Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-chatroom-milestone-3-2024/grade/ob75

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

Submissions:

Submission Selection

1 Submission [active] 4/16/2024 3:25:07 PM

Instructions

COLLAPSE

Implement the Milestone 3 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145Xi
Make sure you add your ucid/date as code comments where code changes are done
All code changes should reach the Milestone3 branch
Create a pull request from Milestone3 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: Milestone3

Tasks: 14 Points: 10.00

Basic UI (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the following

Checklist

*The checkboxes are for your own tracking

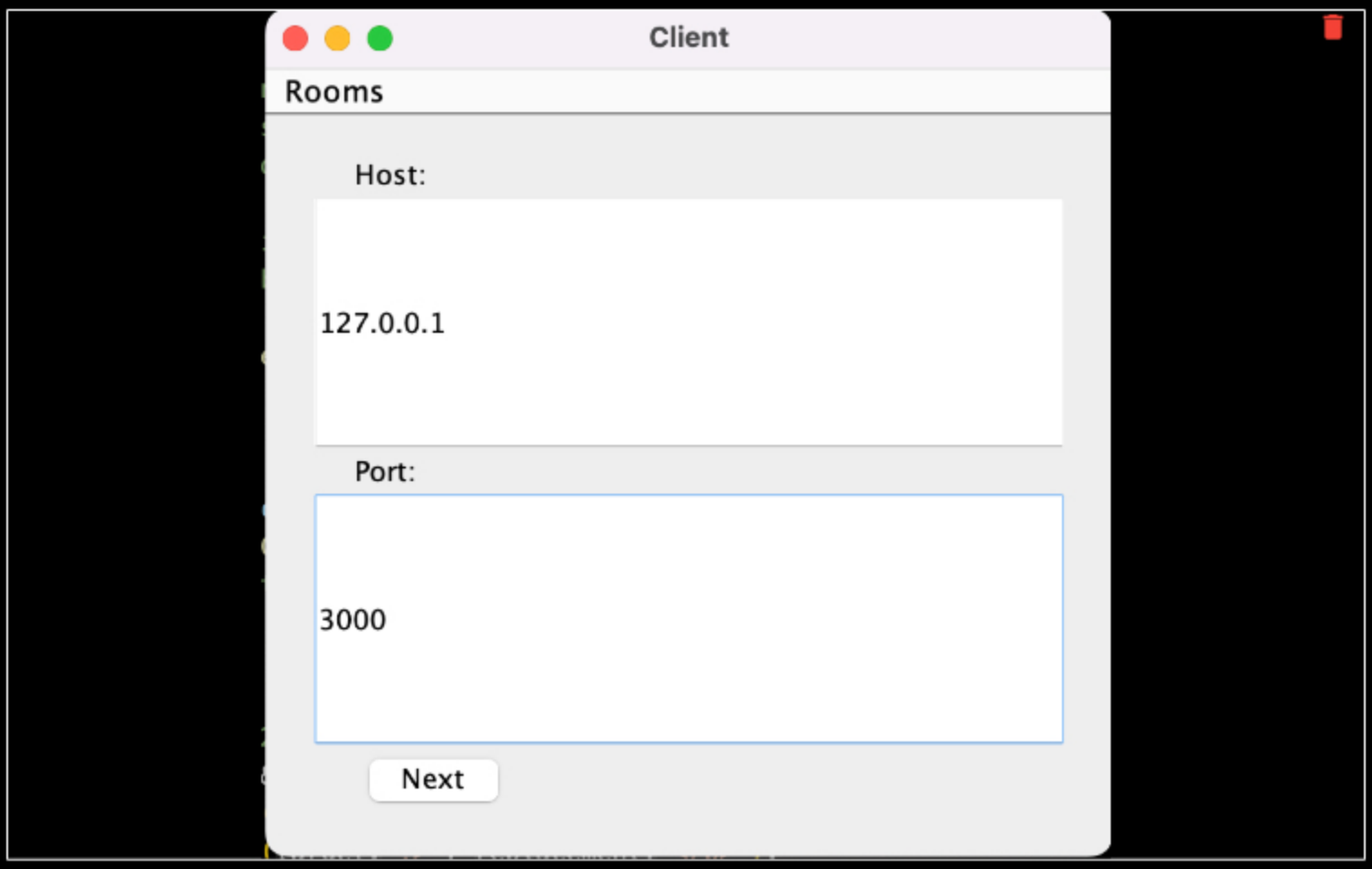
#	Points	Details

#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

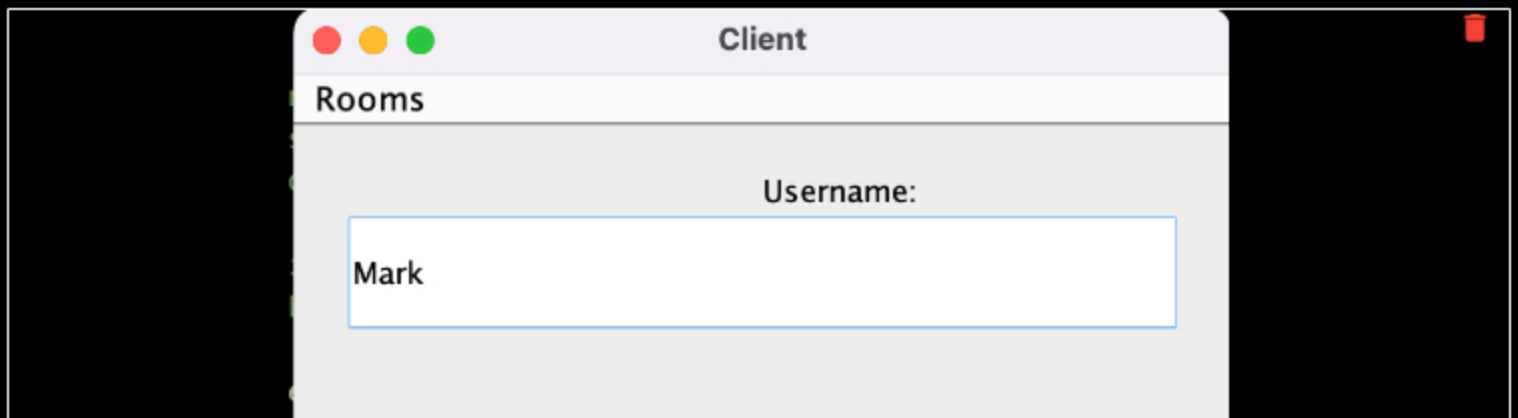


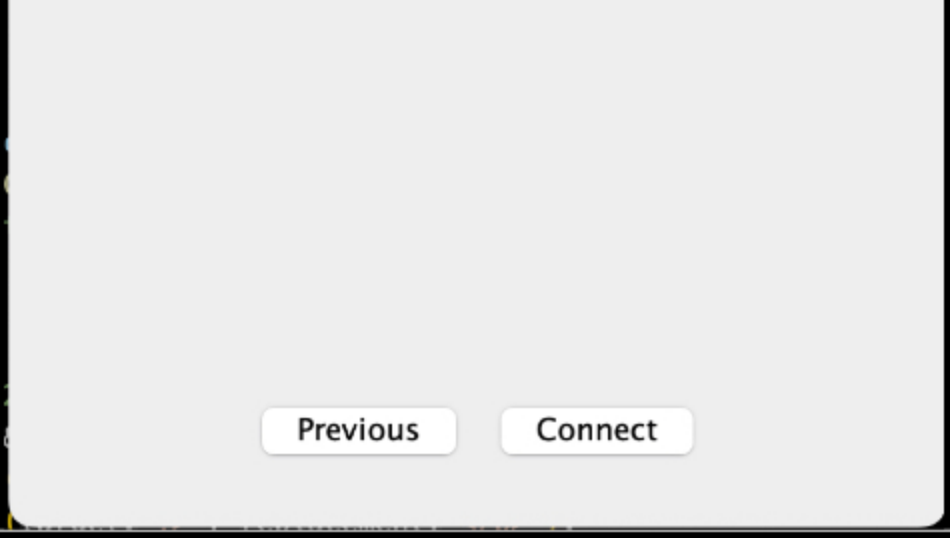
Connection panel startup for client.

Checklist Items (2)

#1 Connection Panel

#4 Clearly caption screenshots



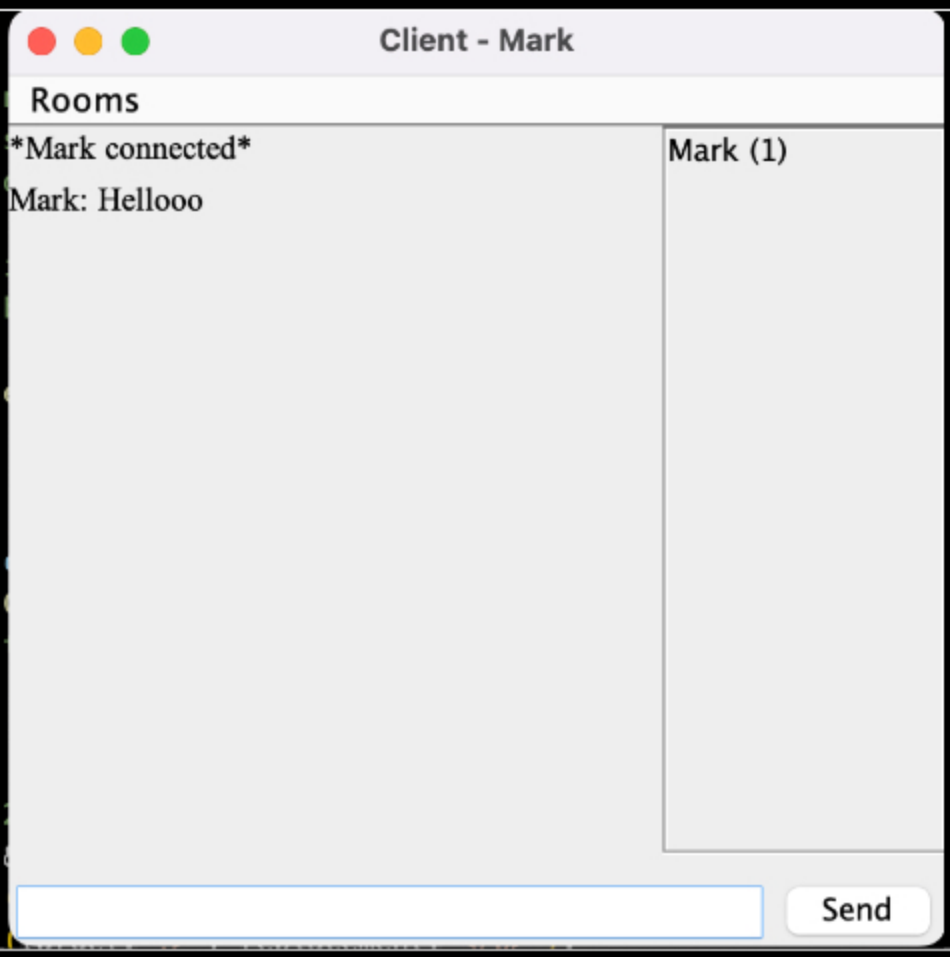


Username panel asking client for name input.

Checklist Items (2)

#2 User Details Panel

#4 Clearly caption screenshots



Chat Panel for client to message room and anyone in room.

Checklist Items (2)

#3 Chat Panel

#4 Clearly caption screenshots

Formatting (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing flip and roll commands

Checklist

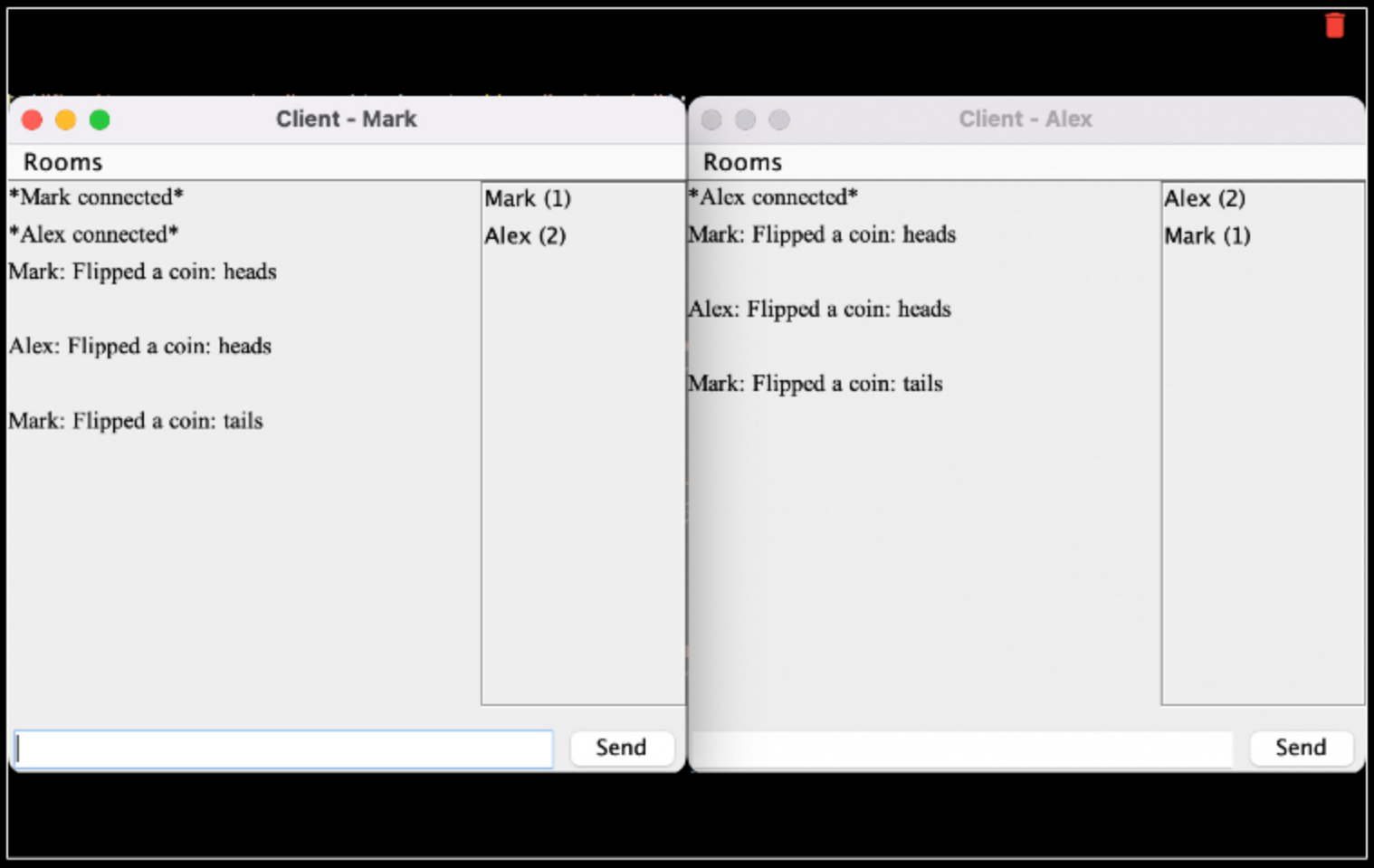
*The checkboxes are for your own tracking

#	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

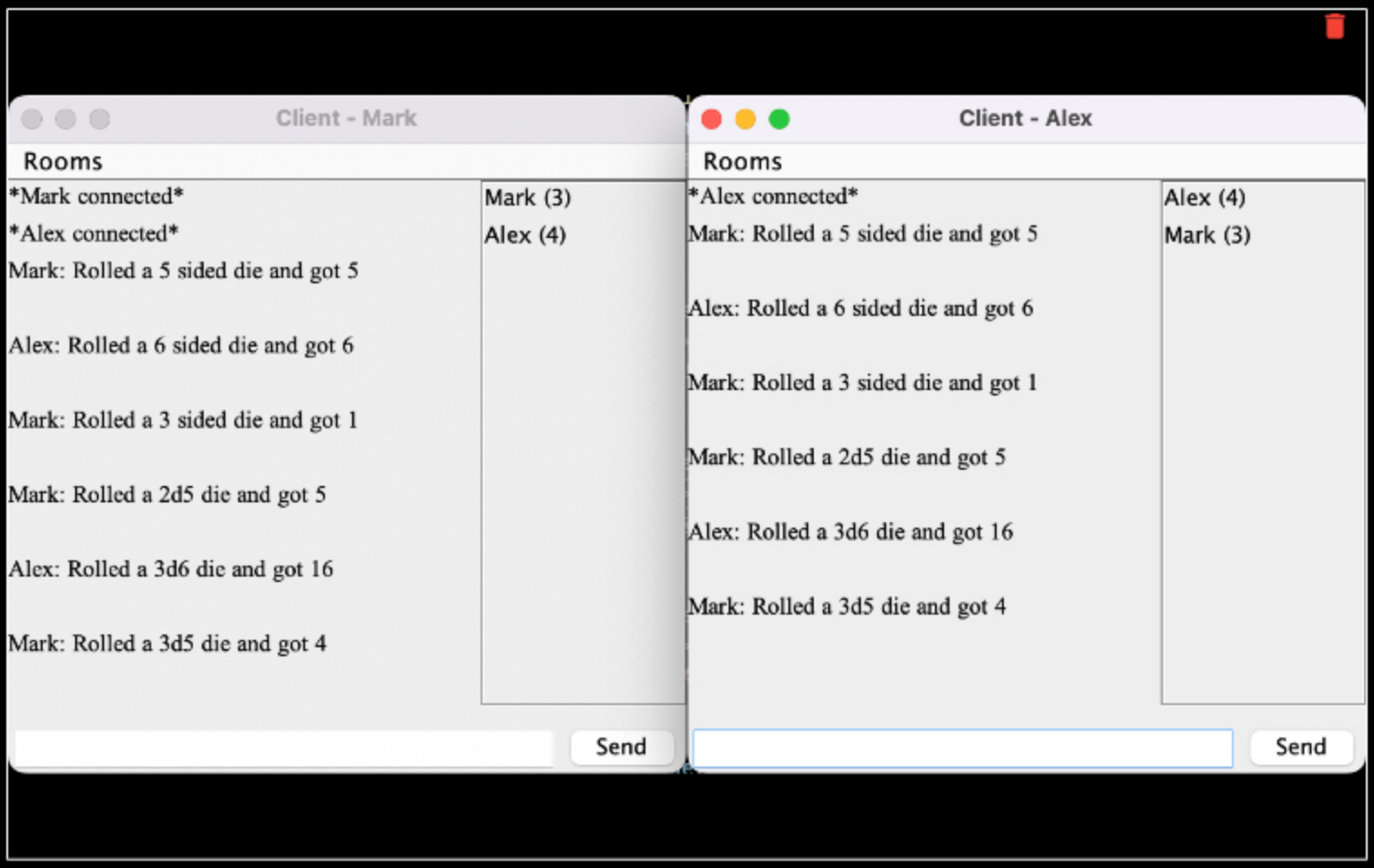


/flip command is done with both client and output of the client is shown as "Flipped a coin:" and result being "tails or heads".

Checklist Items (2)

#1 Flip output in a different format than normal messages

#4 Clearly caption screenshots



/roll command working for both versions of roll, /roll # and /roll #. Shows the amount of dice and sides they rolled, if they did #d#, and the result of that to all clients. Both clients are shown to do both commands of roll.

Checklist Items (3)

#2 Roll # output in a different format than normal messages

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

#4 Clearly caption screenshots

COLLAPSE

Task #2 - Points: 1

Text: Screenshots demoing custom text formatting

Checklist

*The checkboxes are for your own tracking

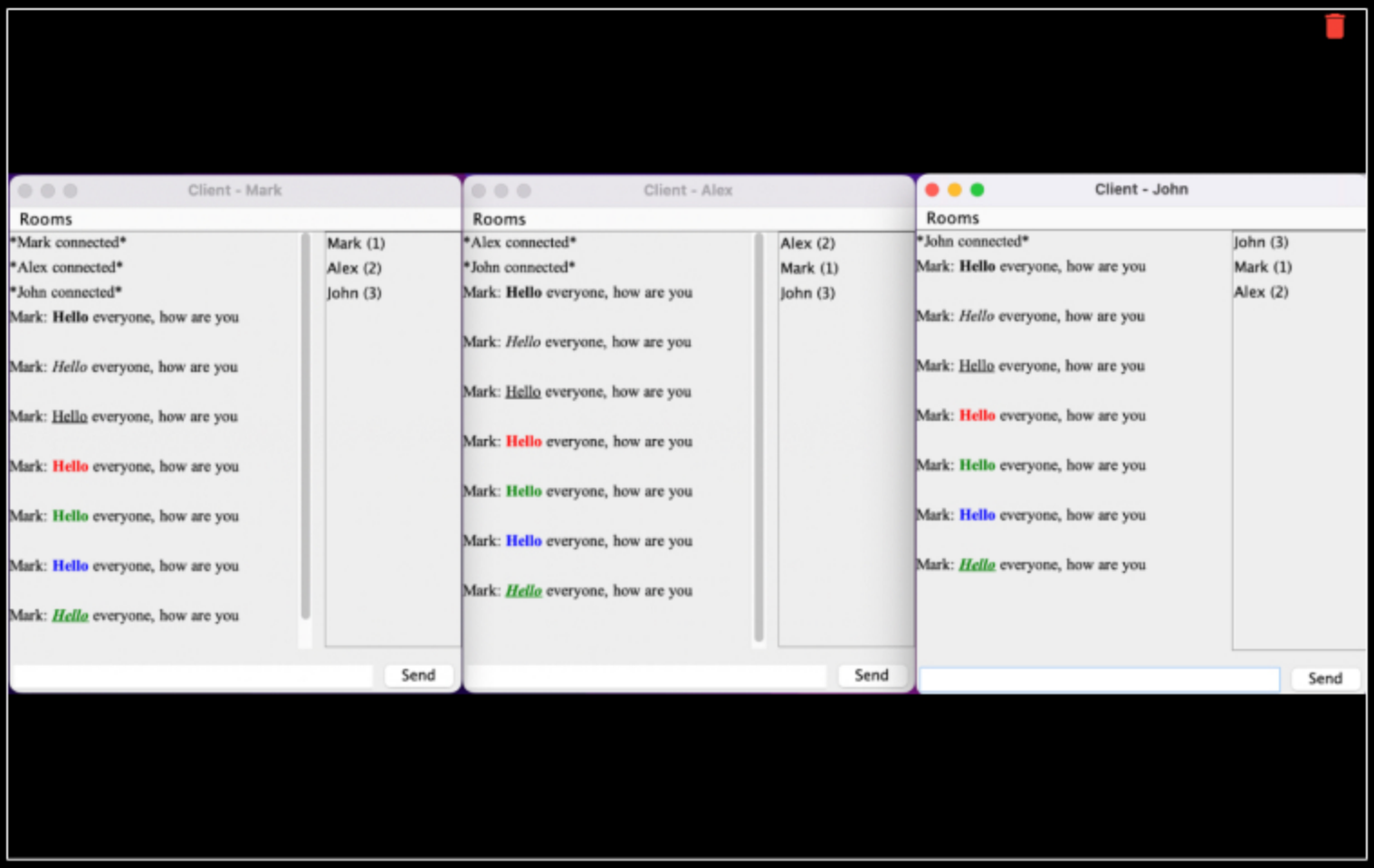
#	Points	Details
#1	1	Custom text formatting for bold working (Part of the message should appear bold)
#2	1	Custom text formatting for italic working (Part of the message should appear italic)
#3	1	Custom text formatting for underline working (Part of the message should appear

		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



Shows the word "Hello" being changed according to what command is placed for them. Bold is first message, italics second, underline third, red fourth, green fifth, and blue sixth. Last one is all text formatting combined and color chosen is green.

Checklist Items (8)

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

#2 Custom text formatting for italic working (Part of the message should appear italic)

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

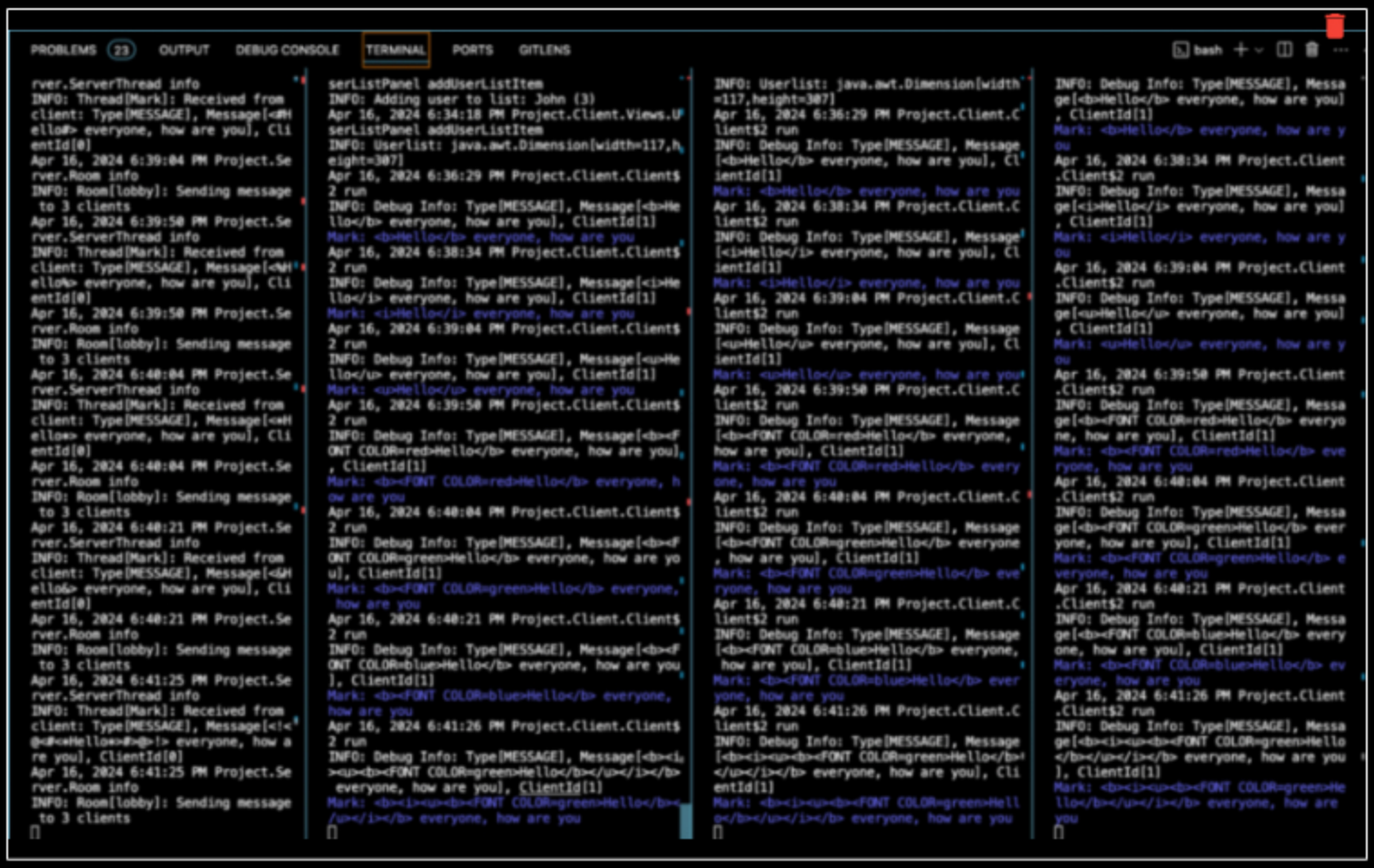
#4 Custom text formatting for red working (Part of the message should appear red)

#5 Custom text formatting for blue working (Part of the message should appear blue)

#6 Custom text formatting for green working (Part of the message should appear green)

#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)

#8 Clearly caption screenshots



Terminal showing raw output of formatting the messages in case the previous screenshot with chat UI doesn't clearly show.

Checklist Items (8)

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

#2 Custom text formatting for italic working (Part of the message should appear italic)

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

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

#5 Custom text formatting for blue working (Part of the message should appear blue)

#6 Custom text formatting for green working (Part of the message should appear green)

#7 Custom text formatting for combined bold, italic, underline, and a color working (Part of the message should

#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)

#8 Clearly caption screenshots

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

```
158     public void addText(String text) {
159         JPanel content = chatArea;
160         // add message
161
162         //UCID - ob75 - April 16, 2024
163         JEditorPane textContainer = new JEditorPane(type:"text/html", text);      You, 25 minutes ago • Uncommitted changes
164
```

Changed from new JEditorPane("text/plain", text); to new JEditorPane("text/html", text); on line 163 under UCID.

Checklist Items (3)

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

#2 Include ucid and date comment

#3 Clearly caption screenshots

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

```
11   public class Room implements AutoCloseable {
244      protected synchronized void sendMessage(ServerThread sender, String message) {
269
270          //UCID - ob75 - April 16, 2024
271          if (message.contains(s:"<%") && message.contains(s:"%>")) {
272              message = message.replace(target:"<%", replacement:"<FONT COLOR=red>");
273              message = message.replace(target:"%>", replacement:"</FONT COLOR=red>");
274          }
275
276          if (message.contains(s:"<+") && message.contains(s:"+>")) {
277              message = message.replace(target:"<+", replacement:"<FONT COLOR=green>");
278              message = message.replace(target:"+>", replacement:"</FONT COLOR=green>");
279          }
280
281          if (message.contains(s:"<&") && message.contains(s:"&>")) {
282              message = message.replace(target:"<&", replacement:"<FONT COLOR=blue>");
283              message = message.replace(target:"&>", replacement:"</FONT COLOR=blue>");
284          }
285
```

Changed formatting code for colors to appear on specified message in Chat UI when command is entered for desired color. From line 271 - 284.

Checklist Items (3)

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

#2 Include ucid and date comment

#3 Clearly caption screenshots

Project > Client > Client.java > Client > sendMessage(String)

```
26   public enum Client {
323      public void sendMessage(String message) throws IOException {
324
325          // UCID - ob75 - April 9,2024
326          if (message.startsWith(prefix:"/") && processClientCommand(message)) {
327              return;
```

328

This processes commands from client and formats it to appear in the UI chat so clients can still do commands in chat.

Checklist Items (3)

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

#2 Include ucid and date comment

#3 Clearly caption screenshots

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 way formatting was made visible to the UI was just simply changing the JEditorPane textContainer, in the ChatPanel.java class inside the addText method, from taking ("text/plain", text) to changing it to take ("text/html", text) to allow html tags to work in the UI. Doing this allowed for what only appeared in the text as just tags behind and in front of the specified message to actually having the format of bold, italics, underline and color between red, green, and blue working in chat and no longer having the tags. For the colors to change as well, there was changes made to the code in the sendMessage method inside the Room.java class for tag to change the font color to specified color. Before the replacement tags for the color would just be for example "<R>" "</R>" and now it was changed to "<FONT COLOR = red>" since this is the html tag for setting the text to a certain color. For /flip and /roll, this was done in the client side in the sendMessage method so that clients in the chat in the UI can still use commands in the chat like /flip and /roll and still get the results of their commands.

Private Message with @ (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing private message

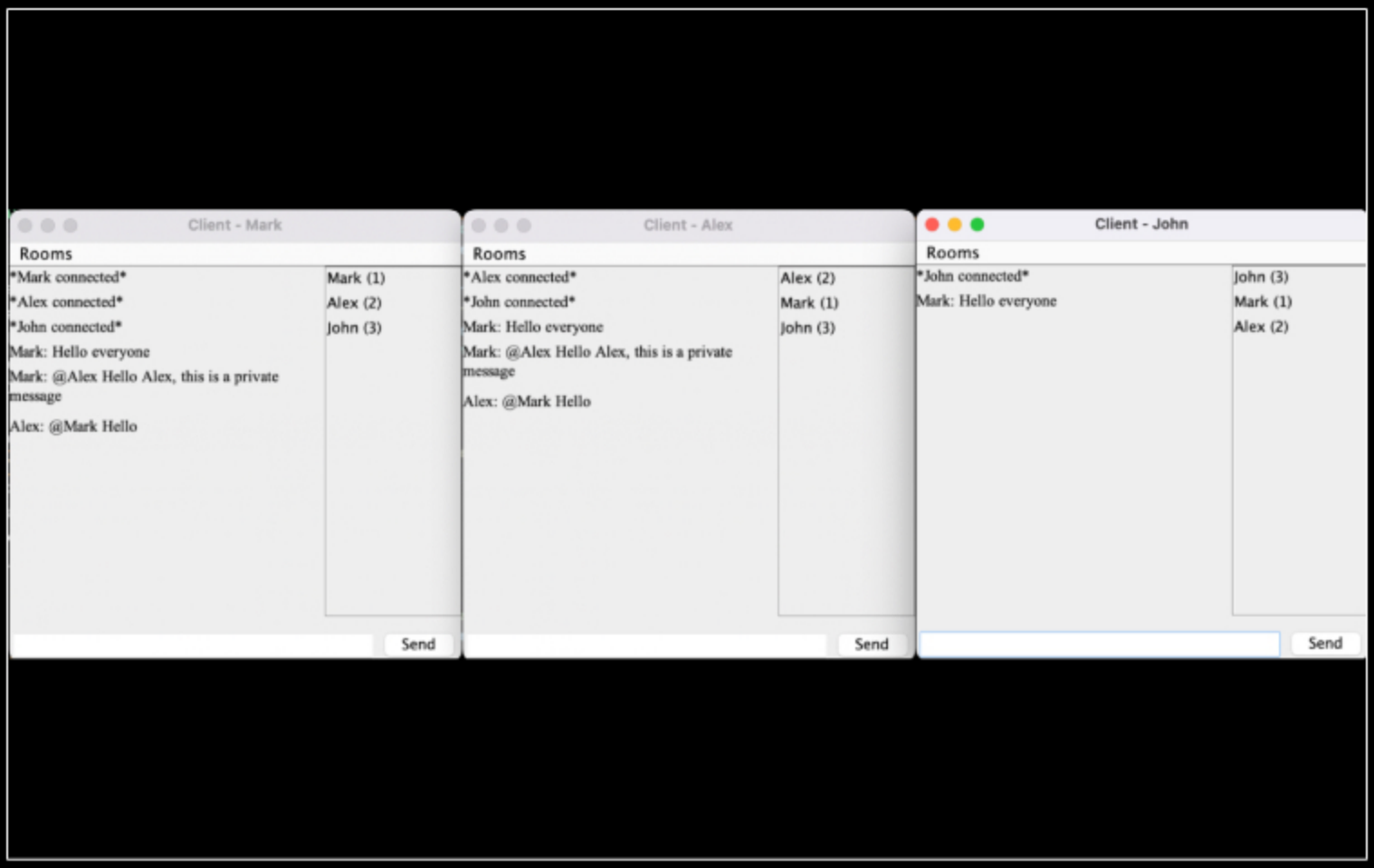
Checklist *The checkboxes are for your own tracking

#	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



Clients are able to send messages to each other privately through @ and then the client they want to message like @Alex. Only target client will see message and other clients in room will not receive that message.

Checklist Items (3)

#1 Should have 3 clients in the same room

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

#3 Clearly caption screenshots

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

Project > Common > PrivateMPayload.java > PrivateMPayload

```java
package Project.Common;

//UCID - ob75 - April 10, 2024

public class PrivateMPayload extends Payload{
    private String clientName;

    public PrivateMPayload(String clientName2, String message2){
        this.clientName = clientName2;
        setMessage(message2);
        setPayloadType(PayloadType.PM);
    }

    public String getClientName() {
        return clientName;
    }

    public void setClientName(String clientName) {
        this.clientName = clientName;
    }


}
```

New Payload class for private messaging and takes in clientName to send out to server. UCID comment at top.

Checklist Items (3)

#1 Show what code processes and handles the private message

#4 Include ucid and date comment

#5 Clearly caption screenshots

```
323     public void sendMessage(String message) throws IOException {
324
325         // UCID - ob75 - April 9,2024
326         if (message.startsWith(prefix:"/") && processClientCommand(message)) {
327             return;
328
329             // UCID - ob75 - April 9,2024
330         } else if (message.startsWith(prefix:"@")) {
331             String privateMessage = message.replace(target:"@", replacement:"").trim();
332             if (privateMessage.contains(s:" ")) {
333                 String[] vals1 = privateMessage.split(regex:" ", limit:2);
334                 if (vals1.length >= 2) {
335                     try {
336                         String clientName = vals1[0];
337                         sendPrivateM(clientName, message);
338
339                     } catch (IOException e) {
340                         System.out.println(TextFX.colorize(text:"Socket error", Color.RED));
341                     } catch (Exception e) {
342                         e.printStackTrace();
343                     }
344                 }
345             }
346             return;
347         }
```

Starts at line 330 under UCID comment and ends line 347. Else if statement checking for @ and performing action of splitting message and sending data.

Checklist Items (4)

#1 Show what code processes and handles the private message

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

#4 Include ucid and date comment

#5 Clearly caption screenshots

```
388     //UCID - ob75 - April 10, 2024
389     public void sendPrivateM(String clientName, String message) throws IOException
390     {
391         PrivateMPayload pm = new PrivateMPayload(clientName, message);
392             pm.setPayloadType(PayloadType.PM);
393             pm.setMessage(message);
394             out.writeObject(pm);
395     }
396
```

Line 389 under UCID up until line 395. Sends data from client to server thread side.

Checklist Items (4)

#1 Show what code processes and handles the private message

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

#4 Include ucid and date comment

#5 Clearly caption screenshots

```
293 // UCID - cb75 - April 10, 2024
294 case PM:
295     try {
296         PrivateMPayload pm = (PrivateMPayload) p;
297         (currentRoom).setPrivateM(this, pm.getClientName(), pm.getMessage());
298     } catch (Exception e) {
299         e.printStackTrace();
300     }
301     break;
302
```

Starts under UCID, line 294 - 301. Server Thread side receiving payload and passing information to room side.

Checklist Items (3)

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

#4 Include ucid and date comment

#5 Clearly caption screenshots

```
83      // UCID - ob75 - April 10, 2024
84      public void setPrivateM(ServerThread client, String clientName, String message) {
85          Iterator<ServerThread> iter = clients.iterator();
86          while (iter.hasNext()) {
87              ServerThread st = iter.next();
88              if (client != null && client.checkMutedList(st.getClientName()) == false
89                      && (st.checkMutedList(client.getClientName()) == false)) {
90              if (st.getClientName().equalsIgnoreCase(clientName)) {
91                  st.sendMessage(client.getClientId(), message);
92
93              }
94              }
95
96              }client.sendMessage(client.getClientId(), message);
97          }
98
99
```

Under UCID comment, starts line 84 - 97. Handles sending message only to client and the target client and checks for a match of the target with the name. Line 88-89 is code requirement for mute/unmute for private message but left in screenshot.

Checklist Items (5)

#1 Show what code processes and handles the private message

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

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

#4 Include ucid and date comment

#5 Clearly caption screenshots

Project > Common > PayloadType.java > PayloadType > UNMUTE

You, 4 days ago | 2 authors (Omar Barrera and others)

```
1  package Project.Common;
2
3  public enum PayloadType {
4      CONNECT, DISCONNECT, MESSAGE, CREATE_ROOM, JOIN_ROOM, LIST_ROOMS, CLIENT_ID, SYNC_CLIENT, ROLL, FLIP //UCID - ob75 - March 30,2024
5      , PM, MUTE, UNMUTE //UCID - ab75 - April 10, 2024      You, 4 days ago • Uncommitted changes
6  }
```

New payload type named PM for private messaging right next to MUTE. UCID comment on same line.

Checklist Items (2)

#4 Include ucid and date comment

#5 Clearly caption screenshots

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:

For the private message, first thing that was done was creating a payload type named PM that would be used for later in the server thread side. An else if statement is then created inside the sendMessage method in the client.java side where it would check for if the message starts with the @ and followed by name and then a message after the name. The target client name and message after is separated so that the client name can be checked individually if they're in the chat and the message wouldn't interfere with the name check. Then it sends the client name and message via payload and using a new sendPrivateM method which creates and sends out the payload to the server side. A separate payload class called PrivateMPayload was created to set the new values to the target client name and set the message to send these out. Server thread side then receives the payload from the client side using the case PM as mentioned before to get the payload of private message to then send the values over to the Room side where the logic is then handled to send the message to the desired client. Here in the Room side, a new method named setPrivateM is created that takes in the client sending the message, the target client, and the message which was all passed from the server thread. Here an iterator is created to loop through the list of clients in the chat and checks first to see if that client is muted. If they are muted, then the message will not be sent to them but if they are unmuted, then it will continue to get the client name that matches the target client name that was passed in and will send them the message privately by specifying the matched name and doing sendMessage after to only send it to them. This was

done by using the variable of the iterator which in this case was "st" and then doing "st.sendMessage()" with it taking the client id and the message which allows for only the target client to see this message and will tell them who sent the message. In the code below after this, another message is sent to the client (sender) to verify that they sent the message to the client so they are able to see their own message as well.

Mute/Unmute Users (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing feature working

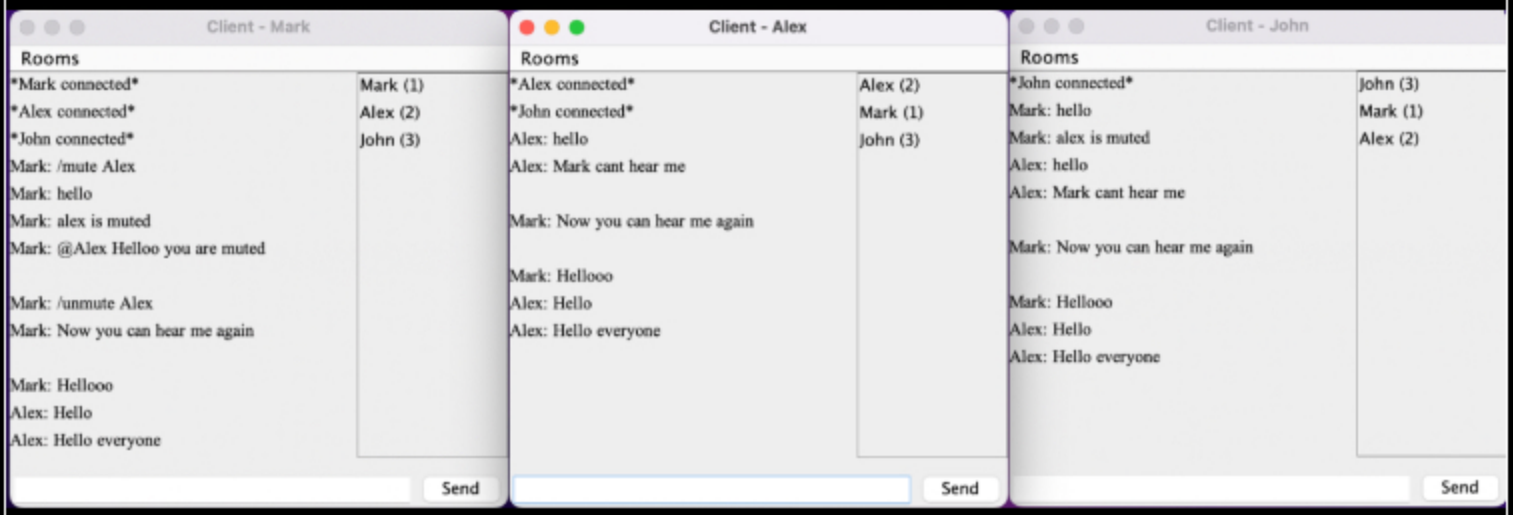
Checklist *The checkboxes are for your own tracking

#	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



Left client is muting the Alex client in middle shown by /mute Alex. Middle client can no longer receive from or send messages to the left client shown by both clients sending messages but don't see each other's. Right side client sees all messages. Private message is also done but still won't send message to middle client since they're still muted. Unmuting is done by /unmute Alex and now both are able to talk again.

Checklist Items (4)

#1 Should have 3 clients in the same room

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

#3 Demo mute also being accounted for with private messages

#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

Project > Client > Client.java > Client > sendMessage(String)

```
26  public enum Client {
392
393     // UCID - ob75 - April 13, 2024
394     public void sendMute(String clientMuteName) throws IOException {
395         Mute_UnmutePayload m = new Mute_UnmutePayload();
396         m.setPayloadType(PayloadType.MUTE);
397         m.setClientMuteName(clientMuteName);
398         out.writeObject(m);
399     }
400
401     // UCID - ob75 - April 13, 2024
402     public void sendUnmute(String clientUnmuteName) throws IOException {
```

```
403         Mute_UnmutePayload um = new Mute_UnmutePayload();
404         um.setPayloadType(PayloadType.UNMUTE);
405         um.setClientUnmuteName(clientUnmuteName);
406         out.writeObject(um);
407     }
408
```

Send Methods for mute and unmute to send target client to server side. Under UCID comment line 394 - 407 for relevant code.

Checklist Items (2)

#5 Include ucid and date comment

#6 Clearly caption screenshots

Project > Client > Client.java > Client > sendMessage(String)

```
26  public enum Client {
49
50      // UCID - ob75 - April 13, 2024
51      private static final String MUTE = "/mute";
52      private static final String UNMUTE = "/unmute";
53
```

/command methods for mute and unmute on line 51 and 52 under UCID command.

Checklist Items (2)

#5 Include ucid and date comment

#6 Clearly caption screenshots

Project > Common > PayloadType.java > PayloadType > UNMUTE

You, 4 days ago | 2 authors (Omar Barrera and others)

```
1  package Project.Common;
2
3  public enum PayloadType {
4      CONNECT, DISCONNECT, MESSAGE, CREATE_ROOM, JOIN_ROOM, LIST_ROOMS, CLIENT_ID, SYNC_CLIENT, ROLL, FLIP //UCID - ob75 - March 30,2024
5      , PM, MUTE, UNMUTE //UCID - ob75 - April 10, 2024      You, 4 days ago • Uncommitted changes
6  }
```

Line 5 with MUTE and UNMUTE is relevant payload types for code. Next to UCID comment.

Checklist Items (2)

#5 Include ucid and date comment

#6 Clearly caption screenshots

Project > Client > Client.java > Client > sendMessage(String)

```
26   public enum Client {
323      public void sendMessage(String message) throws IOException {
348          // UCID - ob75 - April 13, 2024      You, 13 hours ago • Uncommitted changes
349          else if (message.startsWith(MUTE)) {
350              String clientMuteName = message.replace(MUTE, replacement:"").trim();
351
352              try {
353                  sendMute(clientMuteName);
354              } catch (IOException e) {
355                  System.out.println(TextFX.colorize(text:"Socket error", Color.RED));
356              } catch (Exception e) {
357                  e.printStackTrace();
358              }
359              return;
360          }
361
362          // UCID - ob75 - April 13, 2024
363          else if (message.startsWith(UNMUTE)) {
364              String clientUnmuteName = message.replace(UNMUTE, replacement:"").trim();
365
366              try {
367                  sendUnmute(clientUnmuteName);
368
369              } catch (IOException e) {
370                  System.out.println(TextFX.colorize(text:"Socket error", Color.RED));
371              } catch (Exception e) {
372                  e.printStackTrace();
373              }
374              return;
375          }
376
```

Client side checking if message starts with /mute or /unmute and class send method. Under UCID comment, line 349 - 375 is relevant code.

Checklist Items (2)

#5 Include ucid and date comment

#6 Clearly caption screenshots

Project > Common > Mute_UnmutePayload.java > Mute_UnmutePayload > MutePayload(String)

```java
package Project.Common;

//UCID - ob75 - April 13, 2024

public class Mute_UnmutePayload extends Payload{
    private String clientMuteName;
    private String clientUnmuteName;

    public void MutePayload(String clientMuteName2){
        this.clientMuteName = clientMuteName2;
        setPayloadType(PayloadType.MUTE);
    }

    public void UnmutePayload(String clientUnmuteName2){
        this.clientUnmuteName = clientUnmuteName2;
        setPayloadType(PayloadType.UNMUTE);
    }

    public String getClientMuteName() {
        return clientMuteName;
    }

    public void setClientMuteName(String clientMuteName) {
        this.clientMuteName = clientMuteName;
    }

    public String getClientUnmuteName() {
        return clientUnmuteName;
    }

    public void setClientUnmuteName(String clientUnmuteName) {
        this.clientUnmuteName = clientUnmuteName;
    }

}
```

New payload class for mute and unmute and sets payload type. Entire class is relevant code with UCID comment.

Checklist Items (2)

#5 Include ucid and date comment

#6 Clearly caption screenshots

Project > Server > ServerThread.java > ServerThread > processPayload(Payload)

```java
public class ServerThread extends Thread {
```

```
36      // UCID - ob75 - April 13, 2024
37      private List<String> isMutedClients = new ArrayList<String>();
38
```

New array list in server thread side named isMutedClients under UCID comment, line 37.

Checklist Items (3)

#1 ServerThread should have a list of who they muted

#5 Include ucid and date comment

#6 Clearly caption screenshots

Project > Server > ServerThread.java > ServerThread > processPayload(Payload)

```
25  public class ServerThread extends Thread {
136
137     public void addMute(String clientMuteName) {
138
139         if (!isMutedClients.contains(clientMuteName))
140         {
141             isMutedClients.add(clientMuteName);
142         }
143
144     }
145
146     public void removeMute(String clientUnmuteName) {
147         if (isMutedClients.contains(clientUnmuteName))
148         {
149             isMutedClients.remove(clientUnmuteName);
150         }
151     }
152
153     public boolean checkMutedList(String clientMuteName){
154         if (isMutedClients.contains(clientMuteName)){
155             return true;
156         }
157         else {
158             return false;
159         }
160     }
161
```

New methods for adding muted clients (addMute), removing muted clients (removeMute), and checking the muted list (checkMutedList). Under UCID comment Line 137 - 160 is relevant code.

Checklist Items (3)

#2 ServerThread should expose and add, remove, and is muted check to room

#5 Include ucid and date comment

#6 Clearly caption screenshots

Project > Server > ServerThread.java > ServerThread

```
25   public class ServerThread extends Thread {
231      private void processPayload(Payload p) {
302
303          // UCID - ob75 - April 13, 2024
304          case MUTE:
305          try {
306              Mute_UnmutePayload mp = (Mute_UnmutePayload) p;
307              (currentRoom).setMute(this, mp.getClientMuteName());
308          } catch (Exception e) {
309              e.printStackTrace();
310          }
311          break;
312
313          // UCID - ob75 - Aprl 13, 2024
314          case UNMUTE:
315          try {
316              Mute_UnmutePayload mp = (Mute_UnmutePayload) p;
317              (currentRoom).setUnmute(this, mp.getClientUnmuteName());
318          } catch (Exception e) {
319              e.printStackTrace();
320          }
321          break;
322
```

Sending over separate payloads to Room side for MUTE and UNMUTE. Under UCID comment, lines 304 - 321 for relevant code.

Checklist Items (2)

#5 Include ucid and date comment

#6 Clearly caption screenshots

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

```
11   public class Room implements AutoCloseable {
99
100      // UCID - ob75 - April 13, 2024
101      public void setMute(ServerThread client, String clientMuteName) {
102          client.sendMessage(client.getClientId(), String.format(format:"/mute %s", clientMuteName));
103          Iterator<ServerThread> muteIter = clients.iterator();
104          while (muteIter.hasNext()) {
105              ServerThread mClients = muteIter.next();
106              if (mClients.getClientName().equalsIgnoreCase(clientMuteName)) {
107                  client.addMute(clientMuteName);
108
109              }
110          }
111      }
112
113      // UCID - ob75 - April 13, 2024
114      public void setUnmute(ServerThread client, String clientUnmuteName) {
115          client.sendMessage(client.getClientId(), String.format(format:"/unmute %s", clientUnmuteName));
116          Iterator<ServerThread> muteIter = clients.iterator();
```

```
117         while (muteIter.hasNext()) {
118             ServerThread umClients = muteIter.next();
119             if (umClients.getClientName().equalsIgnoreCase(clientUnmuteName)) {
120                 client.removeMute(clientUnmuteName);
121
122             }
123         }
124     }
125
```

New methods for Mute and Unmute named setMute and setUnmute and checks for target name in chat to add or remove from list. Under UCID comment from lines 101 - 124.

Checklist Items (3)

#3 Room should handle the mute list when receiving the appropriate payloads

#5 Include ucid and date comment

#6 Clearly caption screenshots

```
83      // UCID - ob75 - April 10, 2024
84      public void setPrivateM(ServerThread client, String clientName, String message) {
85          Iterator<ServerThread> iter = clients.iterator();
86          while (iter.hasNext()) {
87              ServerThread st = iter.next();
88              if (client != null && client.checkMutedList(st.getClientName()) == false
89                      && (st.checkMutedList(client.getClientName()) == false)) {
90                  if (st.getClientName().equalsIgnoreCase(clientName)) {
91                      st.sendMessage(client.getClientId(), message);
92
93                  }
94              }
95
96              }client.sendMessage(client.getClientId(), message);
97          }
98
99
```

Handling private message from being sent out if user is muted or not. New code for checking mute list is from line 88-89.

Checklist Items (3)

#4 Room should check the mute list during send message and private messages

#5 Include ucid and date comment

#6 Clearly caption screenshots

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

```
11   public class Room implements AutoCloseable {
244      protected synchronized void sendMessage(ServerThread sender, String message) {
291
292          // UCID - ob75 - April 13, 2024
293          if (sender != null && sender.checkMutedList(client.getClientName()) == false
294              && (client.checkMutedList(sender.getClientName()) == false)) {
295              boolean messageSent = client.sendMessage(from, message);
296
297              if (!messageSent) {
298                  handleDisconnect(iter, client);
299
300              }
301          }
302      }
303  }
304
```

In room side, checking mute list for muted clients and won't send message if in list and will send if not in list. Under UCID, lines 293 - 294 is new code for checking mute list.

Checklist Items (3)

#4 Room should check the mute list during send message and private messages

#5 Include ucid and date comment

#6 Clearly caption screenshots

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 first thing that was done for mute and unmute was creating the / commands for them and setting them to the

variables MUTE and UNMUTE. After this, a check method was implemented into the sendMessage method in the client side to look for the command for MUTE or UNMUTE. These are two separate else if statements to send over the name if the user wants to mute or if the user wants to unmute separately so they don't interfere with each other. The command is cut out and the name for the mute or name for unmute is sent to server thread depending which command was done. Two more send methods were created, one named sendMute which takes the clientMuteName and sets the payload type to MUTE and the other sendUnmute which takes the clientUnmuteName and sets the payload type to UNMUTE. Two payload types were created in the PayloadType class named MUTE and UNMUTE for these send message to be set and sent out to server thread. After this, a new payload class was created called Mute_UnmutePayload which takes either of these payload types and creates the new payload with getters and setters corresponding to whether user decided to mute or unmute and sets the clientMuteName or clientUnmuteName to pass this information to server thread. In the server thread side, a new String ArrayList was created called isMutedList so that muted clients will be added or removed from the list depending on the client's command. Three new methods are created to make the list function properly and add to (addMute(clientMuteName)), remove (removeMute(clientUnmuteName)), or check the list (checkMutedList(clientMuteName)). addMute method takes in the clientMuteName and adds them to the list if the list does not contain the client name by doing an if check on the list and if the list does have the name already, then it does nothing. removeMute does the opposite in taking the clientUnmuteName and checks the list to see if the client is on the list and if they are and the client is trying to unmute them, then it removes them from the muted list. checkMutedList checks to see if the muted client name is just on the list and returns true if they are on the list and returns false if not on list. Then in the proccessPayload method, the cases for MUTE and UNMUTE are created and takes in their payloads and passes that information onto the room. In the room side now, two methods are created, one called setMute and takes the client (sender of mute) and clientMuteName and other setUnmute which takes client (sender of unmute) and clientUnmuteName. In the mute method, a message is sent to the sender first saying they muted the client and then create an iterator to iterate through the clients to see which matches the target for the mute and once it grabs the client that matches, it adds the client to the mute list through client.addMute(clientUnmuteName). The same is done for setUnmute but instead of adding, it removes them from the list by doing client.removeMute(clientUnmuteName). Lastly for the logic of not receiving the message from client if muted, in the sendMessage of Room side, an if statement is placed to check first if sender is an actual client in chat and not null, then checks the sender mute list and takes the target client's name and checks if set to false meaning if not in list, and this is also done in reverse to check the other mute list from the target client to see if that is false too and will only send the message if all the conditions pass or in this case they are unmuted. This allows for the muted person to no longer receive messages and the same is done for private messaging in its method in Room before the if statement of checking the target of client private messaging so that if the client tries to private message the muted client, it will not send that message over to that client and only will send if they become unmuted.

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/OmarBarrera1/ob75-IT114-002/pull/10

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

Some Issues I had with this assignment that I eventually learned after solving it is firstly with muting the message so that the client and target don't receive each other's messages. The issues with this I had was trying to figure out how to stop the message from arriving the 2 clients if one of them was muted from the other and the way this was solved was by creating an if statement that checks the mute list and won't send the message if they are on the list. One more issue was with sending the original command of mute and unmute back to the client so they see the action was actually performed and this was solved simply by putting a client.sendMessage() and then sending the information and message of what was done.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

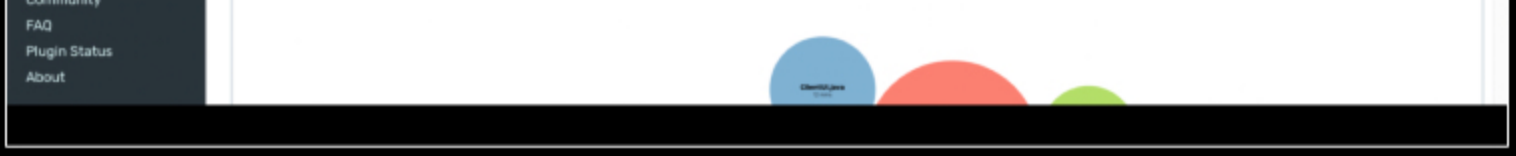
Details:

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.

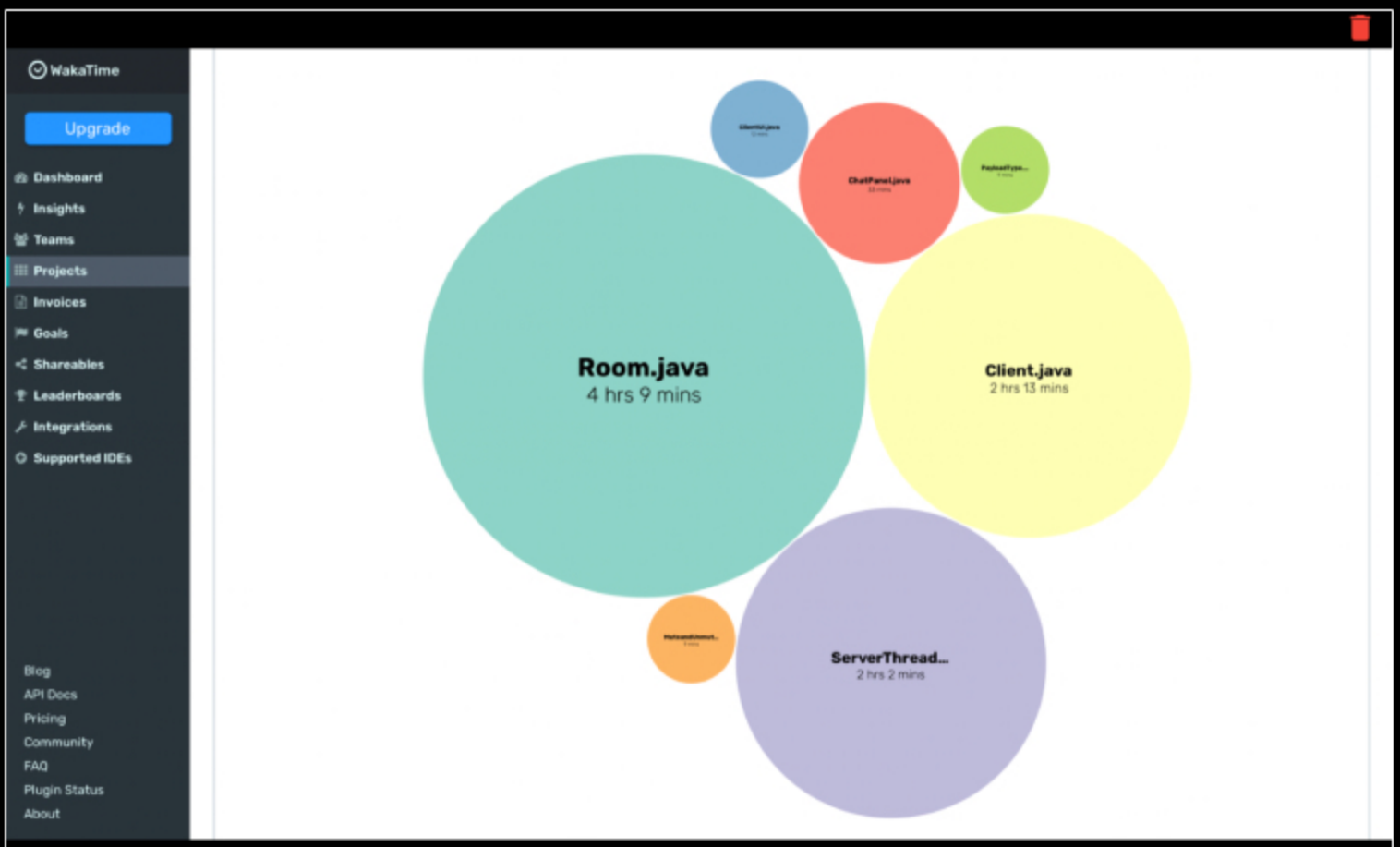
Task Screenshots:

Gallery Style: Large View

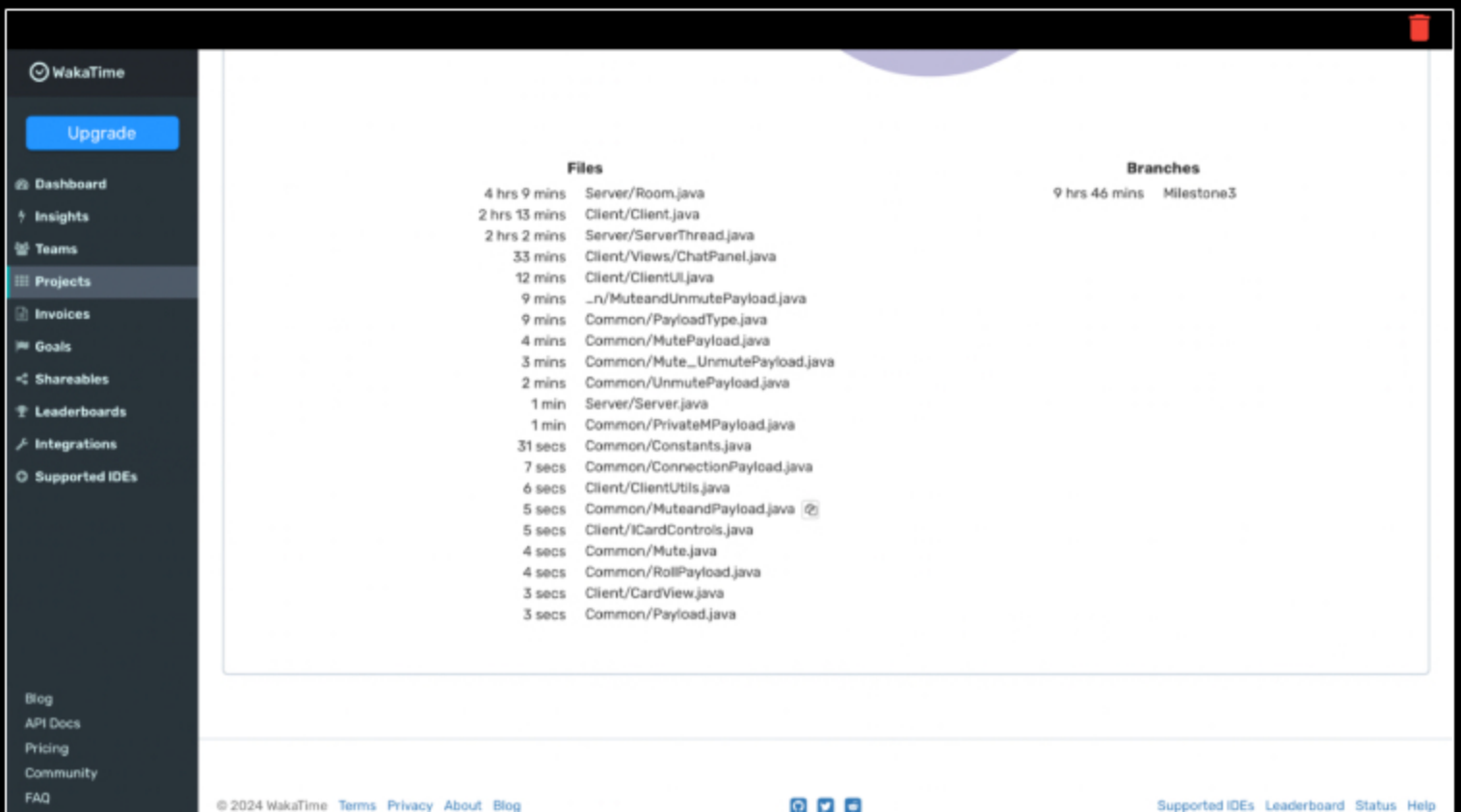
Small Medium Large



Waka time top page showing projects folder and time spent in past week.



Waka time showing middle of page of all classes worked on and time spent on each.



Waka time showing each files and time spent and on the right is the Milestone3 Branch and total time spent in branch.

End of Assignment