```
 2  * To change this license header, choose License Headers in Project Properties.
 6 package chatclient;
 7
 8 import com.nxkundu.client.service.ClientService;
 9 import com.nxkundu.server.bo.Client;
10 import com.nxkundu.server.bo.DataPacket;
11 import java.awt.event.MouseAdapter;
12 import java.awt.event.MouseEvent;
13 import java.awt.image.BufferedImage;
14 import java.io.ByteArrayInputStream;
15 import java.io.FileFilter;
16 import java.io.IOException;
17 import java.util.HashMap;
18 import java.util.Map;
19 import javax.swing.DefaultListModel;
20 import java.util.concurrent.ConcurrentLinkedQueue;
21 import java.util.concurrent.ConcurrentMap;
22 import javax.imageio.ImageIO;
23 import javax.swing.Icon;
24 import javax.swing.ImageIcon;
25 import javax.swing.JFileChooser;
26 import javax.swing.filechooser.FileNameExtensionFilter;
27
28 /**
29  *
30  * @author nxkundu
31  *
32  * @email nxk161830@utdallas.edu
33  * @name Nirmallya Kundu
34  *
35  * CHAT SCREEN UI
36  *
37  * This class contains the main chat screen
38  * After the user logged in
39  * This page is showed to the user
40  * with ONLINE OFFLINE clients
41  * and the chat history of the user
```

```java
 * Also the user can logout from this screen
 */
public class ChatScreen extends javax.swing.JFrame implements Runnable{

    /**
     * Creates new form ChatScreent
     */
    private Client client;
    private ClientService clientService;

    private Thread threadClientChatScreen;
    private Thread threadUpdateFriendList;
    private Thread threadSendReceiveDataPacket;
    private Thread threadSendReceiveAllDataPacket;

    private Map<String, DefaultListModel<DisplayData>> mapModelChatHistory;
    DefaultListModel<String> modelOnlineFriends = null;
    DefaultListModel<String> modelOfflineFriends = null;

    /***************************** Constructors *****************************/

    /**
     * Constructor
     * When the chat Screen is called
     * it shows the
     *
     * 1> List of ONLINE Clients
     * 2> List of OFFLINE Clients
     * 3> The Chat History of all the client
     *
     * @param client
     * @param clientService
     */
    public ChatScreen(Client client, ClientService clientService) {
        initComponents();
        this.client = client;
        this.clientService = clientService;
```

```
79          this.txtMessage.grabFocus();
80          this.LabelUser.setText(client.getUserName());
81
82          modelOnlineFriends = new DefaultListModel<>();
83          modelOnlineFriends.addElement(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE);
84          modelOfflineFriends = new DefaultListModel<>();
85
86          listOnlineFriends.setModel(modelOnlineFriends);
87          listOfflineFriends.setModel(modelOfflineFriends);
88
89
90          listOnlineFriends.addMouseListener(new MouseAdapter() {
91              public void mouseClicked(MouseEvent e) {
92                  if (e.getClickCount() == 2) {
93
94                      listOfflineFriends.setSelectedIndex(-1);
95
96                      String chatFriend = listOnlineFriends.getSelectedValue();
97                      labelChatFriend.setText(chatFriend);
98                      labelChatFriendStatus.setText("Online");
99
100                     listChatHistory.setCellRenderer(new JListRendered());
101                     listChatHistory.setModel(mapModelChatHistory.get(chatFriend));
102                 }
103             }
104         });
105
106         listOfflineFriends.addMouseListener(new MouseAdapter() {
107             public void mouseClicked(MouseEvent e) {
108                 if (e.getClickCount() == 2) {
109
110                     listOnlineFriends.setSelectedIndex(-1);
111
112                     String chatFriend = listOfflineFriends.getSelectedValue();
113                     labelChatFriend.setText(chatFriend);
114
115                     labelChatFriendStatus.setText((clientService.getMapAllClients().get(chatFriend)).lastSeen());
```

```
116
117                     listChatHistory.setCellRenderer(new JListRendered());
118                     listChatHistory.setModel(mapModelChatHistory.get(chatFriend));
119                 }
120             }
121         });
122
123
124         mapModelChatHistory = new HashMap<>();
125         DefaultListModel<DisplayData> modelBroadCastMessage = new DefaultListModel<>();
126
127         listChatHistory.setCellRenderer(new JListRendered());
128         listChatHistory.setModel(modelBroadCastMessage);
129
130         mapModelChatHistory.put(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE, modelBroadCastMessage);
131
132         threadClientChatScreen = new Thread(this, "ClientChatScreenStart");
133         threadClientChatScreen.start();
134
135     }
136
137     @Override
138     public void run() {
139
140         /**
141         * updateFriendListThread() - this method starts the thread threadUpdateFriendList
142         * which updates the Clients as ONLINE/OFFLINE
143         * based on the received From the server
144         */
145         updateFriendListThread();
146
147         /**
148         * receiveChatMessage() - this method starts the thread threadSendReceiveDataPacket
149         * which receives the messages from the server of a particular
150         * client whose chat is current open
151         * and displays it on the respective User Client Chat Screen
152         */
```

```
153         receiveChatMessage();
154 
155         /**
156         * receiveAllChatMessage() - this method starts the thread threadSendReceiveDataPacket
157         * which receives all the messages from the server of a all the clients
158         * EXCEPT the client whose chat is current open
159         * and buffers it to display it on the client as soon as
160         * the client open that user client chat screen
161         */
162         receiveAllChatMessage();
163     }
164 
165     /**
166      * This method is used when the
167      * Client send a Personal/Broadcast
168      * Message to the Other Clients
169      */
170     private void sendMessage() {
171 
172         String message = txtMessage.getText();
173         String toClient = labelChatFriend.getText();
174 
175         if(message.length() == 0) {
176             return;
177         }
178 
179         mapModelChatHistory.get(toClient).addElement(new DisplayData("Me: " + message));
180         txtMessage.setText("");
181 
182         if(toClient.equals(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE)) {
183 
184             message = client.getUserName() +" [BroadcastMessage]: " + message;
185             clientService.sendPacket(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE, message, "");
186         }
187         else {
188 
189             message = client.getUserName() +": " + message;
```

```
190             clientService.sendPacket(DataPacket.MESSAGE_TYPE_MESSAGE, message, toClient);
191         }
192 
193         listChatHistory.ensureIndexIsVisible(mapModelChatHistory.get(toClient).getSize());
194     }
195 
196     /**
197      * This method is used when the
198      * Client send a Personal/Broadcast
199      * Multimedia Message (Image File) to the Other Clients
200      */
201     private void sendImage() {
202 
203         String message = txtMessage.getText();
204         String toClient = labelChatFriend.getText();
205         JFileChooser fileChooser = new JFileChooser();
206 
207         fileChooser.addChoosableFileFilter(new FileNameExtensionFilter("Image Files", "jpg", "png", "tif"));
208 
209         if (fileChooser.showOpenDialog(rootPane) == JFileChooser.APPROVE_OPTION) {
210 
211             java.io.File file = fileChooser.getSelectedFile();
212 
213             if(toClient.equals(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE)) {
214 
215 //                clientService.sendPacket(DataPacket.MESSAGE_TYPE_BROADCAST_IMAGE, file.getPath(), toClient);
216 //
217 //                mapModelChatHistory.get(toClient).addElement(new DisplayData("Me: ", new ImageIcon(file.getPath())));
218             }
219             else {
220 
221                 clientService.sendPacket(DataPacket.MESSAGE_TYPE_IMAGE_MESSAGE, file.getPath(), toClient);
222 
223                 mapModelChatHistory.get(toClient).addElement(new DisplayData("Me: ", new ImageIcon(file.getPath())));
224 
225             }
226             listChatHistory.ensureIndexIsVisible(mapModelChatHistory.get(toClient).getSize());
```

```
227
228         }
229     }
230
231     /**
232      * updateFriendListThread() - this method starts the thread threadUpdateFriendList
233      * which updates the Clients as ONLINE/OFFLINE
234      * based on the received From the server
235      */
236     private void updateFriendListThread() {
237
238         threadUpdateFriendList = new Thread("UpdateFriendList"){
239             @Override
240             public void run() {
241
242         while(ClientService.isLoggedIn) {
243
244                     updateFriendList();
245
246                     try {
247
248                             Thread.sleep(6000);
249                     }
250                     catch(Exception e) {
251
252                             e.printStackTrace();
253                     }
254
255         }
256             }
257     };
258
259     threadUpdateFriendList.start();
260     }
261
262     /**
263      * updateFriendList() - this method updates the Clients as ONLINE/OFFLINE
```

```
264      * based on the received From the server
265      */
266     private void updateFriendList() {
267 
268         ConcurrentMap<String, Client> mapAllClients = clientService.getMapAllClients();
269 
270         if(mapAllClients != null && !mapAllClients.isEmpty()) {
271 
272             mapAllClients.remove(client.getUserName());
273 
274             for(String userName : mapAllClients.keySet()) {
275 
276                 if(mapAllClients.get(userName).isOnline()) {
277 
278                     modelOfflineFriends.removeElement(userName);
279 
280                     if(!modelOnlineFriends.contains(userName)) {
281 
282                         modelOnlineFriends.addElement(userName);
283                     }
284                 }
285 
286                 if(!mapAllClients.get(userName).isOnline()) {
287 
288                     modelOnlineFriends.removeElement(userName);
289 
290                     if(!modelOfflineFriends.contains(userName)) {
291 
292                         modelOfflineFriends.addElement(userName);
293                     }
294                 }
295 
296                 if(!mapModelChatHistory.containsKey(userName)) {
297                     DefaultListModel<DisplayData> model = new DefaultListModel<>();
298                     mapModelChatHistory.put(userName, model);
299                 }
300             }
```

```
301
302             listOnlineFriends.setModel(modelOnlineFriends);
303             listOfflineFriends.setModel(modelOfflineFriends);
304         }
305
306         if(modelOnlineFriends.contains(labelChatFriend.getText())){
307             listOnlineFriends.setSelectedValue(labelChatFriend.getText(), true);
308         }
309         else {
310             listOnlineFriends.setSelectedIndex(-1);
311         }
312
313         if(modelOfflineFriends.contains(labelChatFriend.getText())) {
314             listOfflineFriends.setSelectedValue(labelChatFriend.getText(), true);
315         }
316         else {
317             listOfflineFriends.setSelectedIndex(-1);
318         }
319     }
320
321     /**
322      * receiveChatMessage() - this method starts the thread threadSendReceiveDataPacket
323      * which receives the messages from the server of a particular
324      * client whose chat is current open
325      * and displays it on the respective User Client Chat Screen
326      */
327     private void receiveChatMessage() {
328
329         threadSendReceiveDataPacket = new Thread("SendReceiveDataPacket"){
330             @Override
331             public void run() {
332
333                 while(ClientService.isLoggedIn) {
334
335                     String fromClient = labelChatFriend.getText();
336
337                     if(!fromClient.equals(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE)) {
```

```
338
339                     Map<String, ConcurrentLinkedQueue<DataPacket>> mapClientReceivedDataPacket = null;
340                     mapClientReceivedDataPacket = clientService.getMapClientReceivedDataPacket();
341
342                     ConcurrentLinkedQueue<DataPacket> qDataPacket = mapClientReceivedDataPacket.get(fromClient);
343
344                     if(qDataPacket != null && !qDataPacket.isEmpty()) {
345
346                         DataPacket dataPacket = qDataPacket.poll();
347
348                         DefaultListModel<DisplayData> modelChatHistory = new DefaultListModel<>();
349                         if(mapModelChatHistory.containsKey(fromClient)) {
350                             modelChatHistory = mapModelChatHistory.get(fromClient);
351                         }
352
353                         switch(dataPacket.getMessageType()) {
354
355                             case DataPacket.MESSAGE_TYPE_MESSAGE:
356                                 modelChatHistory.addElement(new DisplayData(dataPacket.getMessage()));
357                                 break;
358
359                             case DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE:
360                                 modelChatHistory.addElement(new DisplayData(dataPacket.getMessage()));
361                                 break;
362
363                             case DataPacket.MESSAGE_TYPE_IMAGE_MESSAGE:
364
365                                 BufferedImage bufferedImage = null;
366                                 try {
367
368                                     bufferedImage = ImageIO.read(new ByteArrayInputStream(dataPacket.getByteImage()));
369                                 }
370                                 catch (IOException e) {
371
372                                     e.printStackTrace();
373                                 }
374                                 modelChatHistory.addElement(new DisplayData(dataPacket.getFromClient().getUserName(),
```

```
   (dataPacket.getByteImage())));
375                                     break;
376                             }
377
378                             listChatHistory.ensureIndexIsVisible(modelChatHistory.getSize());
379                         }
380                     }
381
382                     try {
383
384                             Thread.sleep(500);
385                     }
386                     catch(Exception e) {
387
388                             e.printStackTrace();
389                     }
390
391                 }
392             }
393         };
394
395         threadSendReceiveDataPacket.start();
396
397     }
398
399     /**
400      * receiveAllChatMessage() - this method starts the thread threadSendReceiveDataPacket
401      * which receives all the messages from the server of a all the clients
402      * EXCEPT the client whose chat is current open
403      * and buffers it to display it on the client as soon as
404      * the client open that user client chat screen
405      */
406     private void receiveAllChatMessage() {
407
408         threadSendReceiveAllDataPacket = new Thread("SendReceiveAllDataPacket"){
409             @Override
410             public void run() {
```

```
411
412                 while(ClientService.isLoggedIn) {
413
414                     Map<String, ConcurrentLinkedQueue<DataPacket>> mapClientReceivedDataPacket = null;
415                     mapClientReceivedDataPacket = clientService.getMapClientReceivedDataPacket();
416
417                     String exceptClient = labelChatFriend.getText();
418
419                     for(String fromClient : mapClientReceivedDataPacket.keySet()) {
420
421                         if(fromClient.equals(exceptClient)) {
422                             continue;
423                         }
424
425                         ConcurrentLinkedQueue<DataPacket> qDataPacket = mapClientReceivedDataPacket.get(fromClient);
426
427                         if(qDataPacket != null && !qDataPacket.isEmpty()) {
428
429                             DataPacket dataPacket = qDataPacket.poll();
430
431                             DefaultListModel<DisplayData> modelChatHistory = new DefaultListModel<>();
432                             if(mapModelChatHistory.containsKey(fromClient)) {
433                                 modelChatHistory = mapModelChatHistory.get(fromClient);
434                             }
435                             modelChatHistory.addElement(new DisplayData(dataPacket.getMessage()));
436                         }
437                     }
438
439                     try {
440
441                             Thread.sleep(500);
442                     }
443                     catch(Exception e) {
444
445                             e.printStackTrace();
446                     }
447
```

```
448                 }
449             }
450         };
451 
452         threadSendReceiveAllDataPacket.start();
453 
454     }
455 
456 
457     /**
458      * This method is called from within the constructor to initialize the form.
459      * WARNING: Do NOT modify this code. The content of this method is always
460      * regenerated by the Form Editor.
461      */
462     @SuppressWarnings("unchecked")
463     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
464     private void initComponents() {
465 
466         jPanel1 = new javax.swing.JPanel();
467         jLabel1 = new javax.swing.JLabel();
468         buttonLogout = new javax.swing.JButton();
469         LabelUser = new javax.swing.JLabel();
470         jPanel2 = new javax.swing.JPanel();
471         jPanel4 = new javax.swing.JPanel();
472         jLabel3 = new javax.swing.JLabel();
473         jScrollPane2 = new javax.swing.JScrollPane();
474         listOnlineFriends = new javax.swing.JList<>();
475         jPanel5 = new javax.swing.JPanel();
476         jLabel4 = new javax.swing.JLabel();
477         jScrollPane1 = new javax.swing.JScrollPane();
478         listOfflineFriends = new javax.swing.JList<>();
479         jPanel3 = new javax.swing.JPanel();
480         jScrollPane3 = new javax.swing.JScrollPane();
481         listChatHistory = new javax.swing.JList<>();
482         txtMessage = new javax.swing.JTextField();
483         btnSend = new javax.swing.JButton();
484         jLabel5 = new javax.swing.JLabel();
```

```
485         labelChatFriend = new javax.swing.JLabel();
486         labelChatFriendStatus = new javax.swing.JLabel();
487         ButtonImage = new javax.swing.JButton();
488 
489         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
490 
491         jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder(""));
492 
493         jLabel1.setFont(new java.awt.Font("Lucida Grande", 1, 18)); // NOI18N
494         jLabel1.setText("Welcome, ");
495 
496         buttonLogout.setText("Logout");
497         buttonLogout.addMouseListener(new java.awt.event.MouseAdapter() {
498             public void mouseClicked(java.awt.event.MouseEvent evt) {
499                 buttonLogoutMouseClicked(evt);
500             }
501         });
502 
503         LabelUser.setFont(new java.awt.Font("Lucida Grande", 1, 18)); // NOI18N
504         LabelUser.setText("User");
505 
506         javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
507         jPanel1.setLayout(jPanel1Layout);
508         jPanel1Layout.setHorizontalGroup(
509             jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
510             .addGroup(jPanel1Layout.createSequentialGroup()
511                 .addGap(6, 6, 6)
512                 .addComponent(jLabel1)
513                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
514                 .addComponent(LabelUser)
515                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE,
    Short.MAX_VALUE)
516                 .addComponent(buttonLogout)
517                 .addGap(6, 6, 6))
518         );
519         jPanel1Layout.setVerticalGroup(
520             jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
521             .addGroup(jPanel1Layout.createSequentialGroup()
522                 .addGap(6, 6, 6)
523                 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
524                     .addComponent(jLabel1)
525                     .addComponent(buttonLogout, javax.swing.GroupLayout.DEFAULT_SIZE, 40, Short.MAX_VALUE)
526                     .addComponent(LabelUser))
527                 .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
528         );
529 
530         buttonLogout.getAccessibleContext().setAccessibleName("LogoutButton");
531         LabelUser.getAccessibleContext().setAccessibleName("UserLabel");
532 
533         jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder(""));
534 
535         jPanel4.setBorder(javax.swing.BorderFactory.createTitledBorder(""));
536 
537         jLabel3.setFont(new java.awt.Font("Lucida Grande", 1, 14)); // NOI18N
538         jLabel3.setText("Online Friends");
539 
540         listOnlineFriends.setFont(new java.awt.Font("Lucida Grande", 1, 14)); // NOI18N
541         jScrollPane2.setViewportView(listOnlineFriends);
542         listOnlineFriends.getAccessibleContext().setAccessibleName("onlineList");
543 
544         javax.swing.GroupLayout jPanel4Layout = new javax.swing.GroupLayout(jPanel4);
545         jPanel4.setLayout(jPanel4Layout);
546         jPanel4Layout.setHorizontalGroup(
547             jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
548             .addGroup(jPanel4Layout.createSequentialGroup()
549                 .addGap(6, 6, 6)
550                 .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
551                     .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
552                     .addComponent(jScrollPane2))
553                 .addGap(6, 6, 6))
554         );
555         jPanel4Layout.setVerticalGroup(
556             jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
557             .addGroup(jPanel4Layout.createSequentialGroup()
```

```
558                 .addGap(6, 6, 6)
559                 .addComponent(jLabel3)
560                 .addGap(6, 6, 6)
561                 .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE, 172, Short.MAX_VALUE)
562                 .addGap(6, 6, 6))
563         );
564 
565         jPanel5.setBorder(javax.swing.BorderFactory.createTitledBorder(""));
566 
567         jLabel4.setFont(new java.awt.Font("Lucida Grande", 1, 14)); // NOI18N
568         jLabel4.setText("Offline Friends");
569 
570         listOfflineFriends.setFont(new java.awt.Font("Lucida Grande", 1, 14)); // NOI18N
571         jScrollPane1.setViewportView(listOfflineFriends);
572         listOfflineFriends.getAccessibleContext().setAccessibleName("OfflineList");
573 
574         javax.swing.GroupLayout jPanel5Layout = new javax.swing.GroupLayout(jPanel5);
575         jPanel5.setLayout(jPanel5Layout);
576         jPanel5Layout.setHorizontalGroup(
577             jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
578             .addGroup(jPanel5Layout.createSequentialGroup()
579                 .addGap(6, 6, 6)
580                 .addGroup(jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
581                     .addComponent(jScrollPane1)
582                     .addComponent(jLabel4, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
583                 .addGap(6, 6, 6))
584         );
585         jPanel5Layout.setVerticalGroup(
586             jPanel5Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
587             .addGroup(jPanel5Layout.createSequentialGroup()
588                 .addGap(6, 6, 6)
589                 .addComponent(jLabel4)
590                 .addGap(6, 6, 6)
591                 .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 169, javax.swing.GroupLayout.PREFERRED_SIZE)
592                 .addGap(6, 6, 6))
593         );
594 
```

```
595         javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
596         jPanel2.setLayout(jPanel2Layout);
597         jPanel2Layout.setHorizontalGroup(
598             jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
599             .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel2Layout.createSequentialGroup()
600                 .addGap(6, 6, 6)
601                 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
602                     .addComponent(jPanel5, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
603                     .addComponent(jPanel4, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
604                 .addGap(6, 6, 6))
605         );
606         jPanel2Layout.setVerticalGroup(
607             jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
608             .addGroup(jPanel2Layout.createSequentialGroup()
609                 .addGap(6, 6, 6)
610                 .addComponent(jPanel4, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
611                 .addGap(6, 6, 6)
612                 .addComponent(jPanel5, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
613                 .addContainerGap())
614         );
615 
616         jPanel3.setBorder(javax.swing.BorderFactory.createTitledBorder(""));
617 
618         jScrollPane3.setViewportView(listChatHistory);
619         listChatHistory.getAccessibleContext().setAccessibleName("ChatList");
620 
621         txtMessage.addKeyListener(new java.awt.event.KeyAdapter() {
622             public void keyPressed(java.awt.event.KeyEvent evt) {
623                 txtMessageKeyPressed(evt);
624             }
625         });
626 
627         btnSend.setText("Send");
628         btnSend.addMouseListener(new java.awt.event.MouseAdapter() {
629             public void mouseClicked(java.awt.event.MouseEvent evt) {
630                 btnSendMouseClicked(evt);
```

```
631             }
632         });
633 
634         jLabel5.setFont(new java.awt.Font("Lucida Grande", 1, 14)); // NOI18N
635         jLabel5.setText("Chat With, ");
636 
637         labelChatFriend.setFont(new java.awt.Font("Lucida Grande", 1, 14)); // NOI18N
638         labelChatFriend.setText("BROADCAST_MESSAGE");
639 
640         labelChatFriendStatus.setText("Online");
641 
642         ButtonImage.setText("Image");
643         ButtonImage.setMaximumSize(new java.awt.Dimension(70, 29));
644         ButtonImage.setMinimumSize(new java.awt.Dimension(70, 29));
645         ButtonImage.setPreferredSize(new java.awt.Dimension(75, 29));
646         ButtonImage.addMouseListener(new java.awt.event.MouseAdapter() {
647             public void mouseClicked(java.awt.event.MouseEvent evt) {
648                 ButtonImageMouseClicked(evt);
649             }
650         });
651 
652         javax.swing.GroupLayout jPanel3Layout = new javax.swing.GroupLayout(jPanel3);
653         jPanel3.setLayout(jPanel3Layout);
654         jPanel3Layout.setHorizontalGroup(
655             jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
656             .addGroup(jPanel3Layout.createSequentialGroup()
657                 .addGap(6, 6, 6)
658                 .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
659                     .addComponent(jScrollPane3)
660                     .addGroup(jPanel3Layout.createSequentialGroup()
661                         .addComponent(jLabel5)
662                         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
663                         .addComponent(labelChatFriend, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    Short.MAX_VALUE)
664                         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
665                         .addComponent(labelChatFriendStatus))
666                     .addGroup(jPanel3Layout.createSequentialGroup()
```

```
667                         .addComponent(txtMessage, javax.swing.GroupLayout.PREFERRED_SIZE, 269, javax.swing.GroupLayout.PREFERRED_SIZE)
668                         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE,
    Short.MAX_VALUE)
669                         .addComponent(btnSend, javax.swing.GroupLayout.PREFERRED_SIZE, 66, javax.swing.GroupLayout.PREFERRED_SIZE)
670                         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE,
    Short.MAX_VALUE)
671                         .addComponent(ButtonImage, javax.swing.GroupLayout.PREFERRED_SIZE, 56, javax.swing.GroupLayout.PREFERRED_SIZE)))
672                 .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
673         );
674         jPanel3Layout.setVerticalGroup(
675             jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
676             .addGroup(jPanel3Layout.createSequentialGroup()
677                 .addGap(6, 6, 6)
678                 .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
679                     .addComponent(jLabel5)
680                     .addComponent(labelChatFriend)
681                     .addComponent(labelChatFriendStatus))
682                 .addGap(6, 6, 6)
683                 .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE, 350, javax.swing.GroupLayout.PREFERRED_SIZE)
684                 .addGap(10, 10, 10)
685                 .addGroup(jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
686                     .addComponent(txtMessage)
687                     .addComponent(ButtonImage, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    Short.MAX_VALUE)
688                     .addComponent(btnSend, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
689                 .addContainerGap())
690         );
691 
692         txtMessage.getAccessibleContext().setAccessibleName("ChatText");
693         btnSend.getAccessibleContext().setAccessibleName("SendButton");
694         labelChatFriend.getAccessibleContext().setAccessibleName("FriendLabel");
695 
696         javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
697         getContentPane().setLayout(layout);
698         layout.setHorizontalGroup(
699             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
700             .addGroup(layout.createSequentialGroup()
```

```
701                 .addGap(10, 10, 10)
702                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
703                     .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
704                     .addGroup(layout.createSequentialGroup()
705                         .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    Short.MAX_VALUE)
706                         .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
707                         .addComponent(jPanel3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)))
708                 .addGap(10, 10, 10))
709         );
710         layout.setVerticalGroup(
711             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
712             .addGroup(layout.createSequentialGroup()
713                 .addGap(10, 10, 10)
714                 .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
715                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
716                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
717                     .addComponent(jPanel3, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
718                     .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
719                 .addGap(10, 10, 10))
720         );
721 
722         pack();
723     }// </editor-fold>//GEN-END:initComponents
724 
725     /**
726      * buttonLogoutMouseClicked()
727      * This method handles the Logout of the User
728      * @param evt
729      */
730     private void buttonLogoutMouseClicked(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_buttonLogoutMouseClicked
731         // TODO add your handling code here:
732 
733         clientService.logout();
734         Login login = Login.getInstance();
```

```
735         login.setVisible(true);
736         this.setVisible(false);
737     }//GEN-LAST:event_buttonLogoutMouseClicked
738 
739     private void btnSendMouseClicked(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_btnSendMouseClicked
740         // TODO add your handling code here:
741 
742         sendMessage();
743     }//GEN-LAST:event_btnSendMouseClicked
744 
745     private void txtMessageKeyPressed(java.awt.event.KeyEvent evt) {//GEN-FIRST:event_txtMessageKeyPressed
746         // TODO add your handling code here:
747         if(evt.getKeyCode() == java.awt.event.KeyEvent.VK_ENTER) {
748             sendMessage();
749             listChatHistory.ensureIndexIsVisible(listChatHistory.getMaxSelectionIndex());
750         }
751     }//GEN-LAST:event_txtMessageKeyPressed
752 
753     private void ButtonImageMouseClicked(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_ButtonImageMouseClicked
754         // TODO add your handling code here:
755 
756         sendImage();
757     }//GEN-LAST:event_ButtonImageMouseClicked
758 
759 
760     // Variables declaration - do not modify//GEN-BEGIN:variables
761     private javax.swing.JButton ButtonImage;
762     private javax.swing.JLabel LabelUser;
763     private javax.swing.JButton btnSend;
764     private javax.swing.JButton buttonLogout;
765     private javax.swing.JLabel jLabel1;
766     private javax.swing.JLabel jLabel3;
767     private javax.swing.JLabel jLabel4;
768     private javax.swing.JLabel jLabel5;
769     private javax.swing.JPanel jPanel1;
770     private javax.swing.JPanel jPanel2;
771     private javax.swing.JPanel jPanel3;
```

```
772     private javax.swing.JPanel jPanel4;
773     private javax.swing.JPanel jPanel5;
774     private javax.swing.JScrollPane jScrollPane1;
775     private javax.swing.JScrollPane jScrollPane2;
776     private javax.swing.JScrollPane jScrollPane3;
777     private javax.swing.JLabel labelChatFriend;
778     private javax.swing.JLabel labelChatFriendStatus;
779     private javax.swing.JList<DisplayData> listChatHistory;
780     private javax.swing.JList<String> listOfflineFriends;
781     private javax.swing.JList<String> listOnlineFriends;
782     private javax.swing.JTextField txtMessage;
783     // End of variables declaration//GEN-END:variables
784
785
786 }
787
788
```