

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-chatroom-milestone-4-2024/grade/jeo29>

IT114-002-S2024 - [IT114] Chatroom Milestone 4 2024

Submissions:

Submission Selection

1 Submission [active] 4/27/2024 6:43:20 PM

Instructions

[^ COLLAPSE ^](#)

Implement the Milestone 4 features from the project's proposal document: <https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145X>
Make sure you add your ucid/date as code comments where code changes are done
All code changes should reach the Milestone4 branch
Create a pull request from Milestone4 to main and keep it open until you get the output PDF from this assignment.
Gather the evidence of feature completion based on the below tasks.
Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
Run the necessary git add, commit, and push steps to move it to GitHub
Complete the pull request that was opened earlier
Upload the same output PDF to Canvas

Branch name: Milestone4

Tasks: 15 Points: 10.00

 Demonstrate Chat History Export (2.25 pts.)

[^ COLLAPSE ^](#)

 Task #1 - Points: 1

Text: Screenshots of code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
---	--------	---------

#1	1	Show the code that gets the messages and writes it to a file (recommended to use a StringBuilder)
#2	1	File name should be unique to avoid overwriting (i.e., incorporate timestamp)
#3	1	Screenshots should include ucid and date comment
#4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

Project > Client > `J ClientUI.java` > `ClientUI` > `exportMessage()`

```

33  public class ClientUI extends JFrame implements IClientEvents, ICardControls {
225     // UCID: jw029
226     // Date: April 26, 2024
227     public void exportMessage() {
228         try {
229             List<String> messages = Arrays.asList(chatPanel.getAllChatMessages());
230
231             // Create a JFileChooser instance
232             JFileChooser fileChooser = new JFileChooser();
233
234             // Set the file filter to accept only .txt files
235             FileNameExtensionFilter filter = new FileNameExtensionFilter(description:"Text Files", ...extensions:"txt");
236             fileChooser.setFileFilter(filter);
237
238             // Show the save dialog and get the user's choice
239             int userChoice = fileChooser.showSaveDialog(parent=null);
240
241             // If the user chooses to save the file
242             if (userChoice == JFileChooser.APPROVE_OPTION) {
243                 // Get the selected file
244                 File selectedFile = fileChooser.getSelectedFile();
245
246                 // Create a FileWriter with the selected file
247                 FileWriter fileWriter = new FileWriter(selectedFile);
248                 // Create a BufferedWriter to efficiently write characters to the file
249                 BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
250
251                 // Iterate through each message
252                 for (String mes : messages) {
253                     // Convert HTML message to plain text (You need to implement this)
254                     String plainTextMessage = mes;
255                     // Write the plain text message to the file
256                     bufferedWriter.write(plainTextMessage);
257                     bufferedWriter.newLine(); // Add a new line after each message
258                 } //-- #252-258 for (String mes : messages)
259                 // Close the BufferedWriter
260                 bufferedWriter.close();
261
262                 System.out.println("Messages exported successfully to: " + selectedFile.getAbsolutePath());
263             } else {
264                 System.out.println("Export operation canceled by the user.");
265             }
266         } catch (IOException e) {
267             // Handle the IOException
268             e.printStackTrace();
269         }
270     } //-- #227-270 public void exportMessage()
271

```

Show the code that gets the messages and writes it to a file (recommended to use a StringBuilder)

Checklist Items (1)

#1 Show the code that gets the messages and writes it to a file (recommended to use a StringBuilder)

Task #2 - Points: 1

Text: Screenshot of the file

Checklist

*The checkboxes are for your own tracking

#	Points	Details
---	--------	---------

#1	1	Show content with variation of messages (i.e., flip, roll, formatting, etc)
#2	1	It should be clear who sent each message
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

The screenshots illustrate the 'Large View' and 'Small View' modes of a messaging application. In Large View, each client has its own separate window. In Small View, multiple clients are shown in a single window. The terminal window on the left shows the Java code and log output for the clients.

Flip

Checklist Items (1)

#1 Show content with variation of messages (i.e., flip, roll, formatting, etc)

The screenshots show the clients in a list format. The clients are identified by their names and their ClientId (1, 2, 3). The messages they send are varied, demonstrating different types of content like random numbers or simple statements.

```

Client - Joshua
Rooms
Joshua: Rolled a random number between 1 and 3 and got: 2
Jessica: Rolled a random number between 1 and 5 and got: 4
Jason: Rolled a random number between 1 and 3 and got: 1
Joshua: Rolled a random number between 1 and 8 and got: 2
Jessica: Rolled a random number between 1 and 2 and got: 2
Jason: Rolled a random number between 1 and 2 and got: 2

Client - Jason
Rooms
Jessica (1)
Jason (2)
Joshua (3)

Client - Jessica
Rooms
Joshua: Rolled a random number between 1 and 2 and got: 2
Jessica: This is red text
Joshua: This is blue text
Jason: This is green text
Jessica: This is bold text
Jason: This is italic text
Joshua: This is underlined text
Jessica: This text is bold, italic, underlined, and blue

```

Roll

Checklist Items (1)

#1 Show content with variation of messages (i.e., flip, roll, formatting, etc)

```

Client - Jessica
Rooms
Joshua: Rolled a random number between 1 and 2 and got: 2
Jessica (1)
Jason (2)
Joshua (3)

Client - Jason
Rooms
Joshua: Rolled a random number between 1 and 2 and got: 2
Jessica: This is red text
Joshua: This is blue text
Jason: This is green text
Jessica: This is bold text
Jason: This is italic text
Joshua: This is underlined text
Jessica: This text is bold, italic, underlined, and blue

Client - Joshua
Rooms
Jason: Rolled a random number between 1 and 2 and got: 2
Jessica: This is red text
Joshua: This is blue text
Jason: This is green text
Jessica: This is bold text
Jason: This is italic text
Joshua: This is underlined text
Jessica: This text is bold, italic, underlined, and blue

Client - Joshua
Rooms
Joshua: Rolled a random number between 1 and 2 and got: 2
Jessica (1)
Jason (2)
Joshua (3)

```

Formatting

Checklist Items (1)

#1 Show content with variation of messages (i.e., flip, roll, formatting, etc)

Task #3 - Points: 1

Text: Explain solution

▲ COLLAPSE ▲

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention where the messages are stored and how you fetched them
#2	1	Mention how the file is generated and populated

Response:

The messages are stored when a user is approved for download and is given a path. The file is generated and populated when we loop over all messages present in the chat panel and write them in.

Demonstrate Mute List Persistence (2.25 pts.)

[^COLLAPSE ^](#)

Task #1 - Points: 1

Text: Screenshots of the code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show the code that saves the mute list to a file with the name of the user it belongs to
#2	1	Show the code that loads the mute list when a ServerThread is connected
#3	1	Screenshots should include ucid and date comment
#4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Project > Server > J Room.java > Room > muteUser(String, ServerThread)

```
11  public class Room implements AutoCloseable {  
220     // UCID: jeo29  
221     // Date: April 26, 2024  
222     protected void muteUser(String usernameToMute, ServerThread mutingUser) [  
223         if (findUserThreadInRoom(usernameToMute) != null) {  
224             System.out.println("Muting user: " + usernameToMute);  
225             mutedUsers.add(new MutedUser(usernameToMute, mutingUser.getClientName()));  
226         }  
227     }  
228 }
```

```
226     // Notify the user that they have been muted by another user
227     ServerThread userThread = findUserThreadInRoom(usernameToMute);
228     if (userThread != null) {
229         userThread.sendMessage(Constants.DEFAULT_CLIENT_ID, mutingUser.getClientName() + " muted you.");
230     }
231 } else {
232     System.out.println("User " + usernameToMute + " not found or not in this room.");
233 }
234 <- #222-234 protected void muteUser(String usernameToMute, ServerThread m...
235
```

Show the code that saves the mute list to a file with the name of the user it belongs to

Checklist Items (1)

#1 Show the code that saves the mute list to a file with the name of the user it belongs to

```
Project > Server > J ServerThread.java > ...
18 public class ServerThread extends Thread {
29 |
30     // mute/unmute feature
31     // UCID: jeo29
32     // DATE: April 13, 2024
33     public boolean sendMuteUser(ServerThread mutedUser, ServerThread mutingUser){
34         currentRoom.muteUser(mutedUser.getClientName(), mutingUser);
35         Payload p = new Payload();
36         p.setPayloadType(PayloadType.MUTE);
37         p.setClientName(mutedUser.getClientName());
38         p.setMutedClientId(mutedUser.getClientId());
39         return send(p);
40     } <- #33-40 public boolean sendMuteUser(ServerThread mutedUser, ServerThr...
41
42     // UCID: jeo29
43     // Date: April 26, 2024
44     public boolean sendUnmuteUser(ServerThread unMutedUser, ServerThread mutingUser){
45         currentRoom.unmuteUser(unMutedUser.getClientName());
46         Payload p = new Payload();
47         p.setPayloadType(PayloadType.UNMUTE);
48         p.setClientName(unMutedUser.getClientName());
49         p.setMutedClientId(unMutedUser.getClientId());
50
51         return send(p);
52     } <- #44-52 public boolean sendUnmuteUser(ServerThread unMutedUser, Serve...
```

Show the code that loads the mute list when a ServerThread is connected

Checklist Items (1)

#2 Show the code that loads the mute list when a ServerThread is connected

Task #2 - Points: 1

Text: Screenshots of the demo



Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)
#2	1	This should also be reflected in the UI per related feature in this milestone
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

The screenshots show a messaging application with three clients: Client - Jessica, Client - Jason, and Client - Joshua. The interface includes a header bar, a message list, and a text input field with a 'Send' button. The 'Rooms' section shows a list of users with their names and counts. The conversation log shows messages from both users, with some messages colored red.

Client - Jessica:

- Rooms: Jessica (4), Joshua (6), Jason (5)
- Messages: Hello, How is everyone?, I am well, I am well, Good to know, how's the weather where you're at?, The weather is good, I can't complain, I agree, Describe the weather for me?, Server: Joshua has been muted, Nice

Client - Jason:

- Rooms: Jason (Hello), Hello, How is everyone?, I am well, I am well, Good to know, how's the weather where you're at?, The weather is good, I can't complain, I agree, Describe the weather for me?, Server: Joshua has been muted, Nice

Client - Joshua:

- Rooms: Jason (Hello), Hello, How is everyone?, I am well, I am well, Good to know, how's the weather where you're at?, The weather is good, I can't complain, I agree, Describe the weather for me?, Server: Jason muted you, Jason muted you, Bit chilly but the sun is out, Nice

Bottom Left:

```
pr 27, 2024 9:31:03 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(6.) ClientName(null). Message[how's the weather]
ext color set to red for client: 6
pr 27, 2024 9:31:21 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(4.) ClientName(null). Message[The weather is go]
ext color set to red for client: 4
pr 27, 2024 9:31:34 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(6.) ClientName(null). Message[I agree]
ext color set to red for client: 6
pr 27, 2024 9:31:45 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(4.) ClientName(null). Message[Describe the weat
ext color set to red for client: 4
pr 27, 2024 9:31:59 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(-1.) ClientName(null). Message[Jason muted you]
pr 27, 2024 9:32:06 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(6.) ClientName(null). Message[Bit chilly but th
ext color set to red for client: 6
pr 27, 2024 9:32:42 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(4.) ClientName(null). Message[Nice]
ext color set to red for client: 4
pr 27, 2024 9:32:46 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE].ClientId(4.) ClientName(null). Message[I guess]
ext color set to red for client: 4
```

User muted

Checklist Items (1)

#1 Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)

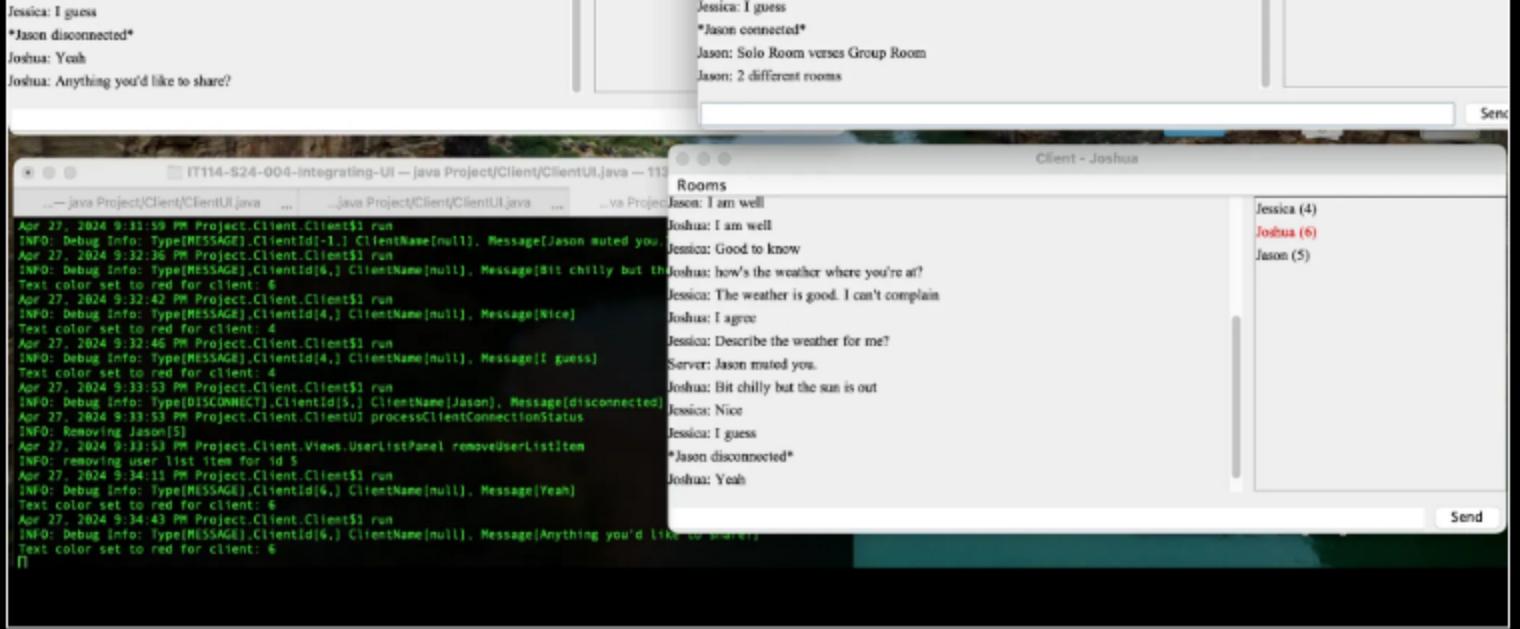
The screenshot shows a messaging application with two clients: Client - Jessica and Client - Jason. The interface includes a header bar, a message list, and a text input field with a 'Send' button. The 'Rooms' section shows a list of users with their names and counts. The conversation log shows messages from both users, with some messages colored red.

Client - Jessica:

- Rooms: Jason (I am well), I am well, Good to know, how's the weather where you're at?, The weather is good, I can't complain, I agree, Describe the weather for me?, Bit chilly but the sun is out, Nice

Client - Jason:

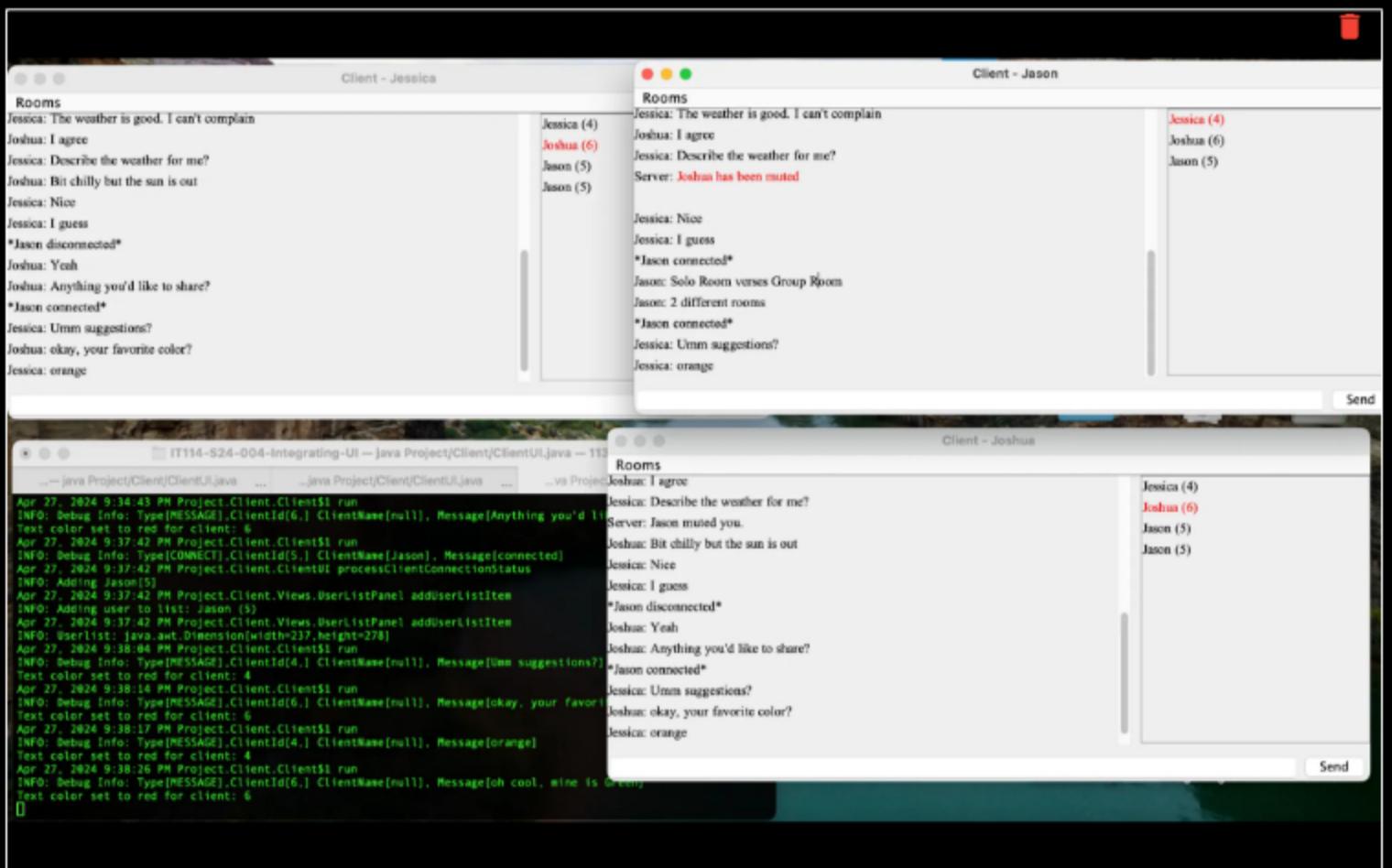
- Rooms: Jason (I am well), Hello, How is everyone?, I am well, I am well, Good to know, how's the weather where you're at?, The weather is good, I can't complain, I agree, Describe the weather for me?, Server: Joshua has been muted, Nice



Jason disconnecting into his own room all while still having Joshua muted

Checklist Items (1)

#1 Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)



Jason reconnecting and still has Joshua muted

Checklist Items (1)

#1 Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)

^COLLAPSE ^

Task #3 - Points: 1

Text: Explain solution

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention how you got the mute list to save and load
#2	1	Discuss the steps to sync the data to the client/ui

Response:

The mute list was saved and loaded by being stored in List and passed from Payload to Client UI, and from ClientUI users were grayed out from UI which were muted. Then back to being unmuted because they were no longer grayed out

^COLLAPSE ^

Demonstrate Mute/Unmute notification (2.25 pts.)

^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshots of the code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show how the message is sent to the target user only if their mute/unmute state changes (i.e., doing mute twice for the same user shouldn't send two mute messages)
#2	1	Screenshots should include ucid and date comment
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```

31 //    //UCID: jeo29
32 //    //DATE: April 13, 2024
33 public boolean sendMuteUser(ServerThread mutedUser, ServerThread mutingUser){
34     if (currentRoom.isUserMuted(mutedUser.getClientName(),mutingUser.getClientName())) {
35         return false; // Don't send mute notification if user is already muted
36     }
37     currentRoom.muteUser(mutedUser.getClientName(),mutingUser);
38     Payload p = new Payload();
39     p.setPayloadType(PayloadType.MUTE);
40     p.setClientName(mutedUser.getClientName());
41     p.setMutedClientID(mutedUser.getClientId());
42     return send(p);

```

```

43 } <- #33-43 public boolean sendMuteUser(ServerThread mutedUser, ServerThr...
44
45 // UCID: jeo29
46 // Date: April 27, 2024
47 public boolean sendUnmuteUser(ServerThread unMutedUser, ServerThread mutingUser){
48     if (!currentRoom.isUserMuted(unMutedUser.getClientName(), mutingUser.getClientName())) {
49         return false; // Don't send mute notification if user is already muted
50     }
51     currentRoom.unmuteUser(unMutedUser.getClientName());
52     Payload p = new Payload();
53     p.setPayloadType(PayloadType.UNMUTE);
54     p.setClientName(unMutedUser.getClientName());
55     p.setMutedClientId(unMutedUser.getClientId());
56
57     return send(p);
58 } <- #47-58 public boolean sendUnmuteUser(ServerThread unMutedUser, Serve...

```

Show how the message is sent to the target user only if their mute/unmute state changes (i.e., doing mute twice for the same user shouldn't send two mute messages)

Checklist Items (1)

#1 Show how the message is sent to the target user only if their mute/unmute state changes (i.e., doing mute twice for the same user shouldn't send two mute messages)

Task #2 - Points: 1

Text: Screenshots of the demo

Checklist

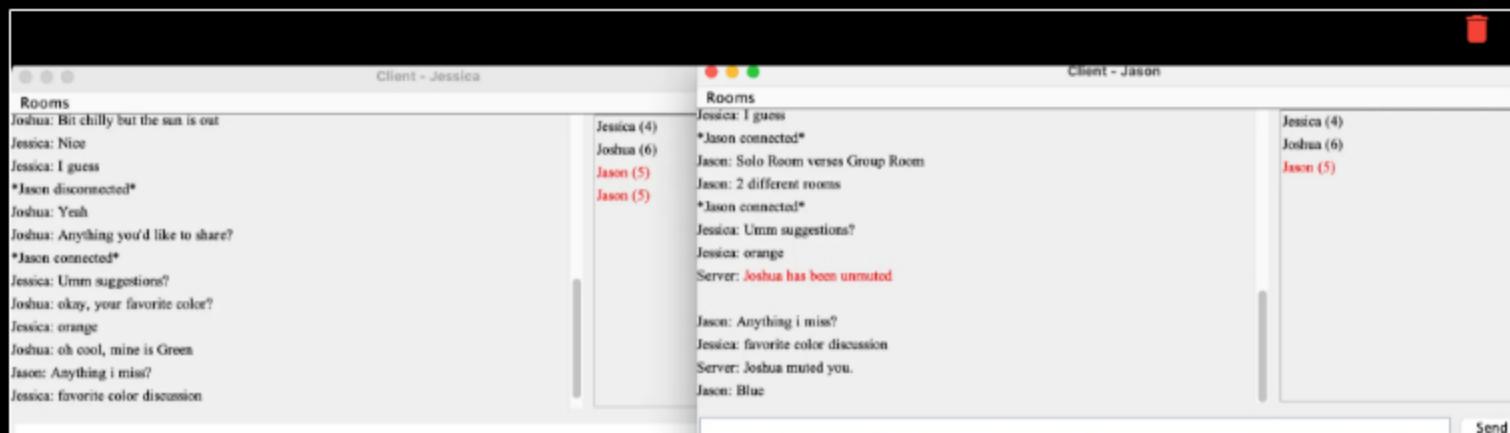
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show examples of doing /mute twice in succession for the same user only yields one message
#2	1	Show examples of doing /unmute twice in succession for the same user only yields one message
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Client - Joshua

```

Rooms
Joshua: Yeah
Joshua: Anything you'd like to share?
*Jason connected*
Jessica: Umm suggestions?
Joshua: okay, your favorite color?
Text color set to red for client: 6
Joshua: orange
Joshua: oh cool, mine is Green
Server: You have been unmuted by another user.

Jason: Anything i miss?
Jessica: favorite color discussion
Server: Jason has been muted

```

INFO: Debug Info: Type[MESSAGE],ClientId[4,] ClientName=null, Message[Umm suggestions?]
 Text color set to red for client: 4
 Apr 27, 2024 9:38:14 PM Project.Client.Client\$1 run
 INFO: Debug Info: Type[MESSAGE],ClientId[4,] ClientName=null, Message[okay, your favorite color?]
 Text color set to red for client: 6
 Apr 27, 2024 9:38:17 PM Project.Client.Client\$1 run
 INFO: Debug Info: Type[MESSAGE],ClientId[4,] ClientName=null, Message[orange]
 Text color set to red for client: 4
 Apr 27, 2024 9:38:19 PM Project.Client.Client\$1 run
 INFO: Debug Info: Type[MESSAGE],ClientId[6,] ClientName=null, Message[oh cool, mine is Green]
 Text color set to red for client: 6
 Apr 27, 2024 9:38:33 PM Project.Client.Client\$1 run
 INFO: Debug Info: Type[MESSAGE],ClientId[-1,] ClientName=null, Message[You have been unmuted by another user.]
 INFO: Debug Info: Type[MUTE],ClientId[0,] ClientName[jason], Message[null]

Mute Joshua

Checklist Items (1)

#1 Show examples of doing /mute twice in succession for the same user only yields one message

Client - Jessica

Rooms

```

Rooms
Jessica: The weather is good. I can't complain
Joshua: I agree
Jessica: Describe the weather for me?
Joshua: Bit chilly but the sun is out
Jessica: Nice
Jessica: I guess
*Jason disconnected*
Joshua: Yeah
Joshua: Anything you'd like to share?
*Jason connected*
Jessica: Umm suggestions?
Joshua: okay, your favorite color?
Jessica: orange

```

Client - Jason

Rooms

```

Rooms
Jessica: Describe the weather for me?
Server: Joshua has been muted
Jessica: Nice
Jessica: I guess
*Jason connected*
Jason: Solo Room verses Group Room
Jason: 2 different rooms
*Jason connected*
Jessica: Umm suggestions?
Jessica: orange
Server: Joshua has been unmuted

```

Client - Joshua

Rooms

```

Rooms
Joshua: Bit chilly but the sun is out
Jessica: Nice
Jessica: I guess
*Jason disconnected*
Joshua: Yeah
Joshua: Anything you'd like to share?
*Jason connected*
Jessica: Umm suggestions?
Joshua: okay, your favorite color?
Jessica: orange
Joshua: oh cool, mine is Green
Server: You have been unmuted by another user.

INFO: Debug Info: Type[CONNECT],ClientId[5,] ClientName[jason], Message[connected]
INFO: Adding Jason(5)
Apr 27, 2024 9:37:42 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[5,] ClientName[jason], Message[connected]
INFO: UserList: java.awt.Dimension@width=237,height=278]
Apr 27, 2024 9:38:04 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[4,] ClientName=null, Message[Umm suggestions?]
Text color set to red for client: 4
Apr 27, 2024 9:38:14 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[6,] ClientName=null, Message[okay, your favorite color?]
Text color set to red for client: 6
Apr 27, 2024 9:38:17 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[4,] ClientName=null, Message[orange]
Text color set to red for client: 4
Apr 27, 2024 9:38:19 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[6,] ClientName=null, Message[oh cool, mine is Green]
Text color set to red for client: 6
Apr 27, 2024 9:38:33 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[-1,] ClientName=null, Message[You have been unmuted by another user.]
INFO: Debug Info: Type[MUTE],ClientId[0,] ClientName[jason], Message[null]

```

/unmute twice in succession for the same user only yields one message

Checklist Items (1)

#2 Show examples of doing /unmute twice in succession for the same user only yields one message

Task #3 - Points: 1

Text: Explain solution

COLLAPSE ^

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention how you limit the messages in each scenario
#2	1	Discuss how you find the correct user to send the message to

Response:

In each scenario, the messages are limited with the help of the mute/unmute feature. You can find the correct user to send a message to with the "@" feature to send a private message.

Demonstrate user list visual changes (2.25 pts.)



Task #1 - Points: 1

 Text: Screenshots of the code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show the code related to "graying out" muted users and returning them to normal when unmuted
#2	1	Show the code related to highlighting the user who last sent a message (and unhighlighting the remainder of the list)
#3	1	Screenshots should include ucid and date comment
#4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Project > Client >  Client.java >  Client >  processPayload(Payload)

```
16  im Client {  
189   e void processPayload(Payload p) {  
226     case MUTE:  
227       // UCID: jeo29  
228       // Date: April 26, 2024  
229       events.onMuteUserInOnMutedClientID().<font color =\"gray\">>" + p.getClientName() + " (" + p.getMutedClientID() + ")" + </font>";
```

```
230 events.onMessageReceive(p.getClientId(), "<font color =\"red\>" + p.getClientName() + " has been muted</font>");
231 break;
232 case UNMUTE:
233 // UCID: jeo29
234 // Date: April 26, 2024
235 events.onUnMuteUser(p.getMutedClientID(),"<font color =\"black\>" + p.getClientName() + " (" + p.getMutedClientID() + ")" + "</font>");
236 events.onMessageReceive(p.getClientId(), "<font color =\"red\>" + p.getClientName() + " has been unmuted</font>");
237 break;
--
```

Show the code related to “graying out” muted users and returning them to normal when unmuted

Checklist Items (1)

#1 Show the code related to “graying out” muted users and returning them to normal when unmuted

```
62 // UCID: jeo29
63 // Date: April, 26, 2024
64 protected void recentUser(long clientId) {
65     // Iterate through the components in the userListArea
66     Component[] cs = userListArea.getComponents();
67     for (Component component : cs) {
68         // Check if the component is a JEditorPane and its name matches the clientId
69         if (component.getName().equals(clientId + " ")) {
70             JEditorPane namePane = (JEditorPane) component;
71             // Update the HTML content to change text color to red
72             String htmlContent = namePane.getText().replace("color=\\\"black\\\"", "color=\\\"" + "red" + "\\\"");
73             namePane.setText(htmlContent);
74             namePane.setName(clientId + " ");
75             System.out.println("Text color set to red for client: " + clientId);
76         } <- #69-76 if (component.getName().equals(clientId + " "))
77         else {
78             JEditorPane namePane = (JEditorPane) component;
79             // Update the HTML content to change text color to black
80             String htmlContent = namePane.getText().replace("color=\\\"red\\\"", "color=\\\"" + "black" + "\\\"");
81             namePane.setText(htmlContent);
82         } <- #77-82 else
83     } <- #67-83 for (Component component : cs)
84     userListArea.revalidate();
85     userListArea.repaint();
86 } <- #64-86 protected void recentUser(long clientId)
--
```

Show the code related to highlighting the user who last sent a message (and unhighlighting the remainder of the list)

Checklist Items (1)

#2 Show the code related to highlighting the user who last sent a message (and unhighlighting the remainder of the list)

Task #2 - Points: 1

Text: Screenshots of the demo



Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show before and after screenshots of the list updating upon mute and unmute
#2	1	Capture variations of "last person to send a message gets highlighted"
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

The screenshot displays three windows. On the left is a terminal window showing Java code and log output. In the center is a window titled 'Client - Jessica' showing a list of messages. On the right is a window titled 'Client - Jason' showing a similar list. Both windows have a 'Rooms' tab at the top. The messages are color-coded: red for Jason (1), blue for Jason (2), and green for Jason (3). The terminal window shows code for ClientUI.java and ClientUI.java, along with log entries for INFO and DEBUG levels.

Client - Jessica

Client - Jason

Rooms

Jason: Jason flipped a coin and got heads

Joshua: privatemessage to you: Hi

Jessica: You privatemessage to Joshua: Hello

Server: Jason has been muted

Jessica: Hope all is well?

Joshua: All is well

Jessica: Great

Server: Jason has been muted

Rooms

Jessica (1)
Jason (2)
Joshua (3)

Client - Jason

Rooms

umber between 1 and 2 and got: 2

umber between 1 and 20 and got: 3

and got heads

Jessica (1)
Jason (2)
Joshua (3)

IT114-S24-004-Integrating-UI -- java Project\Client\ClientUI.java -- 80x24

Apr 27, 2024 7:38:21 PM Project.Client.Client\$1 run
INFO: Debug Info: Type(MESSAGE),ClientId[3,] ClientName(null), Message[You privatemessage to Jessica: Hi]
Text color set to red for client: 3

Apr 27, 2024 7:38:29 PM Project.Client.Client\$1 run
INFO: Debug Info: Type(MESSAGE),ClientId[1,] ClientName(null), Message[privatemessage to you: Hello]
Text color set to red for client: 1

Apr 27, 2024 7:38:42 PM Project.Client.Client\$1 run
INFO: Debug Info: Type(MESSAGE),ClientId[1,] ClientName(null), Message[Hope all is well?]

Text color set to red for client: 1

Apr 27, 2024 7:38:53 PM Project.Client.Client\$1 run
INFO: Debug Info: Type(MESSAGE),ClientId[2,] ClientName(null), Message[Yes, all is well]

Text color set to red for client: 2

Apr 27, 2024 7:38:57 PM Project.Client.Client\$1 run
INFO: Debug Info: Type(MESSAGE),ClientId[3,] ClientName(null), Message[All is well!]

Text color set to red for client: 3

Apr 27, 2024 8:05:29 PM Project.Client.Client\$1 run
INFO: Debug Info: Type(MESSAGE),ClientId[1,] ClientName(null), Message[Great!]

Text color set to red for client: 1

Joshua: Rolled a random number between 1 and 20 and got: 3

Jason: Jason flipped a coin and got heads

Joshua: You privatemessage to Jessica: Hi

Jessica: privatemessage to you: Hello

Jessica: Hope all is well?

Jason: Yes, all is well

Joshua: All is well

Jessica: Great

Send

Send

Send

Jason muted

Checklist Items (1)

#1 Show before and after screenshots of the list updating upon mute and unmute

The screenshot displays three windows. On the left is a terminal window showing Java code and log output. In the center is a window titled 'Client - Jessica' showing a list of messages. On the right is a window titled 'Client - Jason' showing a similar list. Both windows have a 'Rooms' tab at the top. The messages are color-coded: red for Jason (1), blue for Jason (2), and green for Jason (3). The terminal window shows code for ClientUI.java and ClientUI.java, along with log entries for INFO and DEBUG levels.

Client - Jessica

Client - Jason

Rooms

Joshua: privatemessage to you: Hi

Jessica: You privatemessage to Joshua: Hello

Server: Jason has been muted

Jessica: Hope all is well?

Rooms

Jessica (1)
Jason (2)
Joshua (3)

Client - Jason

Rooms

Joshua: Rolled a random number between 1 and 20 and got: 3

Jason: Jason flipped a coin and got heads

Server: Jessica muted you.

Jessica: Hope all is well?

Jason: Yes, all is well

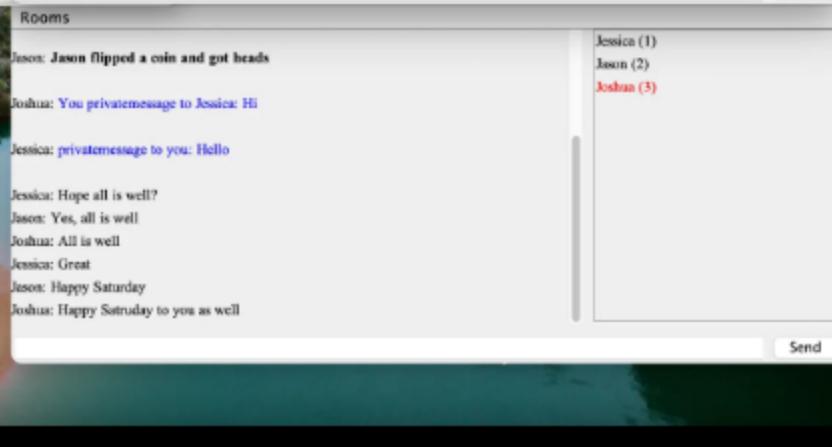
Jessica (1)
Jason (2)
Joshua (3)

Joshua: All is well
Jessica: Great
Server: Jason has been muted

Joshua: Happy Saturday to you as well

Joshua: All is well
Jessica: Great
Server: Jason muted you.
Jason: Happy Saturday
Joshua: Happy Saturday to you as well

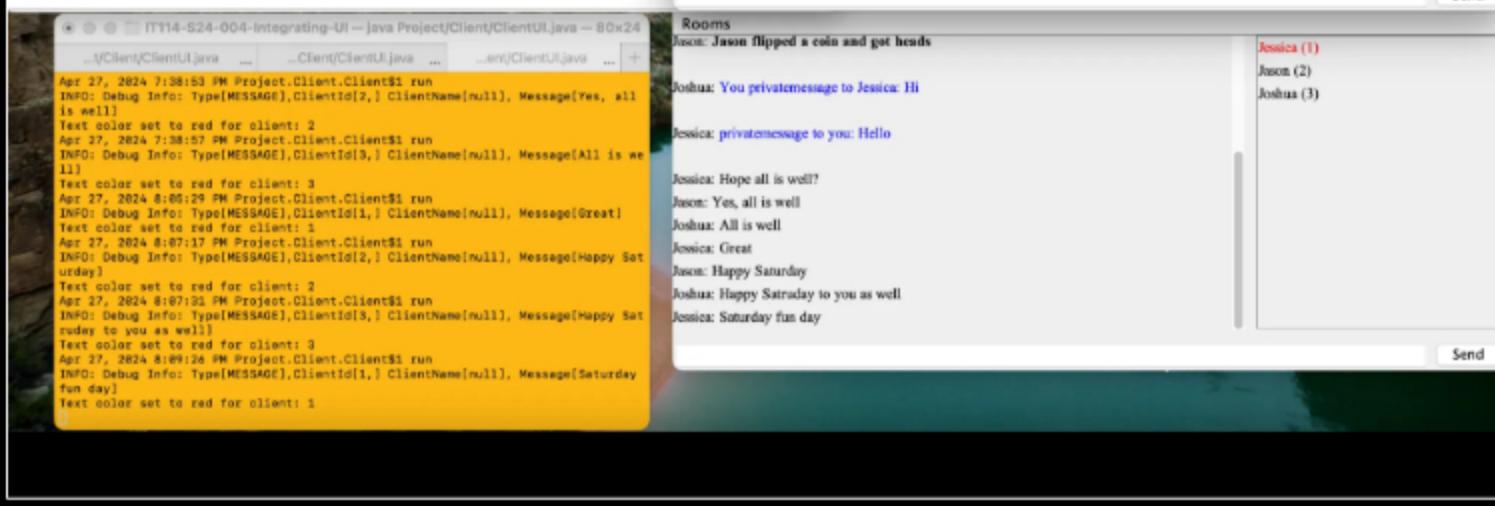
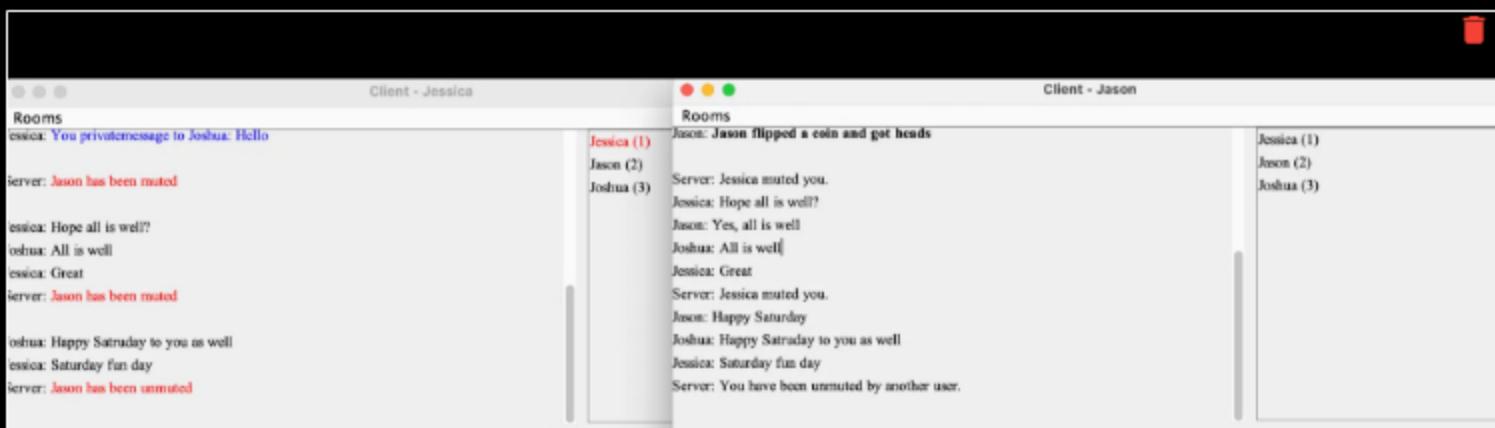
```
IT114-S24-004-Integrating-UI -- java Project/Client/ClientUI.java -- 80x24
...t/Client/ClientUI.java ... .Client/ClientUI.java ... .ser/ClientUI.java ... +
Apr 27, 2024 7:38:42 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Hope all is well?]
Text color set to red for client: 1
Apr 27, 2024 7:38:53 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[2,] ClientName[null], Message[Yes, all is well]
Text color set to red for client: 2
Apr 27, 2024 7:38:57 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[3,] ClientName[null], Message[All is well!]
Text color set to red for client: 3
Apr 27, 2024 8:05:29 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[1,] ClientName[null], Message[Great]
Text color set to red for client: 1
Apr 27, 2024 8:07:17 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[2,] ClientName[null], Message[Happy Saturday]
Text color set to red for client: 2
Apr 27, 2024 8:07:31 PM Project.Client.Client$1 run
INFO: Debug Info: Type[MESSAGE],ClientId[3,] ClientName[null], Message[Happy Saturday to you as well!]
Text color set to red for client: 3
```



Jason muted Part 2

Checklist Items (1)

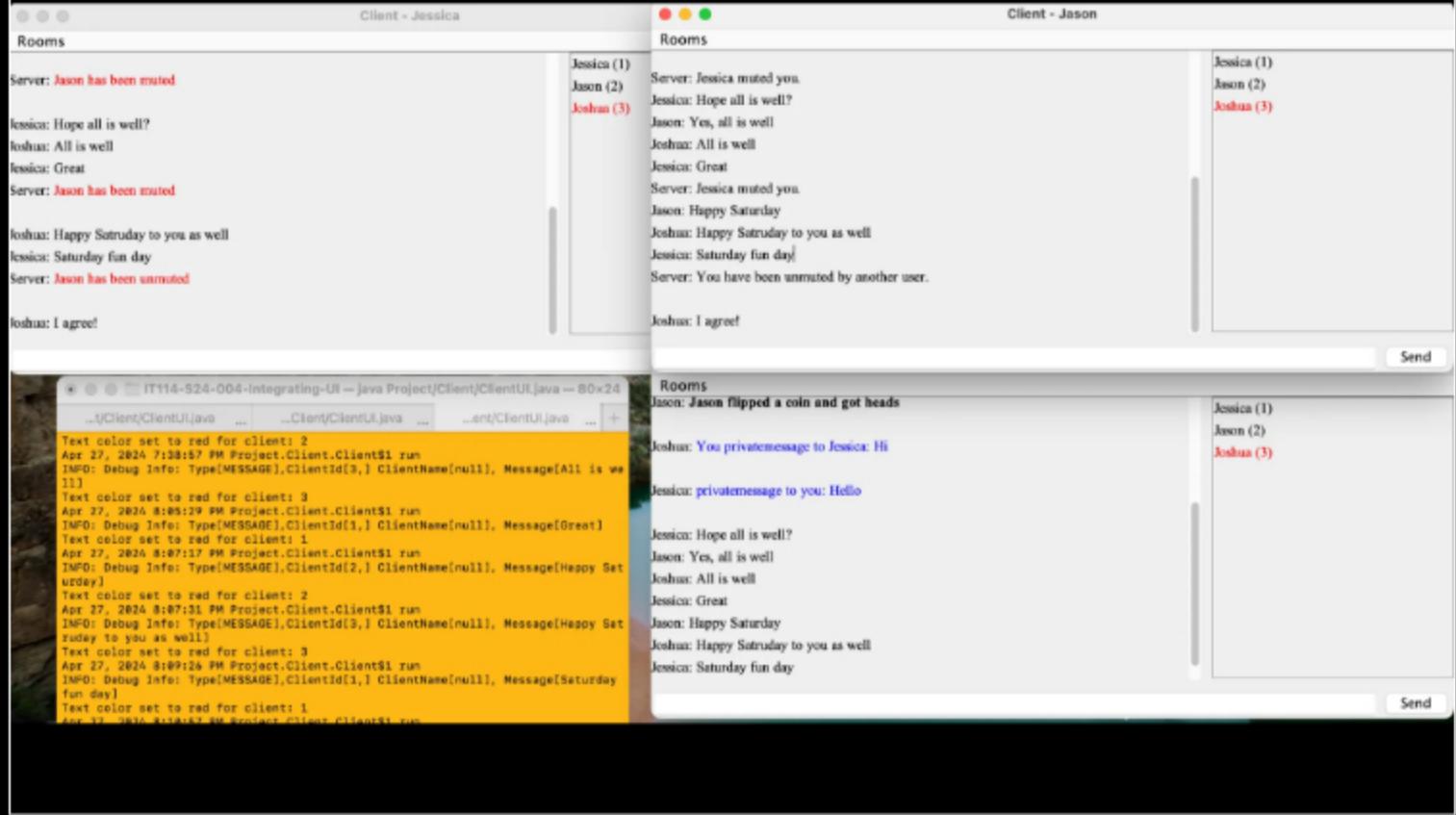
#1 Show before and after screenshots of the list updating upon mute and unmute



Jason unmute

Checklist Items (1)

#1 Show before and after screenshots of the list updating upon mute and unmute



"last person to send a message gets highlighted"

Checklist Items (1)

#2 Capture variations of "last person to send a message gets highlighted"

Task #3 - Points: 1

Text: Explain solution

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention how you got the mute/unmute effect implemented
#2	1	Mentioned how you got the highlight effect implemented (including unhighlighting the other users)

Response:

When a mute/unmute payload is received and processed the client side is triggered. For the /mute action, the muted user's name is displayed in gray, and a red message indicates that the user has been muted. Similarly, for /unmute, the user's name is shown in black with the message showing the action that was completed.

The visual changes are based on exactly what is being done to the user.

Misc (1 pt.)

▲ COLLAPSE ▲

COLLAPSE

Task #1 - Points: 1

Text: Add the pull request link for the branch

Details:

Note: the link should end with /pull/#

URL #1

<https://github.com/jessicaodoom/jeo29-IT114-002/pull/19>

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

The issues that I had during this assignment first stemmed from my laptop being overloaded when running the clients and then being able to export the chat. My device froze during the second time I tried to export a chat. I had to force quit. My next issue was trying to connect with the clients. The client would start, and allow me to enter a name, but once I selected connect, it wouldn't connect. For some odd reason, it wouldn't start up anymore. I reached out to the professor who told me to make sure nothing else was trying to connect to the port. That didn't help so I ended up uninstalling Java just starting the needed installations for this class from the beginning to see if that would help. I knew there was no issue with the code because I had not made any changes within my code when the client connect stopped working.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

Details:

Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved

Task Screenshots:

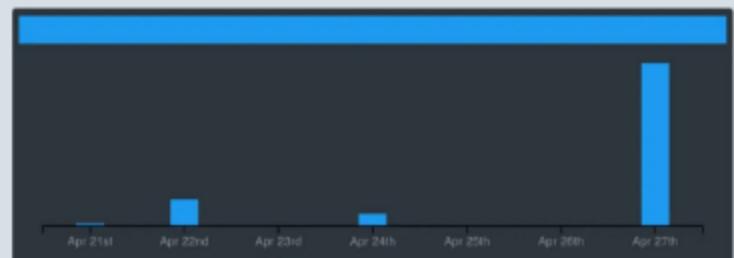
Gallery Style: Large View

Small

Medium

Large

2 hrs 23 mins over the Last 7 Days. 



 **1 hr 53 mins Today** 



IT114-S24-004-Integrating-UI 1:53



 **1 hr 53 mins Today** 



Java 1:53



Final time April 27, 2024

End of Assignment