Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-chatroom-milestone-3-2024/grade/jeo29

IT114-002-S2024 - [IT114] Chatroom Milestone 3 2024

Submissions:

Submission Selection

1 Submission [active] 4/14/2024 11:30:56 PM

Instructions

COLLAPSE

1. Implement the Milestone 3 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145Xi
2. Make sure you add your ucid/date as code comments where code changes are done
3. All code changes should reach the Milestone3 branch
4. Create a pull request from Milestone3 to main and keep it open until you get the output PDF from this assignment.
5. Gather the evidence of feature completion based on the below tasks.
6. Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
7. Run the necessary git add, commit, and push steps to move it to GitHub
8. Complete the pull request that was opened earlier
9. Upload the same output PDF to Canvas

Branch name: Milestone3

Tasks: 14 Points: 10.00

Basic UI (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the following

Checklist *The checkbox are for your own tracking

#	Points	Details

#1	1	Connection Panel
#2	1	User Details Panel
#3	1	Chat Panel
#4	1	Clearly caption screenshots

Task Screenshots:

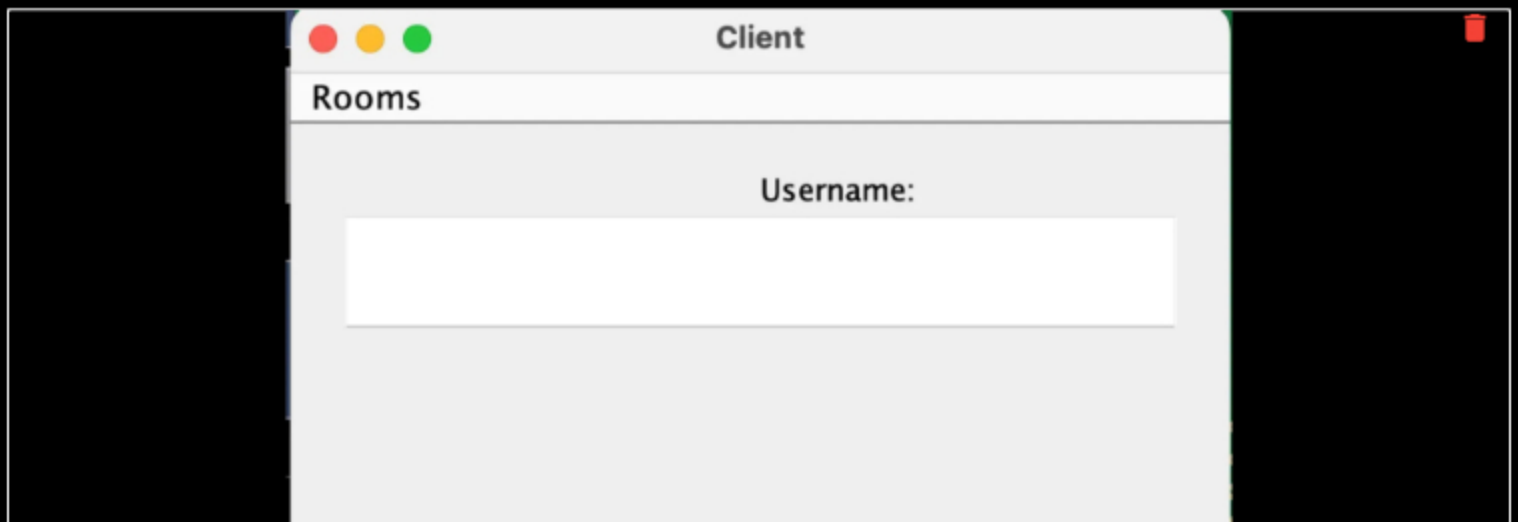
Gallery Style: Large View

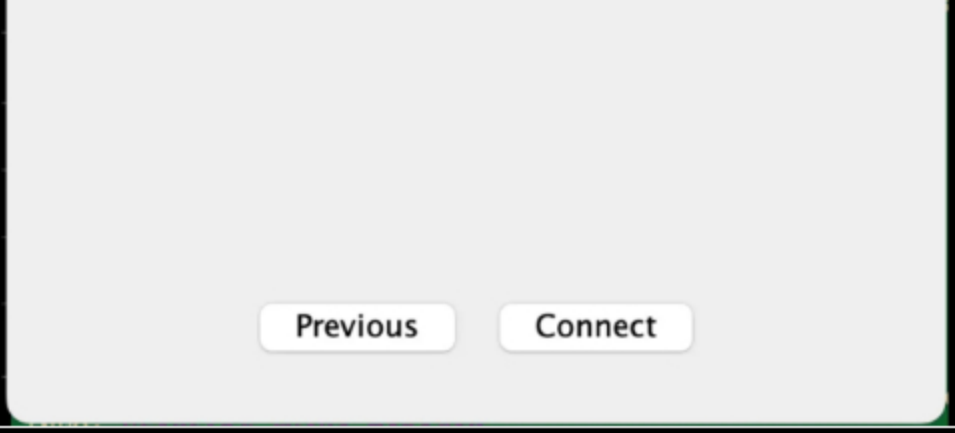
Small Medium Large

Connection Panel

Checklist Items (1)

#1 Connection Panel





User Details Panel

Checklist Items (1)

#2 User Details Panel

Chat Panel

Checklist Items (1)

#3 Chat Panel

Formatting (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing flip and roll commands

Checklist

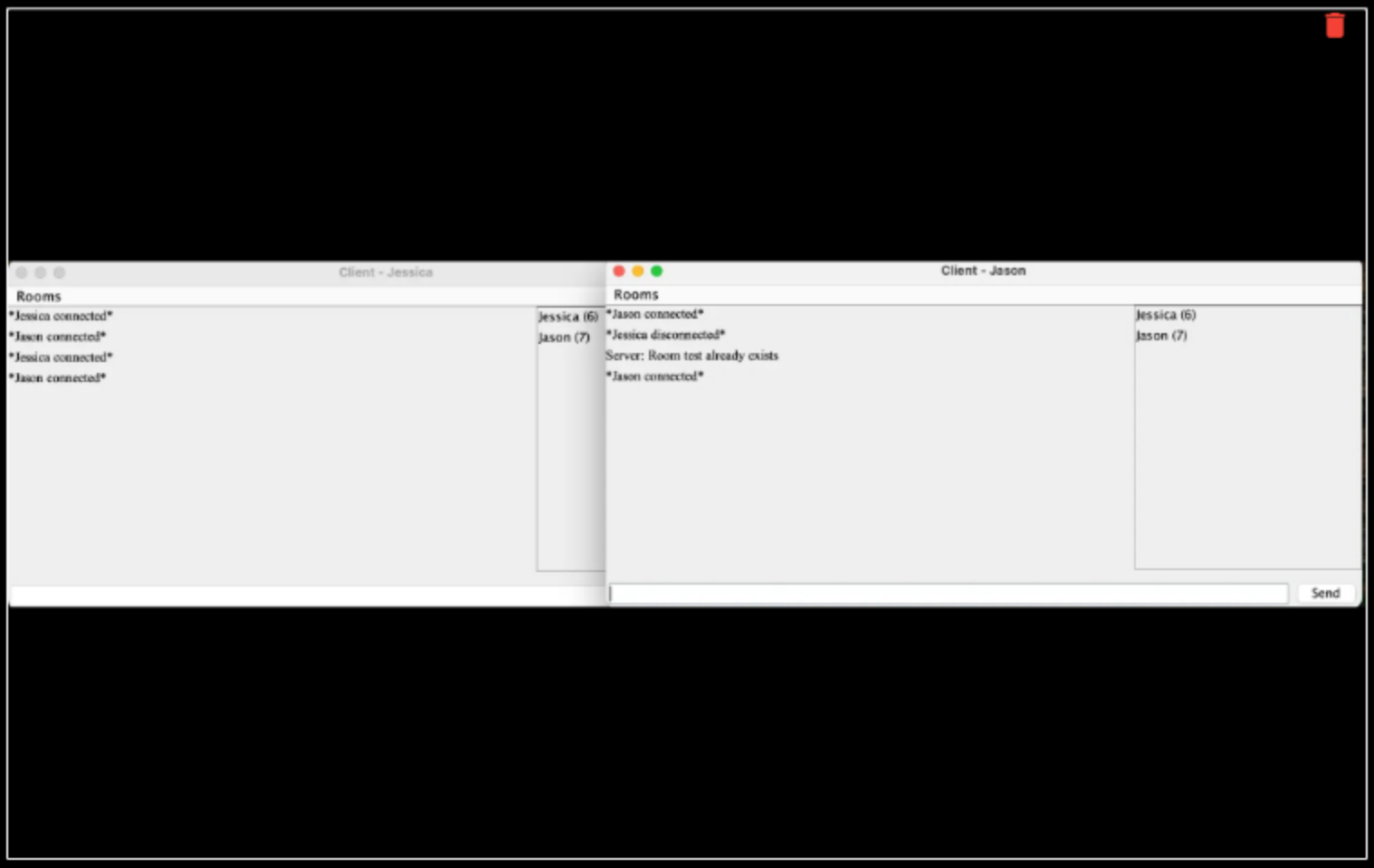
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Flip output in a different format than normal messages
#2	1	Roll # output in a different format than normal messages
#3	1	Roll #d# output in a different format than normal messages
#4	1	Clearly caption screenshots

Task Screenshots:

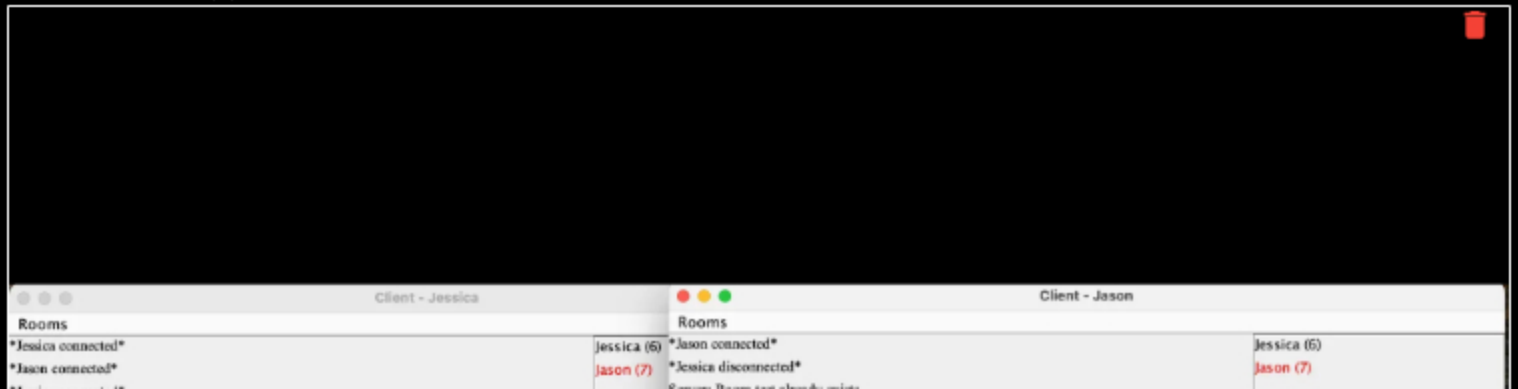
Gallery Style: Large View

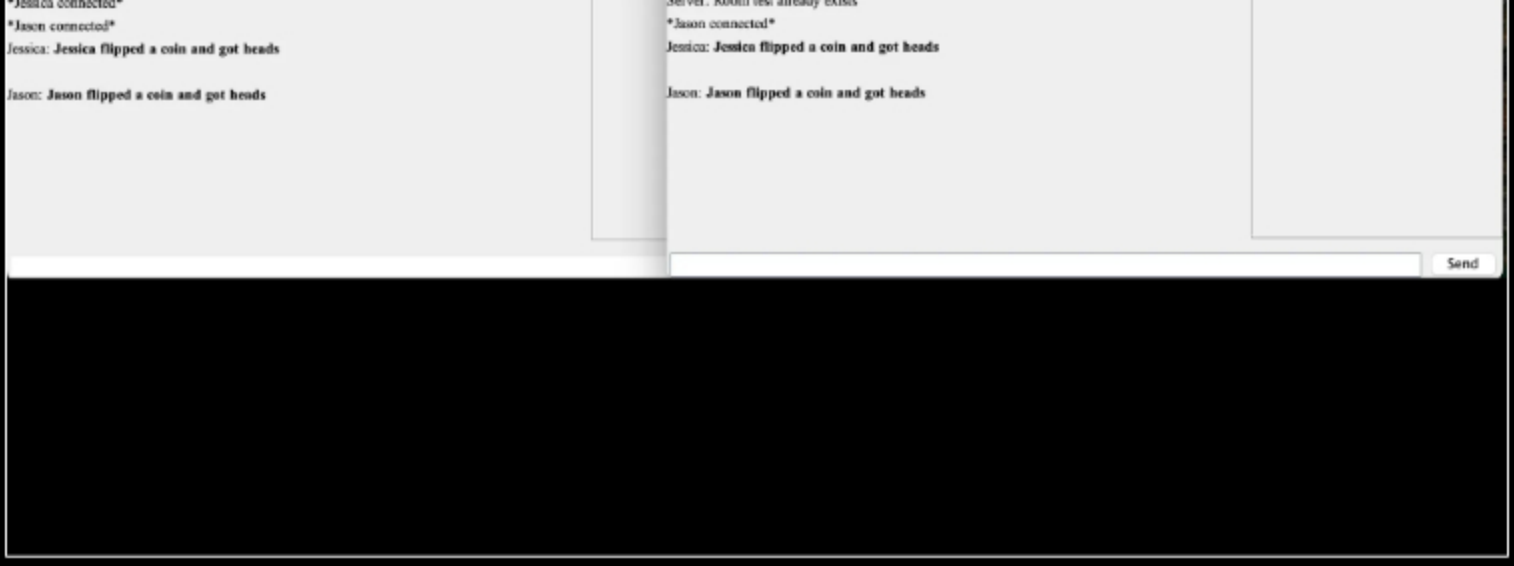
Small Medium Large



Connection

Checklist Items (0)

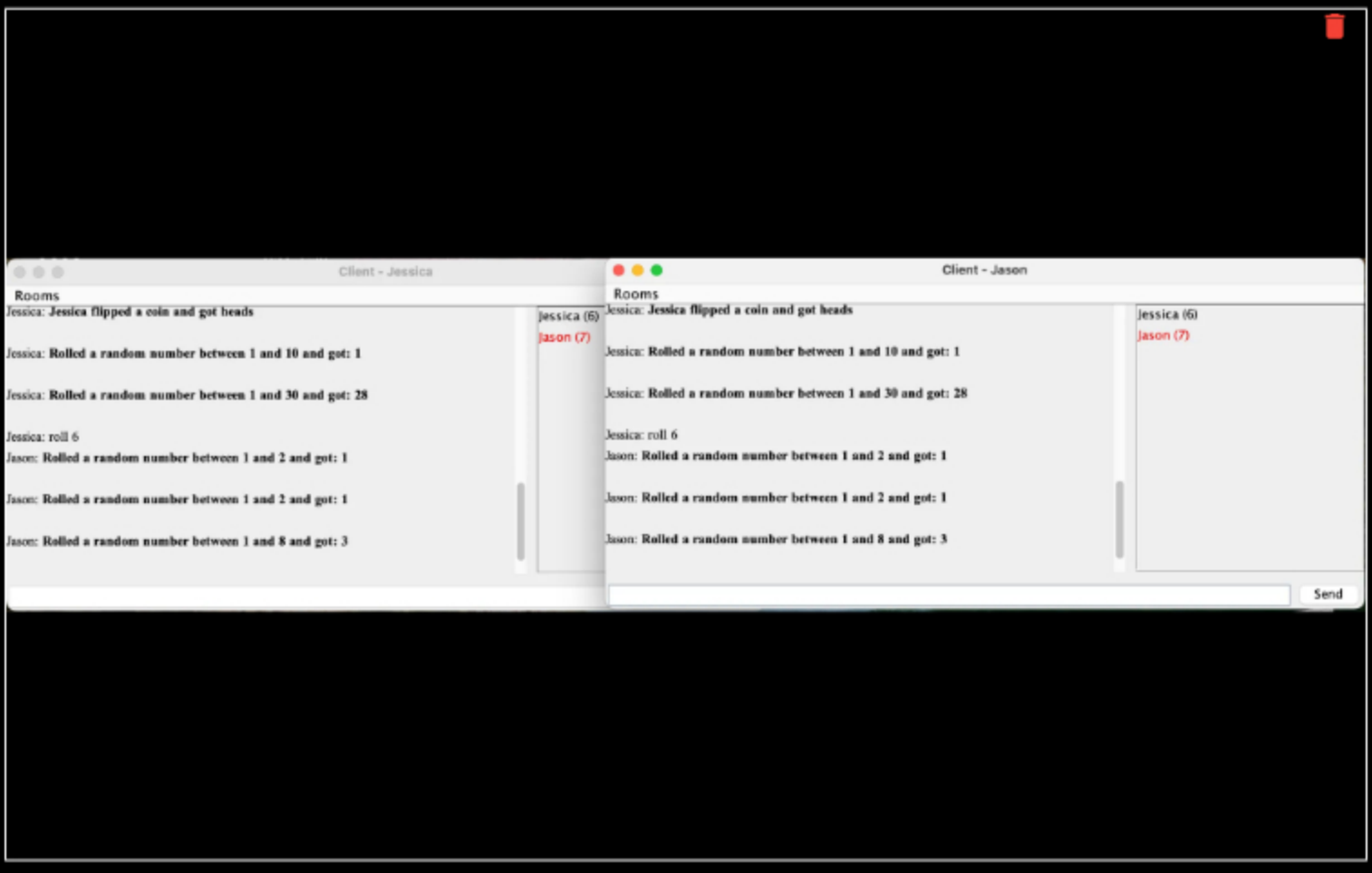




Flip

Checklist Items (1)

#1 Flip output in a different format than normal messages



Roll

Checklist Items (1)

#3 Roll #d# output in a different format than normal messages

COLLAPSE

Task #2 - Points: 1

Text: Screenshots demoing custom text formatting

Checklist

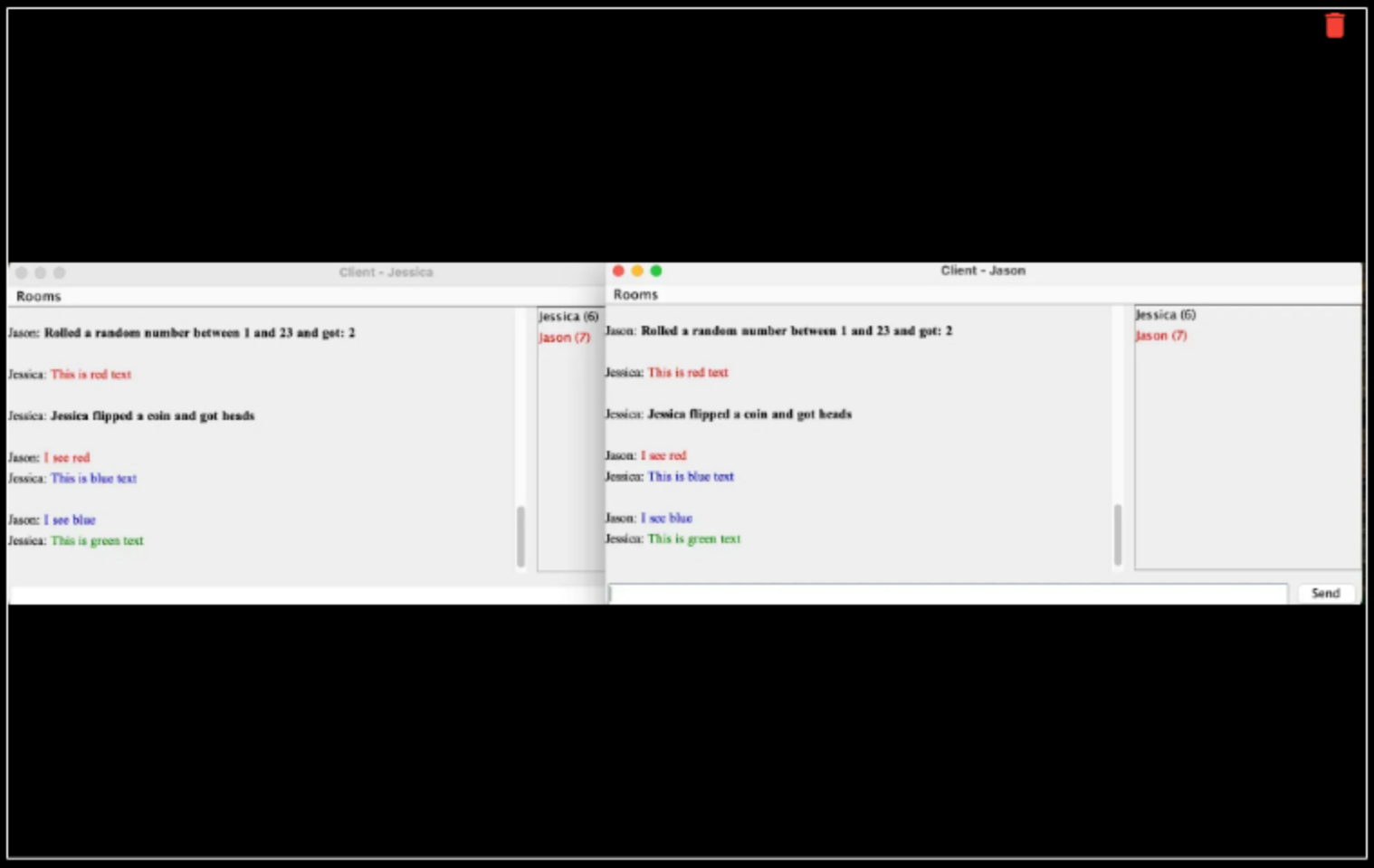
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Custom text formatting for bold working (Part of the message should appear bold)
#2	1	Custom text formatting for italic working (Part of the message should appear italic)
#3	1	Custom text formatting for underline working (Part of the message should appear underline)
#4	1	Custom text formatting for red working (Part of the message should appear red)
#5	1	Custom text formatting for blue working (Part of the message should appear blue)
#6	1	Custom text formatting for green working (Part of the message should appear green)
#7	1	Custom text formatting for combined bold, italic, underline, and a color working (Part of the message should have all 4 formats applied at once)
#8	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

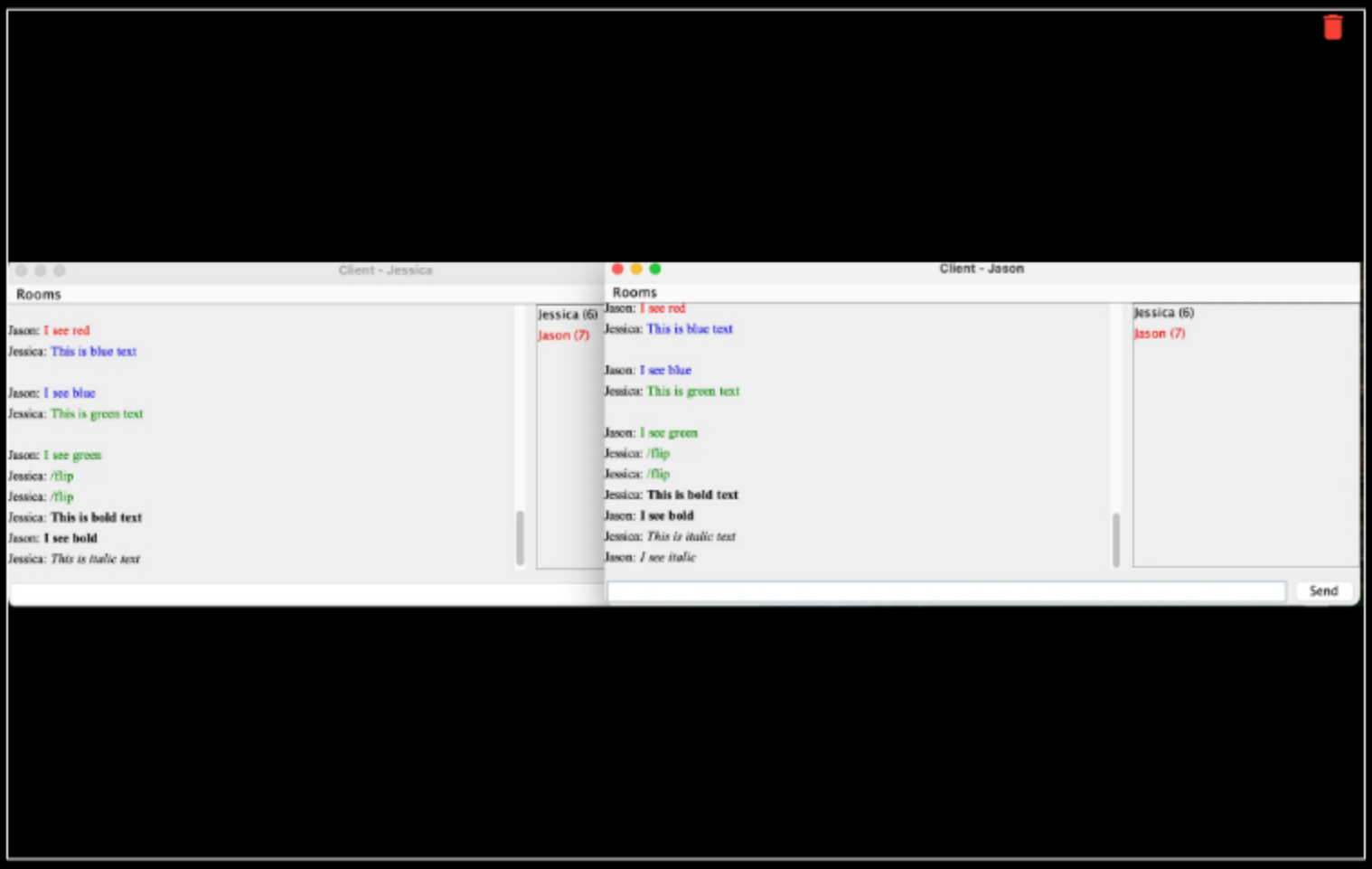
Small Medium Large



Custom text formatting for red, blue, green

Checklist Items (1)

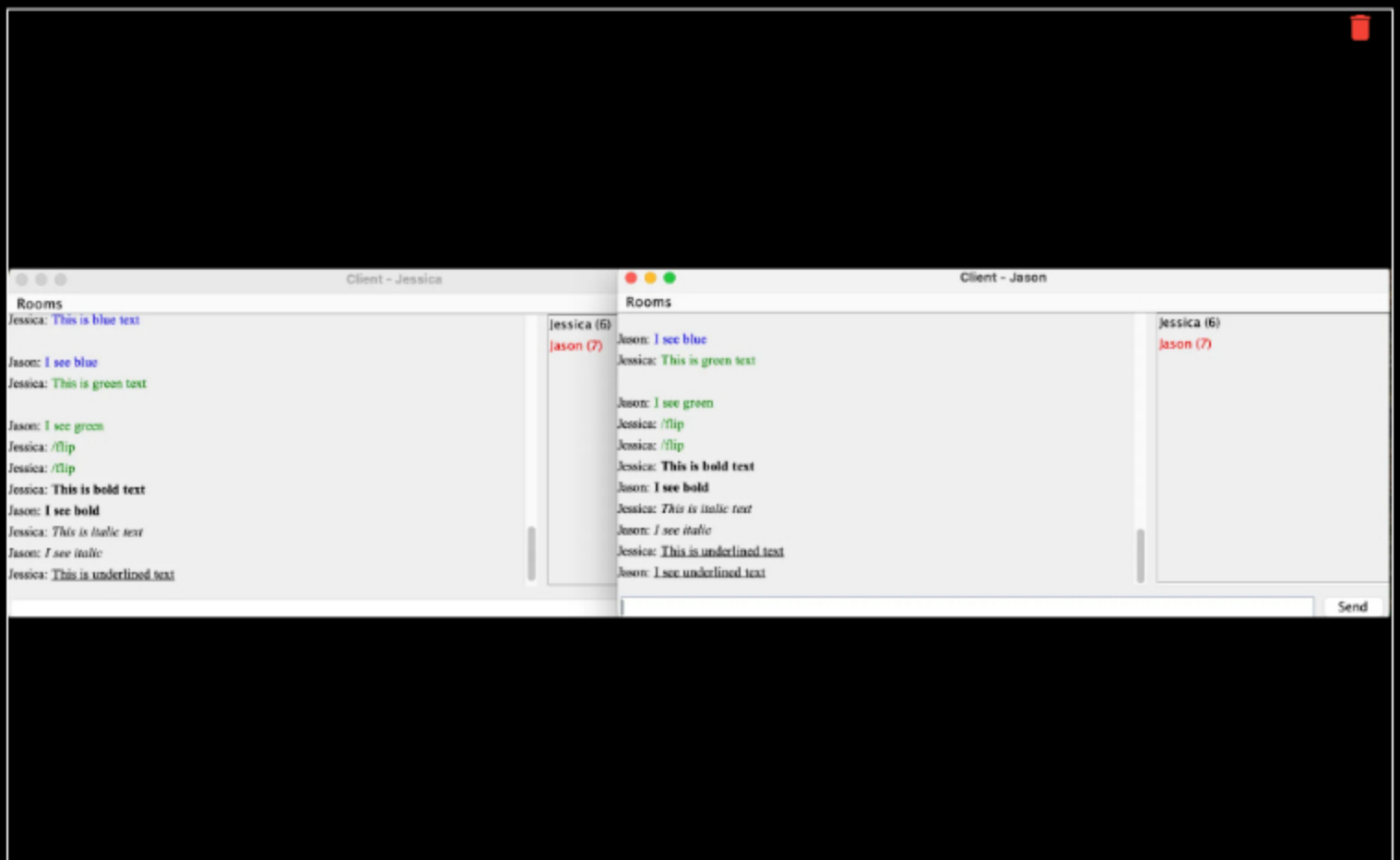
#4 Custom text formatting for red working (Part of the message should appear red)



Bold & Italic working

Checklist Items (1)

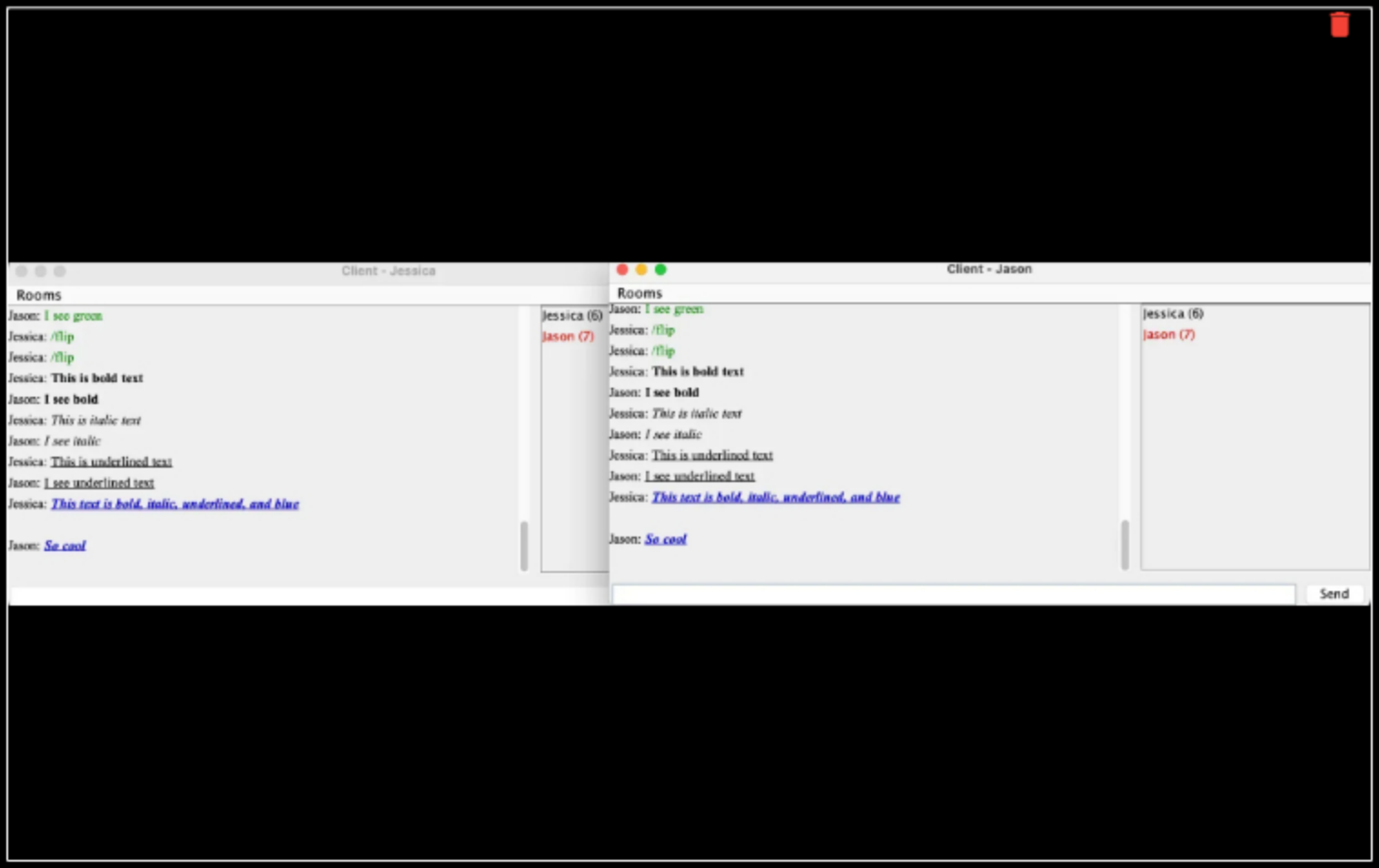
#1 Custom text formatting for bold working (Part of the message should appear bold)



Underline working

Checklist Items (1)

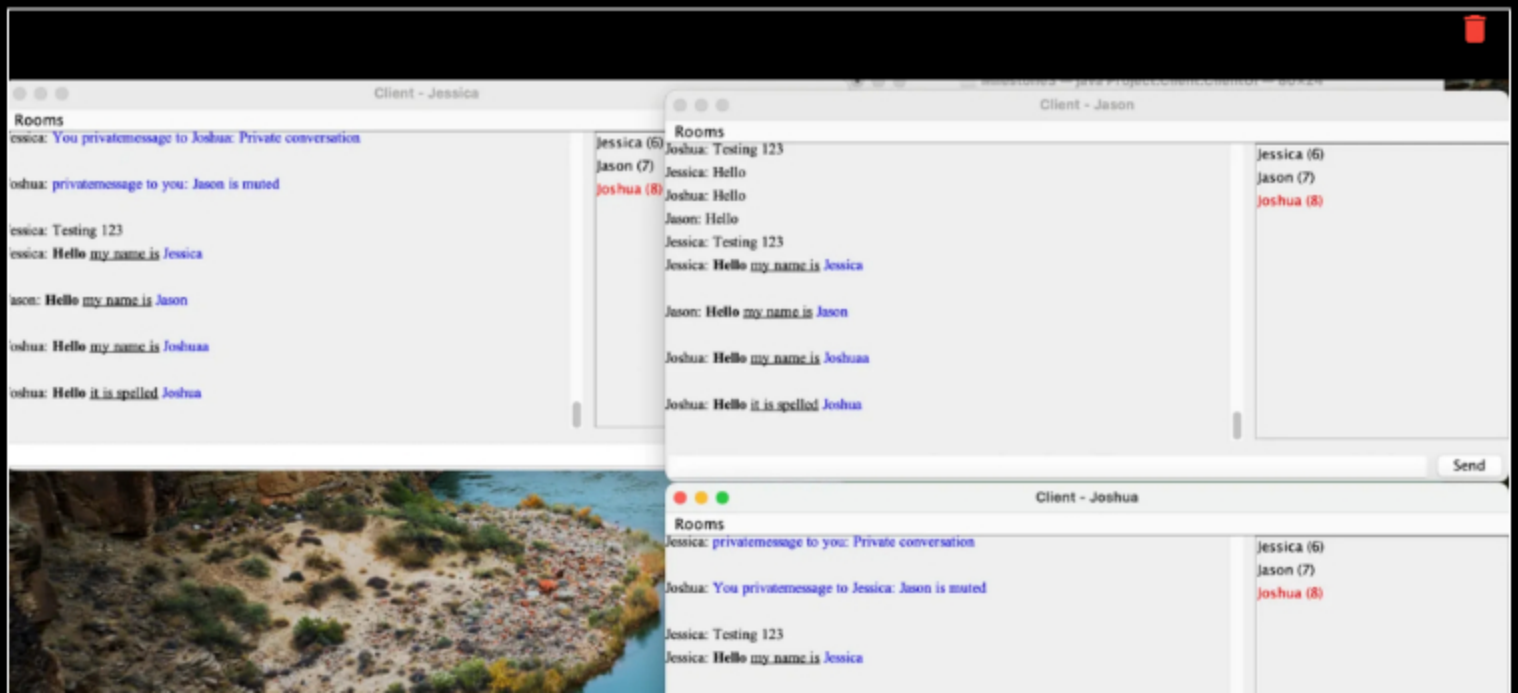
#3 Custom text formatting for underline working (Part of the message should appear underline)



^~!_This text is bold, italic, underlined, and blue!_~^

Checklist Items (1)

#7 Custom text formatting for combined bold, italic, underline, and a color working (Part of the message should have all 4 formats applied at once)



Custom text formatting for combined bold, italic, underline, and a color working

Checklist Items (1)

#7 Custom text formatting for combined bold, italic, underline, and a color working (Part of the message should have all 4 formats applied at once)

COLLAPSE

Task #3 - Points: 1

Text: Screenshot of the code solving the formatting display

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show each relevant file this was done in (may be one or more)
#2	1	Include ucid and date comment
#3	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

Project > Server > Room.java > Room > sendMessage(ServerThread, String)

```
10   public class Room implements AutoCloseable {
243      protected synchronized void sendMessage(ServerThread sender, String message) {
342          //UCID: jeo29
343          //Date: April 13, 2024
344
345          if (message.contains(s:"%")){
346              String[] tEffects = message.split(regex:"");
347              message = "";
348              int count = 0;
349              int count2 = 0;
350              int indexcount = 0;
351              for (int i = 0; i < tEffects.length; i++){
352                  if (tEffects[i].equals(anObject:"%")){
353                      count++;
354                      count2++;
355                      if (count == 1){
356                          indexcount = i;
357                          tEffects[i] = "<font color=\"red\">";      You, 7 minutes ago • Uncommitted
358                      }
359                      if (count == 2){
360                          tEffects[i] = "</font>";
361                          count = 0;
362                      }
363                  } <- #352-363 if (tEffects[i].equals("%"))
364              } <- #351-364 for (int i = 0; i < tEffects.length; i++)
365              if (count2%2 == 1){
366                  tEffects[indexcount] = "%";
367              }
368              for(String i: tEffects){
```

```
370         }
371     } <- #345-371 if (message.contains("%"))
```

Red

Checklist Items (1)

#1 Show each relevant file this was done in (may be one or more)

```
Project > Server > Room.java > Room > sendMessage(ServerThread, String)
10  public class Room implements AutoCloseable {
243     protected synchronized void sendMessage(ServerThread sender, String message) {
373         //UCID: jeo29
374         //Date: April 13, 2024
375
376         if (message.contains(s:"*")){
377             String[] tEffects = message.split(regex:"");
378             message = "";
379             int count = 0;
380             int count2 = 0;
381             int indexcount = 0;
382             for (int i = 0; i < tEffects.length; i++){
383                 if (tEffects[i].equals(anObject:"*")){
384                     count++;
385                     count2++;
386                     if (count == 1){
387                         indexcount = i;
388                         tEffects[i] = "<font color=\"green\">";
389                     }
390                     if (count == 2){
391                         tEffects[i] = "</font>";
392                         count = 0;
393                     }
394                 } <- #383-394 if (tEffects[i].equals("*"))
395             } <- #382-395 for (int i = 0; i < tEffects.length; i++)
396             if (count2%2 == 1){
397                 tEffects[indexcount] = "*";
398             }
399             for(String i: tEffects){
400                 message+= i;
401             }
402         } <- #376-402 if (message.contains("*"))
```

Green

Checklist Items (1)

#1 Show each relevant file this was done in (may be one or more)

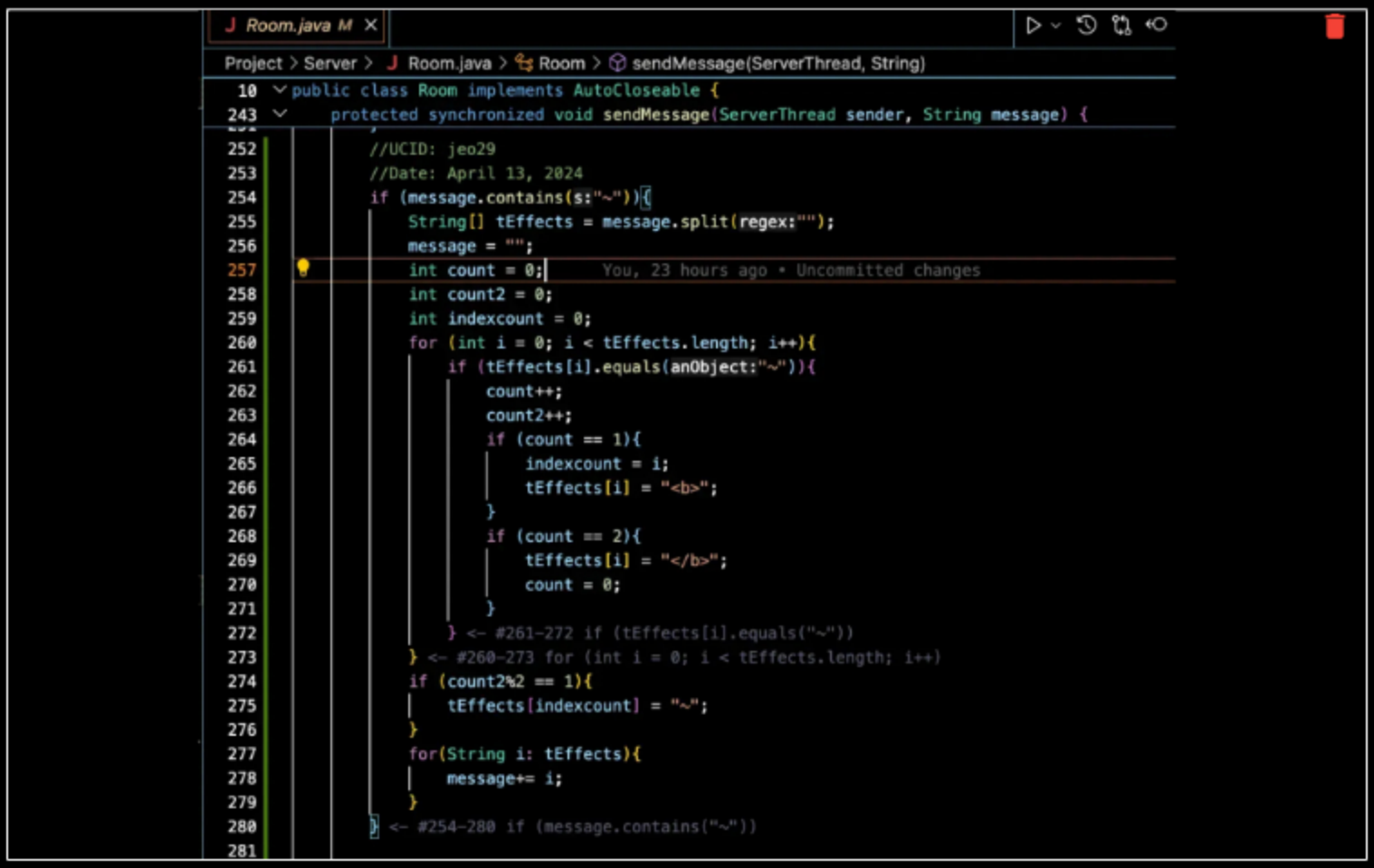
```
403
404         //UCID: jeo29
405         //Date: April 13, 2024
406
407         if (message.contains(s:"^")){
408             String[] tEffects = message.split(regex:"");
409             message = "";
410             int count = 0;
411             int count2 = 0;
412             int indexcount = 0;
413             for (int i = 0; i < tEffects.length; i++){
414                 if (tEffects[i].equals(anObject:"^")){
415                     count++;
416                     count2++;
417                     if (count == 1){
418                         indexcount = i;
419                         tEffects[i] = "<font color=\"blue\">";
420                     }
421                     if (count == 2){
422                         tEffects[i] = "</font>";
423                         count = 0;
```

```
425            } <- #414-425 if (tEffects[i].equals("^"))
426        } <- #413-426 for (int i = 0; i < tEffects.length; i++)
427        if (count2%2 == 1){
428            tEffects[indexcount] = "^";
429        }
430        for(String i: tEffects){
431            message+= i;
432        }
433    } <- #407-433 if (message.contains("^"))
```

Blue

Checklist Items (1)

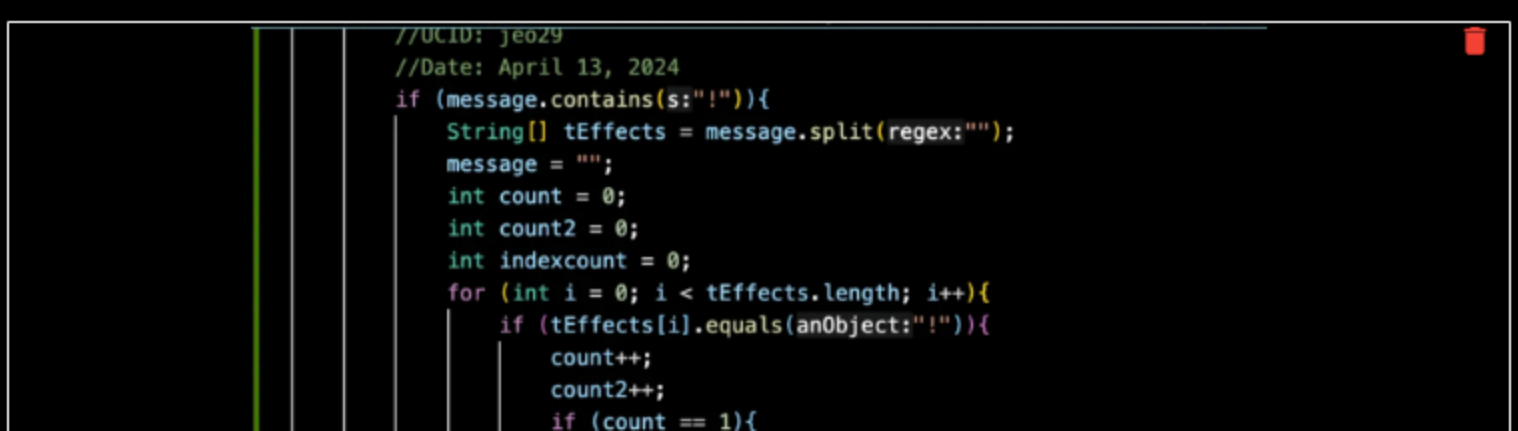
#1 Show each relevant file this was done in (may be one or more)



Bold

Checklist Items (1)

#3 Clearly caption screenshots



```
                indexcount = i;
                tEffects[i] = "<i>";
            }
            if (count == 2){
                tEffects[i] = "</i>";
                count = 0;
            }
        } <- #291-302 if (tEffects[i].equals("!"))
    } <- #290-303 for (int i = 0; i < tEffects.length; i++)
    if (count2%2 == 1){
        tEffects[indexcount] = "!";
    }
    for(String i: tEffects){
        message+= i;
    }
} <- #284-310 if (message.contains("!"))
```

Italics

Checklist Items (1)

#2 Include ucid and date comment

```
311
312     //UCID: jeo29
313     //Date: April 13, 2024
314
315     if (message.contains(s:"_")){
316         String[] tEffects = message.split(regex:"");
317         message = "";
318         int count = 0;
319         int count2 = 0;
320         int indexcount = 0;
321         for (int i = 0; i < tEffects.length; i++){
322             if (tEffects[i].equals(anObject:"_")){
323                 count++;
324                 count2++;
325                 if (count == 1){
326                     indexcount = i;
327                     tEffects[i] = "<u>";
328                 }
329                 if (count == 2){
330                     tEffects[i] = "</u>";
331                     count = 0;
332                 }
333             } <- #322-333 if (tEffects[i].equals("_"))
334         } <- #321-334 for (int i = 0; i < tEffects.length; i++)
335         if (count2%2 == 1){
336             tEffects[indexcount] = "_";
337         }
338         for(String i: tEffects){
339             message+= i;
340         }
341     } <- #315-341 if (message.contains("_"))
```

Underline

Checklist Items (1)

#2 Include ucid and date comment

COLLAPSE

Task #4 - Points: 1

Text: Explain how the formatting was made to be visible/rendered in the UI

Details:
Note each scenario

Response:

The formatting was made to be visible/rendered in the UI by including special characters. The code checks for specific characters (~, !, _, %, *, ^) and replaces them with corresponding HTML tags needed to assist with the formatting process. This is so that the UI is able to identify/pull these special character needed to function properly with its special style bold, italic, underline, or colored text.

Private Message with @ (2 pts.)

^COLLAPSE^

^COLLAPSE^

Task #1 - Points: 1

Text: Screenshots demoing private message

Checklist

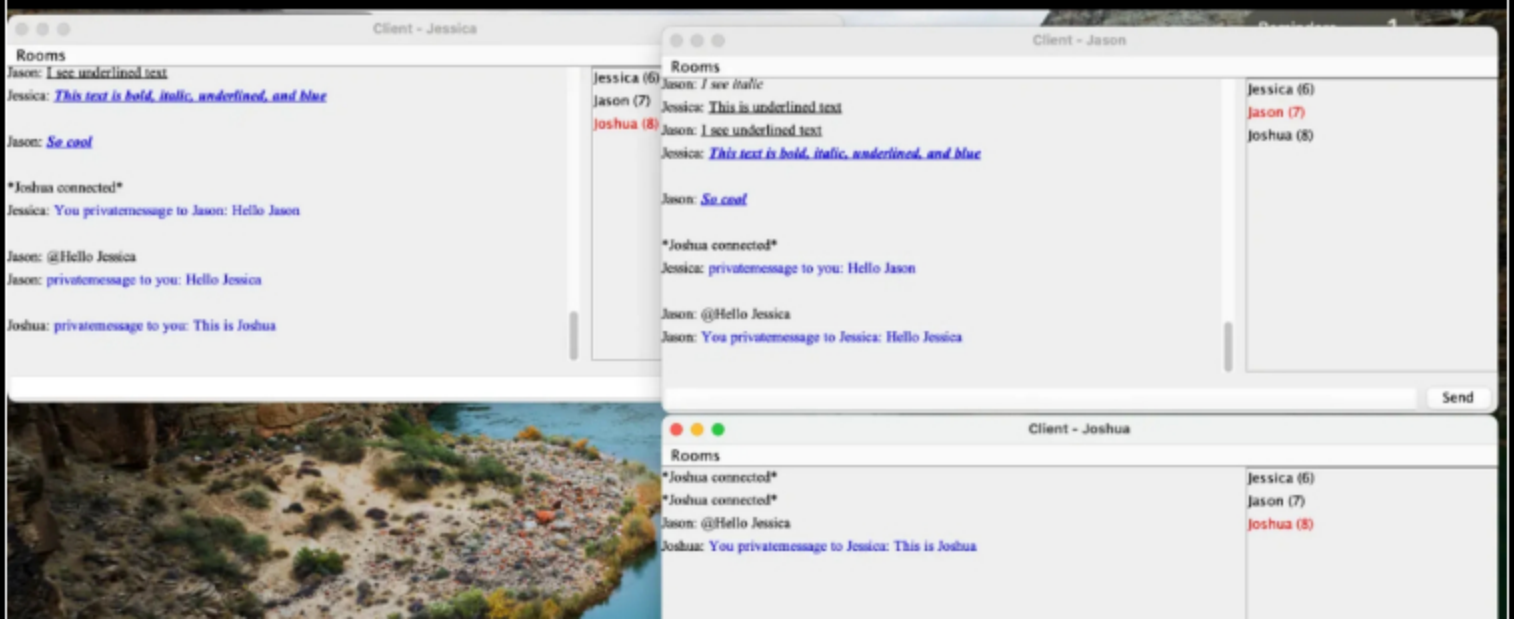
*The checkboxes are for your own tracking

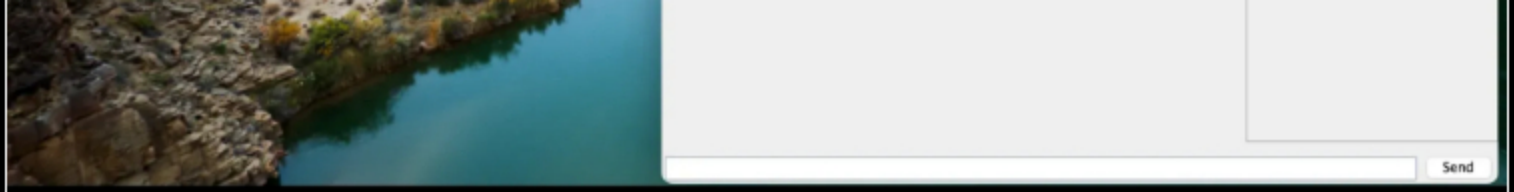
#	Points	Details
#1	1	Should have 3 clients in the same room
#2	1	Demo a private message where only the sender and target see the message
#3	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large





3 clients in the same room Private message where only the sender and target see the message

Checklist Items (1)

#2 Demo a private message where only the sender and target see the message

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the related code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show what code processes and handles the private message
#2	1	The message should only be sent to the receiver and the target
#3	1	The client should be targeting the username and the server side should be fetching the correct recipient
#4	1	Include ucid and date comment
#5	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
435     //UCID: jeo29
436     //Date: April 13, 2024
437     if (message.contains(s:"@")){
438         String[] words = message.split(regex:"\\s+");
439         for(String word : words){
440             if (word.startsWith(prefix:"@")){
441                 String privatemessageName = word.substring(beginIndex:1);
442                 ServerThread targetUser = findUser(privatemessageName);
443
444                 if (targetUser != null){
445                     String senderMessage = "<font color=\"blue\">You privatemessage to " + target
446                     String receiverMessage = "<font color=\"blue\">privatemessage to you: " + mes
447                     targetUser.sendMessage(sender.getClientId(), receiverMessage);
448                     sender.sendMessage(sender.getClientId(), senderMessage);
449                     return;
450                 }  <- #444-450 if (targetUser != null)
451             }      You, 2 seconds ago • Uncommitted changes  <- #440-451 if (word.startsWith("@")
452         } <- #439-452 for(String word : words)
453     } <- #437-453 if (message.contains("@"))
454     long from = sender == null ? Constants.DEFAULT_CLIENT_ID : sender.getClientId();
455     Iterator<ServerThread> iter = clients.iterator();
456     while (iter.hasNext()) {
457         ServerThread client = iter.next();
458         if(client.isMuted(sender.getClientName())){
```

```
459         continue;
460     }
461
462     boolean messageSent = client.sendMessage(from, message);
463     if (!messageSent) {
464         handleDisconnect(iter, client);
465     }
```

code processes and handles the private message

Checklist Items (1)

#1 Show what code processes and handles the private message

```
252 //UCID: jeo29
253 //Date: April 13, 2024
254 if (message.contains(s:"~")){
255     String[] tEffects = message.split(regex:"");
256     message = "";
257     int count = 0;
258     int count2 = 0;
259     int indexcount = 0;
260     for (int i = 0; i < tEffects.length; i++){
261         if (tEffects[i].equals(anObject:"~")){
262             count++;
263             count2++;
264             if (count == 1){
265                 indexcount = i;
266                 tEffects[i] = "<b>";
267             }
268             if (count == 2){
269                 tEffects[i] = "</b>";
270                 count = 0;
271             }
```

The message should only be sent to the receiver and the target

Checklist Items (1)

#2 The message should only be sent to the receiver and the target

```
435     //UCID: jeo29
436     //Date: April 13, 2024
437     if (message.contains(s:"@")){
438         String[] words = message.split(regex:"\\s+");
439         for(String word : words){
440             if (word.startsWith(prefix:"@")){
441                 String privatemessageName = word.substring(beginIndex:1);
442                 ServerThread targetUser = findUser(privatemessageName);
443
444                 if (targetUser != null){
445                     String senderMessage = "<font color=\"blue\">You privatemessage to " + targ
446                     String receiverMessage = "<font color=\"blue\">privatemessage to you: " + m
```

```
447                     targetUser.sendMessage(sender.getClientId(), receiverMessage);
448                     sender.sendMessage(sender.getClientId(), senderMessage);
449                     return;
450                 }  <- #444-450 if (targetUser != null)
451             } <- #440-451 if (word.startsWith("@"))
452         } <- #439-452 for(String word : words)
453     } <- #437-453 if (message.contains("@"))
454     long from = sender == null ? Constants.DEFAULT_CLIENT_ID : sender.getClientId();
455     Iterator<ServerThread> iter = clients.iterator();
456     while (iter.hasNext()) {
457         ServerThread client = iter.next();
458         if(client.isMuted(sender.getClientName())){
```

The client should be targeting the username and the server side should be fetching the correct recipient

Checklist Items (1)

#3 The client should be targeting the username and the server side should be fetching the correct recipient

COLLAPSE

Task #3 - Points: 1

Text: Explain how private message works related to the code above

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Include how the sender and receiver are handled
#2	1	Include how the username is used to get the proper id

Response:

The sender and the receiver are handled by targeting the user by it's username. Example @(insert username) then retrieving the persons username that followed after the @ then handling the message being sent to them privately.

Mute/Unmute Users (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing feature working

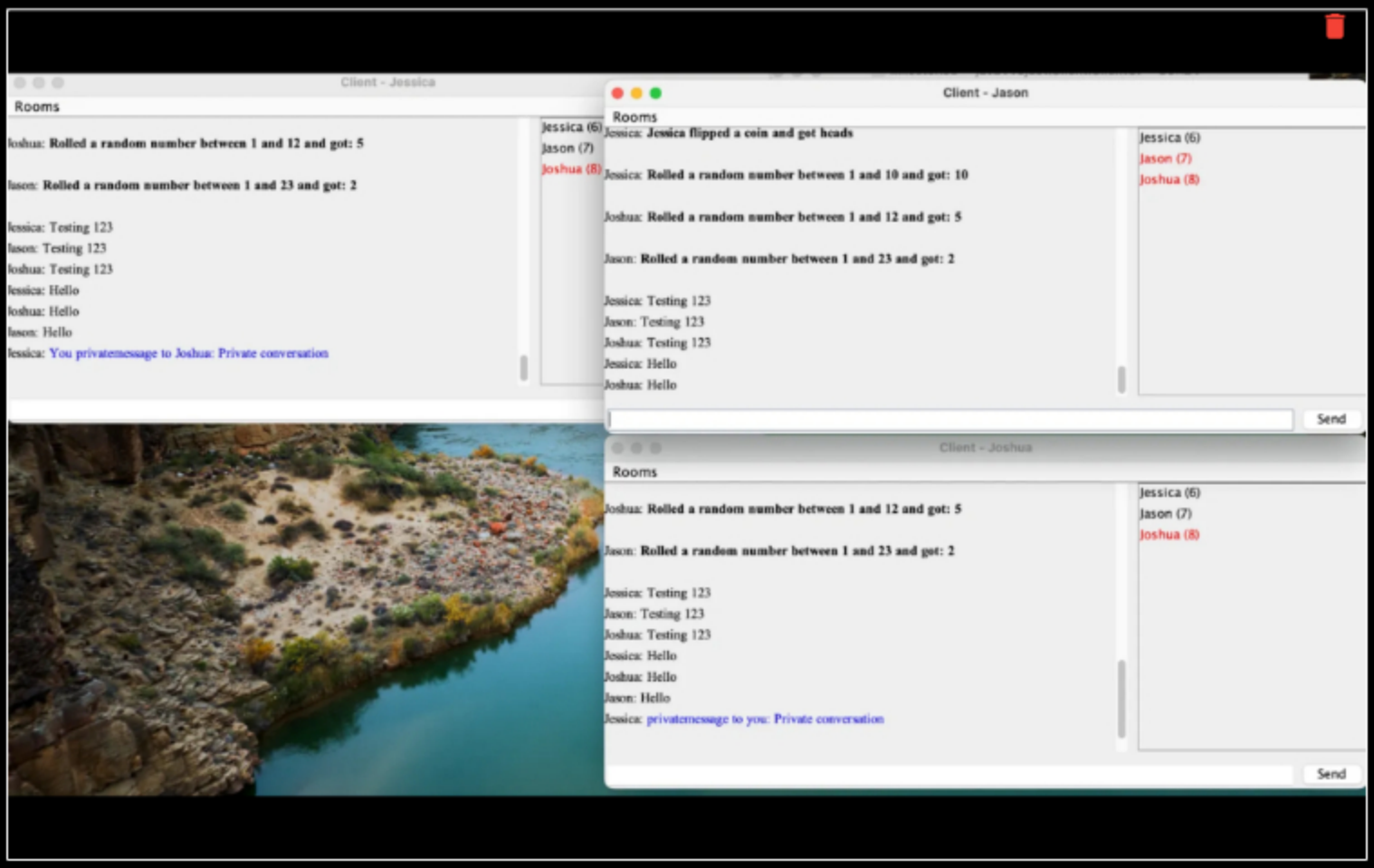
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Should have 3 clients in the same room
#2	1	Demo mute preventing messages between the muter and the target
#3	1	Demo mute also being accounted for with private messages
#4	1	Demo unmute allowing the messages again from the target to the unmuter

Task Screenshots:

Gallery Style: Large View

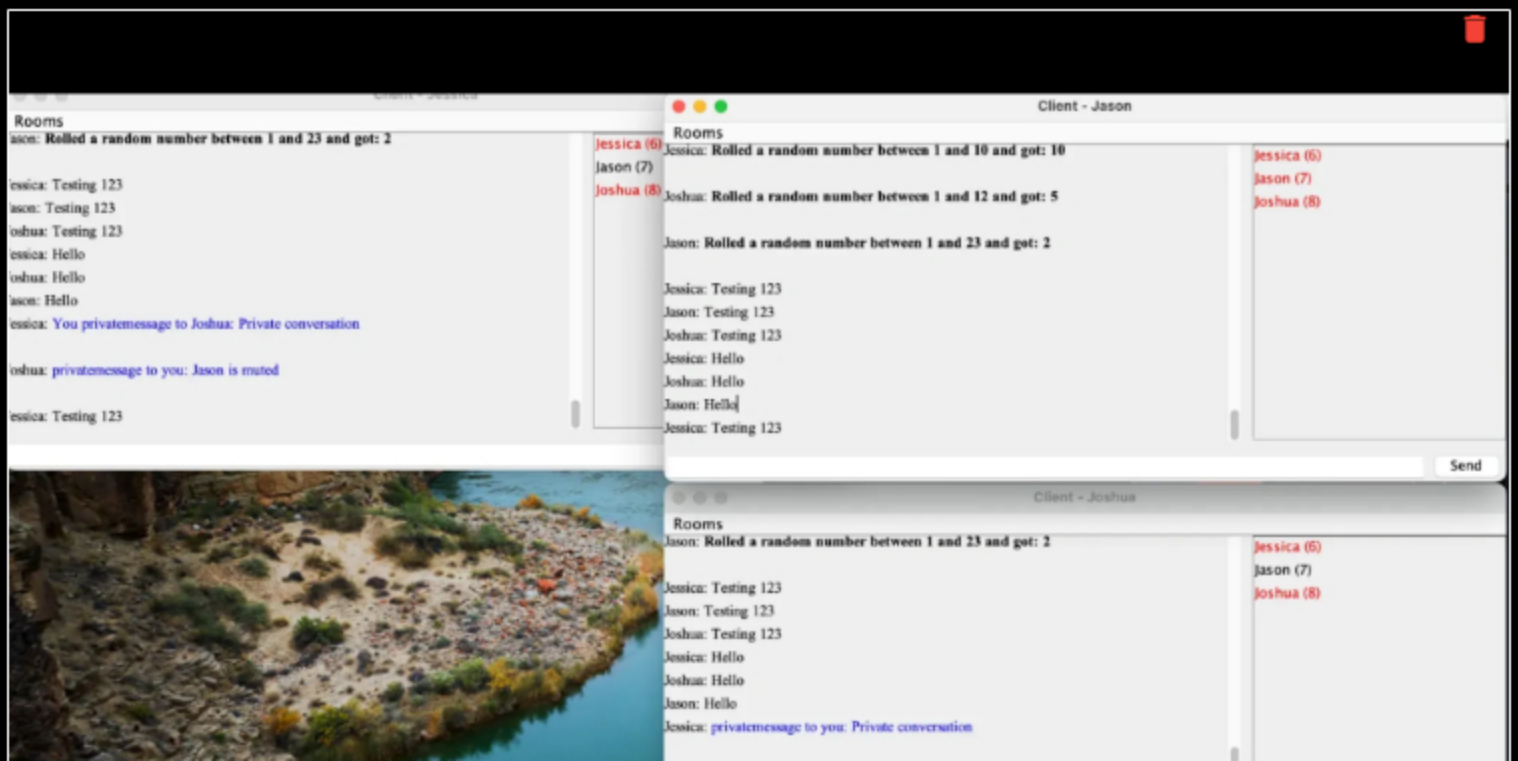
Small Medium Large



Demo mute

Checklist Items (1)

#2 Demo mute preventing messages between the muter and the target



Demo unMute

Checklist Items (1)

#4 Demo unmute allowing the messages again from the target to the unmuter

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the related code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	ServerThread should have a list of who they muted
#2	1	ServerThread should expose and add, remove, and is muted check to room
#3	1	Room should handle the mute list when receiving the appropriate payloads
#4	1	Room should check the mute list during send message and private messages
#5	1	Include ucid and date comment
#6	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
//UCID: jeo29
//DATE: April 13, 2024
public boolean sendMuteUser(String name){
    Payload p = new Payload();
    p.setPayloadType(PayloadType.MUTE);
    p.setClientName(name);
    return send(p);
} <- #33-38 public boolean sendMuteUser(String name)
public boolean sendUnmuteUser(String name){
    Payload p = new Payload();
    p.setPayloadType(PayloadType.UNMUTE);
    p.setClientName(name);
    return send(p);
} <- #39-44 public boolean sendUnmuteUser(String name)
public boolean isMuted(String name){
    for(String i: muteList){
        if(i.equals(name)){
            return true;
        }
    } <- #46-50 for(String i: muteList)
    return false;
} <- #45-52 public boolean isMuted(String name)
```

```
    public void setClientId(long id) {
        myId = id;
    }
```

sendMuteUser

Checklist Items (1)

#1 ServerThread should have a list of who they muted

```
    public boolean isMuted(String name){
        for(String i: muteList){
            if(i.equals(name)){
                return true;
            }
        } <- #46-50 for(String i: muteList)
        return false;
```

isMuted

Checklist Items (1)

#1 ServerThread should have a list of who they muted

```
                //UCID: jeo29
                //DATE: April 13, 2024
                case MUTE:
                        muteList.add(p.getClientName());
                        sendMuteUser(p.getClientName());
                    break;
                case UNMUTE:
                        muteList.remove(p.getClientName());
```

```
                sendUnmuteUser(p.getClientName());
            break;
        default:
            break;
```

Mute/Unmute

Checklist Items (1)

#1 ServerThread should have a list of who they muted

COLLAPSE

Task #3 - Points: 1

Text: Explain how the mute and unmute logic works in relation to the code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Explain how your mute list is handled
#2	1	Explain how it's handled/processed in send message and private message

Response:

The mute list is handled by three methods in the ServerThread.

- isMuted
- sendUnMuteUser
- sendMuteUser

It is handled/processed in send message and private message.

- The sendMuteUser method adds a client's name to the server-side mute list and sends a payload to the client to inform them that they have been muted.
- The sendUnmuteUser method removes a client's name from the mute list and sends a payload to the client to inform them that they have been unmuted.
- Mute/Unmute Process
- isMuted method checks if the client in the Chatroom is on the mute list, if they are it will respond true, if not of course the opposite will occur, which is false.
- isMuted if the sender is muted by the recipient the message also won't send.

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/18

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

My first issue of several was that I was not transferring the files properly. I was missing a few files. This was making it difficult for me to be able to even properly continue. I went ahead and deleted all the files and then re-downloaded the zip. I also then was having trouble in which I had to rename the folders from lowercase to uppercase and vice versa to continue with the layout I had already started. Fixing package names and imports is key. Overall, asking questions to professor and messaging him throughout was helpful.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

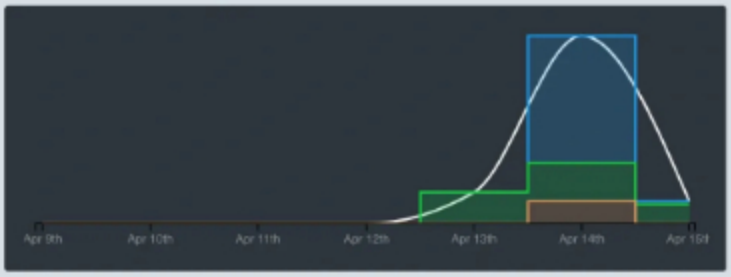
Details:
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved.

Task Screenshots:

Gallery Style: Large View

Small Medium Large

6 hrs 17 mins over the Last 7 Days.



WakaTime

End of Assignment