostException, IOException.
         */
        private void createConnection(){
                try{
                        this.echoSocket = new Socket("localhost", 9001);
                        this.messageOut = new PrintWriter(this.echoSocket.getOutputStream(),
true);
                        this.messageIn = new BufferedReader(new
InputStreamReader(this.echoSocket.getInputStream()));

                        sendName(this.username);

                        this.start();
                }catch (UnknownHostException e) {
        System.err.println("Don't know about host ");
        JOptionPane.showMessageDialog(null, StringLiterals.UNABLE_TO_CONNECT);
```

```
        System.exit(1);
    } catch (IOException e) {
        System.err.println("Couldn't get I/O for the connection to ");
        JOptionPane.showMessageDialog(null, StringLiterals.UNABLE_TO_CONNECT);
        System.exit(1);
    }
        }


        /**
         * Overrides the run method of thread class.
         * Handles messages received in socket input stream.
         * Calls functions like display message, show error message and display online users list.
         * Handles disconnect functionality.
         * @param none
         * @return none
         */
        public void run(){
                try {
                        while(true){

                                String input = this.messageIn.readLine();
                                if(input == null){
                                        this.disconnect = true;
                                        break;
                                }
                                String[] messageArray = input.split(":");
```

```
                                switch(messageArray[0]){
                                        case StringLiterals.USER_NOT_ACCEPTED:
                                                this.parent.showErrorMessage();
                                        break;

                                        case StringLiterals.FROM:
                                                this.parent.displayMessageToUser(input);
                                        break;

                                        default:
                                                this.parent.displayOnlineUserUI(input,
this.username);
                                }
                        }

                        if(this.disconnect)
                                this.disconnect();

                } catch (IOException e) {
                        JOptionPane.showMessageDialog(null,
StringLiterals.UNABLE_TO_CONNECT);
        System.exit(1);
                }
          }

          /**
```

```
 * Close the socket connection for the user.
 * Display the login page.
 * @param none.
 * @return none.
 */
private void disconnect() throws IOException{
        this.echoSocket.close();
        this.parent.showLoginPage();
}

/**
 * Send username to server entered by the user.
 * @param username contains the username(String) entered by the user.
 * @return none.
 */
public void sendName(String username){
        String messageToBeSent = StringLiterals.USERNAME + ":" + username;
        this.username = username;
        sendMessage(messageToBeSent);
}

/**
 * Writes the message to output stream.
 * @param message String.
 * @return none.
 */
```

```
	public void sendMessage(String message){
		this.messageOut.println(message);
		this.messageOut.flush();
	}
}
```

6.3.10 OnlineUsersListUI.java

```
import javax.swing.ImageIcon;

/**
 *
 * @author Neha Mahajan
 *
 */
public class OnlineUsersListUI {

	// Contains the references for all chat windows opened.
	private static ArrayList<ChatWindow> ChatWindows = new
ArrayList<ChatWindow>();

	private JFrame frame;
	private JScrollPane scrollPane;
	private JPanel scrollPanel;
	private WPIChatClient parent;
	private JLabel usernameText, noUserOnlineLabel;
	private String username;

	// Contains the reference to broadcast chat window.
	private ChatWindow sendToAll;
	private int onlineUserCount = 0;
	private JLabel listUI;
	private int pX, pY;

	/**
	 * Constructor.
	 * @param  parent contains reference to WPIChatClient class.
	 * @return none
	 */
	public OnlineUsersListUI(WPIChatClient parent) {
		this.parent = parent;
		initialize();
	}

	/**
	 * Creates the JFrame and adds all UI components to it.
	 * Contains the functionality for close, minimize and logout button.
	 * @param  none.
	 * @return none
	 */
	private void initialize() {
		frame = new JFrame();
```

```
		frame.addWindowListener(new WindowAdapter() {
			@Override
			public void windowClosing(WindowEvent arg0) {
				try {
					disconnectFromServer();
					System.exit(0);
				} catch (IOException e) {
					e.printStackTrace();
				}
			}
		});
		frame.setBounds(550, 220, 258, 374);
		frame.setUndecorated(true);
		frame.setShape(new RoundRectangle2D.Double(0, 0,
frame.getWidth(), frame.getHeight(), 10, 10));
		frame.getContentPane().setLayout(null);

		usernameText = new JLabel();
		usernameText.setFont(new Font("Segoe UI Semibold", Font.BOLD,
14));
		usernameText.setForeground(new Color(162, 10, 35));
		usernameText.setBounds(10, 55, 180, 23);
		frame.getContentPane().add(usernameText);

		noUserOnlineLabel = new
JLabel(StringLiterals.NO_USER_ONLINE_LABEL);
		noUserOnlineLabel.setBounds(85, 100, 245, 260);
		noUserOnlineLabel.setVisible(false);
		frame.getContentPane().add(noUserOnlineLabel);

		JButton minimizeBtn = new JButton();
		minimizeBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				frame.setState(Frame.ICONIFIED);
			}
		});
		minimizeBtn.setBounds(205, 5, 20, 20);
		minimizeBtn.setBorder(null);
		minimizeBtn.setBorderPainted(false);
		minimizeBtn.setOpaque(false);
		minimizeBtn.setContentAreaFilled(false);
		ImageIcon minimizeImage = new ImageIcon("images\\minimize.png");
		minimizeBtn.setIcon(minimizeImage);
		frame.getContentPane().add(minimizeBtn);

		JButton logoutBtn = new JButton();
		logoutBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				try {
					disconnectFromServer();
				} catch (IOException e) {
					e.printStackTrace();
				}
			}
		});
		logoutBtn.setBounds(225, 5, 20, 20);
		logoutBtn.setBorder(null);
```

```
		logoutBtn.setBorderPainted(false);
		logoutBtn.setOpaque(false);
		logoutBtn.setContentAreaFilled(false);
		ImageIcon logoutImage = new ImageIcon("images\\logout.png");
		logoutBtn.setIcon(logoutImage);
		frame.getContentPane().add(logoutBtn);

		JPanel panel = new JPanel();
		panel.addMouseMotionListener(new MouseMotionAdapter() {
			@Override
			public void mouseDragged(MouseEvent mouseEvent) {

	frame.setLocation(frame.getLocation().x+mouseEvent.getX()-
pX,frame.getLocation().y+mouseEvent.getY()-pY);
			}
		});
		panel.addMouseListener(new MouseAdapter() {
			@Override
			public void mousePressed(MouseEvent mouseEvent) {
				pX=mouseEvent.getX();
			    pY=mouseEvent.getY();
			}
		});
		panel.setBounds(0, 0, 200, 45);
		panel.setOpaque(false);

		frame.getContentPane().add(panel);

		scrollPanel = new JPanel();
		scrollPane = new JScrollPane(scrollPanel);
		scrollPane.setBounds(5, 100, 245, 260);
		scrollPane.setBorder(null);
		frame.getContentPane().add(scrollPane);

		JButton btnSendToAll = new JButton();
		btnSendToAll.setBounds(200, 55, 34, 25);
		btnSendToAll.setBorder(null);
		btnSendToAll.setBorderPainted(false);
		btnSendToAll.setOpaque(false);
		btnSendToAll.setContentAreaFilled(false);
		ImageIcon broadcastImage = new
ImageIcon("images\\broadcast_icon.png");
		btnSendToAll.setIcon(broadcastImage);
		btnSendToAll.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent ae) {
				if(onlineUserCount == 0){
					JOptionPane.showMessageDialog(null,
StringLiterals.NO_USER_ONLINE);
				} else {
					createSendAllWindow();
				}
			}
		});
		frame.getContentPane().add(btnSendToAll);

		listUI = new JLabel();
```

```
            ImageIcon listUIbgImage = new
ImageIcon("images\\online_user_bg.png");
            listUI.setIcon(listUIbgImage);
            listUI.setBounds(0, 0, 258, 374);
            frame.getContentPane().add(listUI);

            frame.setVisible(true);
      }

      /**
       * Creates an instance of broadcast chat window and stores it
reference.
       * If chat window already exists then just display the chat window.
       * @param  none.
       * @return none
       */
      private void createSendAllWindow(){
            if(sendToAll == null){
                  sendToAll = new ChatWindow(StringLiterals.SEND_TO_ALL,
this);
            } else{
                  this.sendToAll.showPage();
            }
      }

      /**
       * Updates the UI for displaying online users list.
       * @param  onlineUsers contains list of all online users received from
server.
       *             username contains username of the logged in user.
       * @return none
       */
      public void updateOnlineUsersList(String onlineUsers, String username)
{
            String nameText = StringLiterals.LOGGED_IN_AS + username;
            usernameText.setText(nameText);
            this.username = username;
            String[] users = onlineUsers.split(":");

            if(this.scrollPanel.getComponentCount() != 0){
                  this.scrollPanel.removeAll();
            }

            onlineUserCount = 0;
            for(int i = 1; i < users.length; i++){
                  if(!(users[i].equals(username))){
                        onlineUserCount++;
                        OnlineUserCustomButton btnUsers = new
OnlineUserCustomButton(users[i], 0, (onlineUserCount-1) * 36);
                        btnUsers.addMouseListener(new MouseAdapter(){
                              public void mouseClicked(MouseEvent
mouseEvent){
                                    openChatWindow(((OnlineUserCustomButton)
mouseEvent.getSource()).getUsername());
                              }
                        });
                        this.scrollPanel.add(btnUsers);
```

```
                }
            }
            this.scrollPanel.setPreferredSize(new
Dimension(225,onlineUserCount * 36 + (onlineUserCount - 1)* 11));
            this.scrollPanel.revalidate();
            this.scrollPane.revalidate();
            this.scrollPane.repaint();

            if(onlineUserCount == 0){
                  noUserOnlineLabel.setVisible(true);
            }
            else {
                  noUserOnlineLabel.setVisible(false);
            }

            showUserLoggedOff(users);
            showUserLoggedIn(users);
      }

      /**
       * Show user as logged in if its chat window is open.
       * If online users are greater than 1 then enable the broadcast chat
window else disable it.
       * @param  users contains list of all online users.
       * @return none
       */
      private void showUserLoggedIn(String[] users) {
            for(int j = 0; j < ChatWindows.size(); j++){

      if(Arrays.asList(users).contains(ChatWindows.get(j).getUsername())){
                        ChatWindows.get(j).showLoggedInAgain();
                  }
            }

            if(sendToAll != null){
                  if(onlineUserCount == 0){
                        sendToAll.disableSendForBroadcast();
                  } else {
                        sendToAll.enableSendForBroadcast();
                  }
            }
      }

      /**
       * Show user as logged off if its chat window is open.
       * If online users are greater than 1 then enable the broadcast chat
window else disable it.
       * @param  users contains list of all online users.
       * @return none
       */
      private void showUserLoggedOff(String[] users) {
            for(int j = 0; j < ChatWindows.size(); j++){

      if(!Arrays.asList(users).contains(ChatWindows.get(j).getUsername())){
                        ChatWindows.get(j).showLoggedOutMessage();
                  }
            }
```

```
			if(sendToAll != null){
				if(onlineUserCount == 0){
					sendToAll.disableSendForBroadcast();
				} else {
					sendToAll.enableSendForBroadcast();
				}
			}
		}

		/**
		 * Create instances of chat window for each user selected in the online
users list and store there references in ArrayList.
		 * If chat window already exists then just display the chat window.
		 * @param  user contains name of the recipient user.
		 * @return none
		 */
		private void openChatWindow(String user){

			boolean userNotFound = true;
			for(int i = 0; i < ChatWindows.size(); i++){
				ChatWindow tempUserChatWindow = ChatWindows.get(i);
				if(tempUserChatWindow.getUsername().equals(user)){
					tempUserChatWindow.showPage();
					userNotFound  = false;
					break;
				}
			}
			if(userNotFound == true){
				ChatWindow chatWindow = new ChatWindow(user, this);
				ChatWindows.add(chatWindow);
			}
		}

		/**
		 * Display the message in the appropriate chat window depending
		 * upon the sender name received in the message.
		 * If the sender is the user itself then display message in chat window
for the recipient name.
		 * If message is broadcast, then display message in broadcast chat
window.
		 * @param  input contains message from the server.
		 * @return none
		 */
		public void displayMessageToUser(String input) {
			String[] messageArray = input.split(":");
			ChatWindow tempUserChatWindow;
			Boolean userNotFound = true;

			if(messageArray[3].equals(StringLiterals.SEND_TO_ALL)){
				createSendAllWindow();
				sendToAll.displayMessage(input);
			} else {
				if(messageArray[1].equals(this.username)){
					for(int i = 0; i < ChatWindows.size(); i++){
						tempUserChatWindow = ChatWindows.get(i);
```

```
		if(tempUserChatWindow.getUsername().equals(messageArray[3])){
					tempUserChatWindow.displayMessage(input);
					break;
				}
			}
		} else {
			for(int i = 0; i < ChatWindows.size(); i++){
				tempUserChatWindow = ChatWindows.get(i);

		if(tempUserChatWindow.getUsername().equals(messageArray[1])){
					tempUserChatWindow.displayMessage(input);
					userNotFound = false;
					break;
				}
			}
			if(userNotFound == true){
				ChatWindow chatWindow = new
ChatWindow(messageArray[1], this);
				ChatWindows.add(chatWindow);
				chatWindow.displayMessage(input);
			}
		}
	}
	}

	public void sendMessage(String message){
		String tempMessage = StringLiterals.FROM + ":" + this.username +
":" + message;
		this.parent.sendMessage(tempMessage);
	}

	/**
	 * Returns the username of the user.
	 * @param  none.
	 * @return username
	 */
	public String getUsername(){
		return this.username;
	}

	/**
	 * Remove all the references of chat windows including broadcast chat
window.
	 * Clear the ArrayList of chat windows.
	 * This function is called when client clicks on log out/disconnect
button.
	 * @param  none.
	 * @return none.
	 * @exception IOException
	 */
	private void disconnectFromServer() throws IOException {
		for(int i = 0; i < ChatWindows.size(); i++){
			ChatWindow tempUserChatWindow = ChatWindows.get(i);
			tempUserChatWindow.hidePage();
			tempUserChatWindow = null;
		}
```

```
                ChatWindows.clear();

                if(sendToAll != null){
                        sendToAll.hidePage();
                        sendToAll = null;
                }

                this.parent.disconnectFromServer();
        }

        /**
         * Hides the JFrame.
         * @param  none.
         * @return none.
         */
        public void hidePage(){
                this.frame.setVisible(false);
        }
}
```

6.3.11 OnlineUserCustomButton.java

```
import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;


import javax.swing.ImageIcon;

import javax.swing.JLabel;

import javax.swing.JPanel;




/**

*

* @author Neha Mahajan

*

*/
```

```
public class OnlineUserCustomButton extends JPanel{
	private static final long serialVersionUID = 1L;
	private JLabel usernameText;

	/**
	 * Constructor.
	 * Creates a custom UI button for a online user to be displayed in OnlineUsersListUI page.
	 * @param username contains the username(String) for a online user.
	 * @return none
	 */
	public OnlineUserCustomButton(String username, int x, int y){
		this.setLayout(null);
		this.setPreferredSize(new Dimension(225,36));
		this.setLocation(x,y);

		usernameText = new JLabel(username);
		usernameText.setFont(new Font("Segoe UI Semibold", Font.BOLD, 14));
		usernameText.setForeground(new Color(162, 10, 35));
		usernameText.setBounds(10, 0, 190, 36);
		this.add(usernameText);

		JLabel onlineIcon = new JLabel();
		onlineIcon.setBounds(190, 10, 17, 17);
		ImageIcon onlineIconImage = new ImageIcon("images\\online.png");
		onlineIcon.setIcon(onlineIconImage);
		this.add(onlineIcon);
```

```
		JLabel bgIcon = new JLabel();
		bgIcon.setBounds(0, 0, 225, 36);
		ImageIcon bgIconImage = new ImageIcon("images\\online_button_bg.png");
		bgIcon.setIcon(bgIconImage);
		this.add(bgIcon);
	}

	/**
	 * Returns the username associated with the button.
	 * @param none.
	 * @return username(String)
	 */
	public String getUsername(){
		return this.usernameText.getText();
	}
}
```

6.3.12 ChatWindow.java

```
import javax.swing.ImageIcon;

/**
 *
 * @author Neha Mahajan
 *
 */
public class ChatWindow {

	private JFrame frame;
	private TextArea inputChat,outputChat;
	private String recieverUserName = "";

	// Contains the reference to object of OnlineUsersListUI class.
	private OnlineUsersListUI parent;
```

```
	private int pX, pY;
	private JButton btnSend;
	private Boolean isOnline = true;

	/**
	 * Constructor.
	 * @param parent contains reference to OnlineUsersListUI object.
	 *            user contains the username(String) of the recipient.
	 * @return none
	 */
	public ChatWindow(String user, OnlineUsersListUI parent) {
		this.parent = parent;
		this.recieverUserName = user;
		initialize(user);
	}

	/**
	 * Creates the JFrame and adds all UI components to it.
	 * Contains function calls for close and minimize button.
	 * Drag functionality for JFrame is implemented in the function.
	 * @param user contains the username(String) of the recipient.
	 * @return none
	 */
	private void initialize(String user) {
		frame = new JFrame();
		frame.setBounds(550, 220, 258, 374);
		frame.setUndecorated(true);
		frame.setShape(new RoundRectangle2D.Double(0, 0,
frame.getWidth(), frame.getHeight(), 10, 10));
		frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
		frame.getContentPane().setLayout(null);

		JButton minimizeBtn = new JButton();
		minimizeBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				frame.setState(Frame.ICONIFIED);
			}
		});
		minimizeBtn.setBounds(205, 5, 20, 20);
		minimizeBtn.setBorder(null);
		minimizeBtn.setBorderPainted(false);
		minimizeBtn.setOpaque(false);
		minimizeBtn.setContentAreaFilled(false);
		ImageIcon minimizeImage = new ImageIcon("images\\minimize.png");
		minimizeBtn.setIcon(minimizeImage);
		frame.getContentPane().add(minimizeBtn);

		JButton closeBtn = new JButton();
		closeBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				hidePage();
			}
		});
		closeBtn.setBounds(233, 8, 11, 12);
		closeBtn.setBorder(null);
		closeBtn.setBorderPainted(false);
		closeBtn.setOpaque(false);
```

```
            closeBtn.setContentAreaFilled(false);
            ImageIcon closeImage = new ImageIcon("images\\close.png");
            closeBtn.setIcon(closeImage);
            frame.getContentPane().add(closeBtn);

            JPanel panel = new JPanel();
            panel.addMouseMotionListener(new MouseMotionAdapter() {
                  @Override
                  public void mouseDragged(MouseEvent mouseEvent) {

      frame.setLocation(frame.getLocation().x+mouseEvent.getX()-
pX,frame.getLocation().y+mouseEvent.getY()-pY);
                  }
            });
            panel.addMouseListener(new MouseAdapter() {
                  @Override
                  public void mousePressed(MouseEvent mouseEvent) {
                        pX=mouseEvent.getX();
                    pY=mouseEvent.getY();
                  }
            });
            panel.setBounds(0, 0, 200, 40);
            panel.setOpaque(false);

            frame.getContentPane().add(panel);

            outputChat = new TextArea("",2,50,
TextArea.SCROLLBARS_VERTICAL_ONLY);
            outputChat.setEditable(false);
            outputChat.setBounds(10, 85, 234, 223);
            frame.getContentPane().add(outputChat);

            JLabel lblTo = new JLabel((StringLiterals.TO).toUpperCase() + " :
" + user);
            lblTo.setForeground(new Color(162, 10, 35));
            lblTo.setFont(new Font("Segoe UI Semibold", Font.BOLD, 14));
            lblTo.setBounds(10, 55, 234, 14);
            frame.getContentPane().add(lblTo);

            btnSend = new JButton(StringLiterals.SEND);
            btnSend.setBackground(new Color(162, 10, 35));
            btnSend.setForeground(new Color(255, 255, 255));
            btnSend.setBorder(null);
            btnSend.setFocusPainted(false);
            btnSend.setBorderPainted(false);
            btnSend.setFont(new Font("Segoe UI Symbol", Font.PLAIN, 11));
            btnSend.setBounds(184, 326, 60, 29);
            btnSend.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        String input = inputChat.getText();
                        if(input.equals("")){
                              JOptionPane.showMessageDialog(null,
StringLiterals.MESSAGE_EMPTY);
                        } else {
                              String message = StringLiterals.TO + ":" +
recieverUserName + ":";
                              inputChat.setText("");
```

```
                                input =
input.replaceAll(System.getProperty("line.separator"), " ");
                                message += input;
                                parent.sendMessage(message);
                          }
                    }
              });
              frame.getContentPane().add(btnSend);

              inputChat = new TextArea("",2,50,
TextArea.SCROLLBARS_VERTICAL_ONLY);
              inputChat.setBounds(10, 319, 160, 44);
              frame.getContentPane().add(inputChat);

              JLabel listUI = new JLabel();
              ImageIcon listUIbgImage = new
ImageIcon("images\\online_user_bg.png");
              listUI.setIcon(listUIbgImage);
              listUI.setBounds(0, 0, 258, 374);
              frame.getContentPane().add(listUI);

              frame.setVisible(true);
        }

        /**
         * Returns the recipient username.
         * @param none.
         * @return recieverUserName
         */
        public String getUsername(){
              return this.recieverUserName;
        }

        /**
         * Displays the message in the output text area of chat window.
         * @param input contains the input message(String) from the sender.
         * @return none.
         */
        public void displayMessage(String input) {
              String[] messageArray = input.split(":");
              if(messageArray[1].equals(parent.getUsername())){
                    String messageString = StringLiterals.YOU + " : " +
messageArray[4];
                    outputChat.append("\n"+messageString);
              }else {
                    String messageString = messageArray[1].toUpperCase() + "
" + messageArray[4];
                    outputChat.append("\n"+messageString);
              }
              this.inputChat.requestFocus();
        }

        /**
         * Displays message that recipient is online in message window if
recipient logs in again.
         * Enables the send button and output text.
         * @param none.
```

```
	 * @return none.
	 */
	public void showLoggedInAgain(){
		if(this.isOnline == false){
			this.isOnline = true;
			String messageString = this.recieverUserName + " : is
online";
			outputChat.append("\n"+messageString);
			this.btnSend.setEnabled(true);
			this.inputChat.setEditable(true);
			this.inputChat.setEnabled(true);
		}
	}

	/**
	 * Displays message that recipient is offline in message window if
recipient logs off again.
	 * Disables the send button and output text.
	 * @param none.
	 * @return none.
	 */
	public void showLoggedOutMessage() {
		if(this.isOnline == true){
			this.isOnline = false;
			String messageString = this.recieverUserName + " : is
offline";
			outputChat.append("\n"+messageString);
			this.btnSend.setEnabled(false);
			this.inputChat.setEditable(false);
			this.inputChat.setEnabled(false);
		}
	}

	/**
	 * Hides the JFrame of the chat window.
	 * @param none.
	 * @return none.
	 */
	public void hidePage() {
		this.frame.setVisible(false);
	}

	/**
	 * Displays the JFrame of the chat window.
	 * @param none.
	 * @return none.
	 */
	public void showPage() {
		this.frame.setVisible(true);
	}

	/**
	 * Disable send button when no user is online for broadcast window.
	 * Input chat area is not editable.
	 * @param none.
	 * @return none.
	 */
```

```
	public void disableSendForBroadcast() {
		this.btnSend.setEnabled(false);
		this.inputChat.setEditable(false);
		this.inputChat.setEnabled(false);
	}

	/**
	 * Enables send button when user is online for broadcast window.
	 * Input chat area is editable.
	 * @param none.
	 * @return none.
	 */
	public void enableSendForBroadcast() {
		this.btnSend.setEnabled(true);
		this.inputChat.setEditable(true);
		this.inputChat.setEnabled(true);
	}
}
```