Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-004-S2024/it114-chatroom-milestone-3-2024/grade/mbh3

IT114-004-S2024 - [IT114] Chatroom Milestone 3 2024

Submissions:

Submission Selection

1 Submission [active] 4/28/2024 4:01:21 AM

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 checkbox are for your own tracking

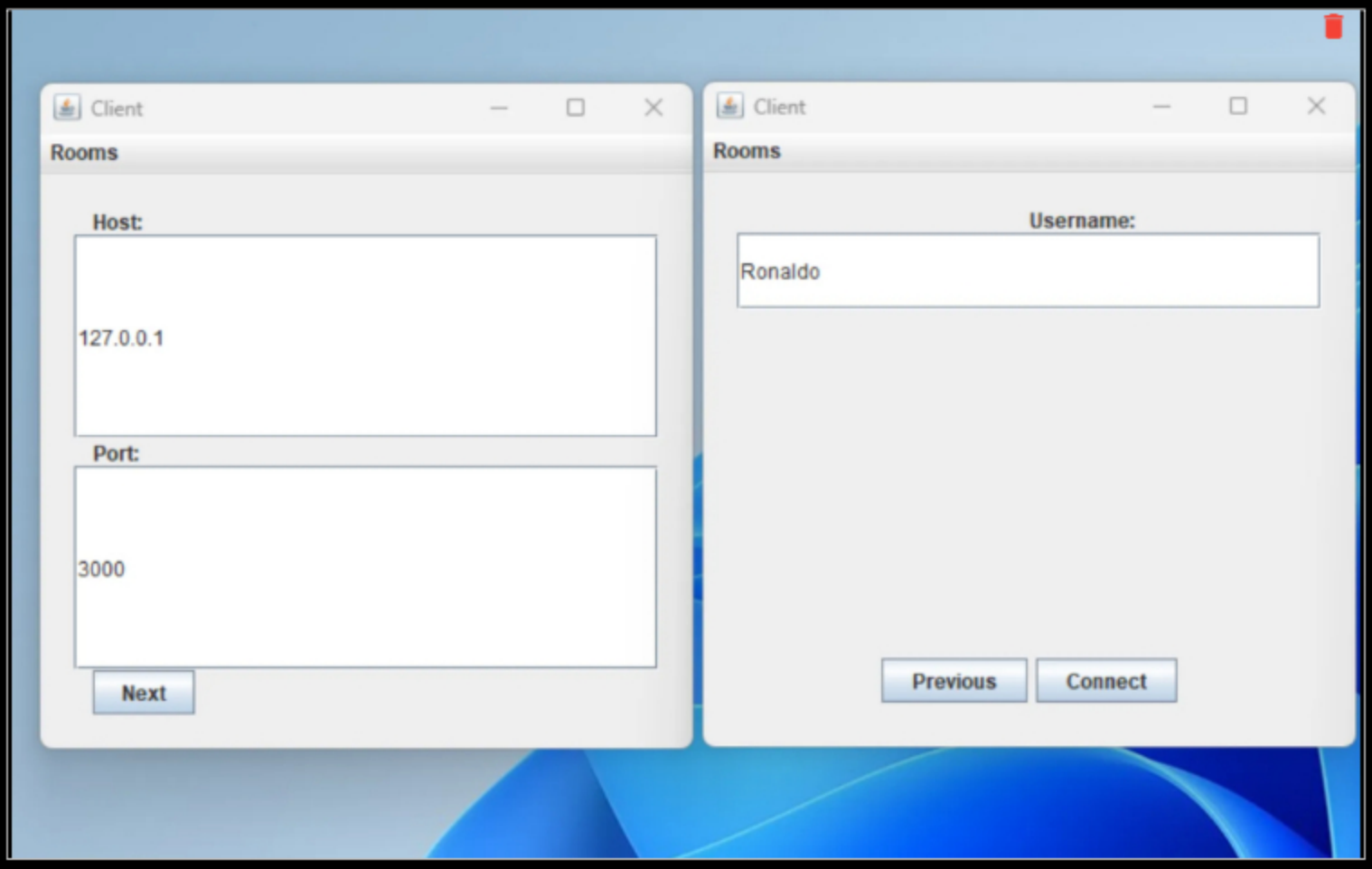
#	Points	Details

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

Task Screenshots:

Gallery Style: Large View

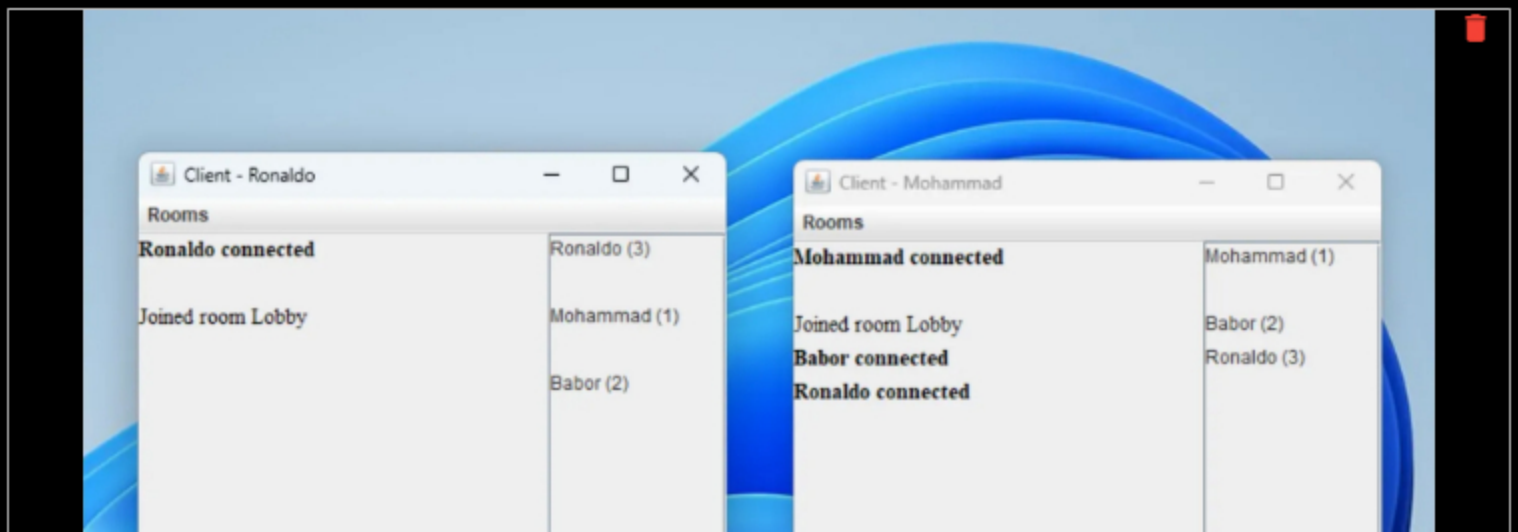
Small Medium Large

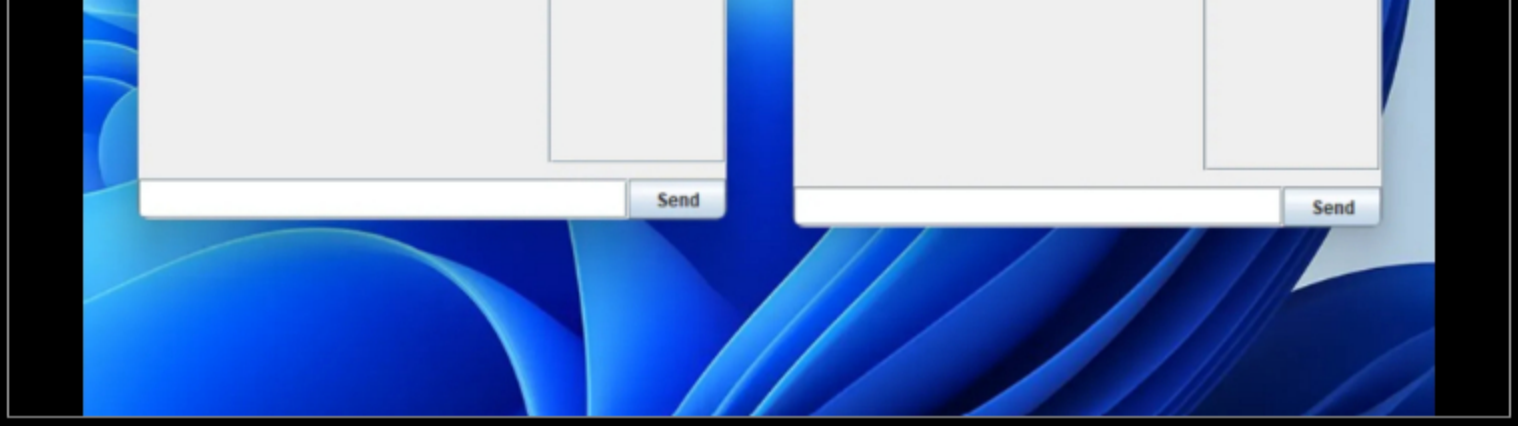


User Details Panel

Checklist Items (1)

#2 User Details Panel



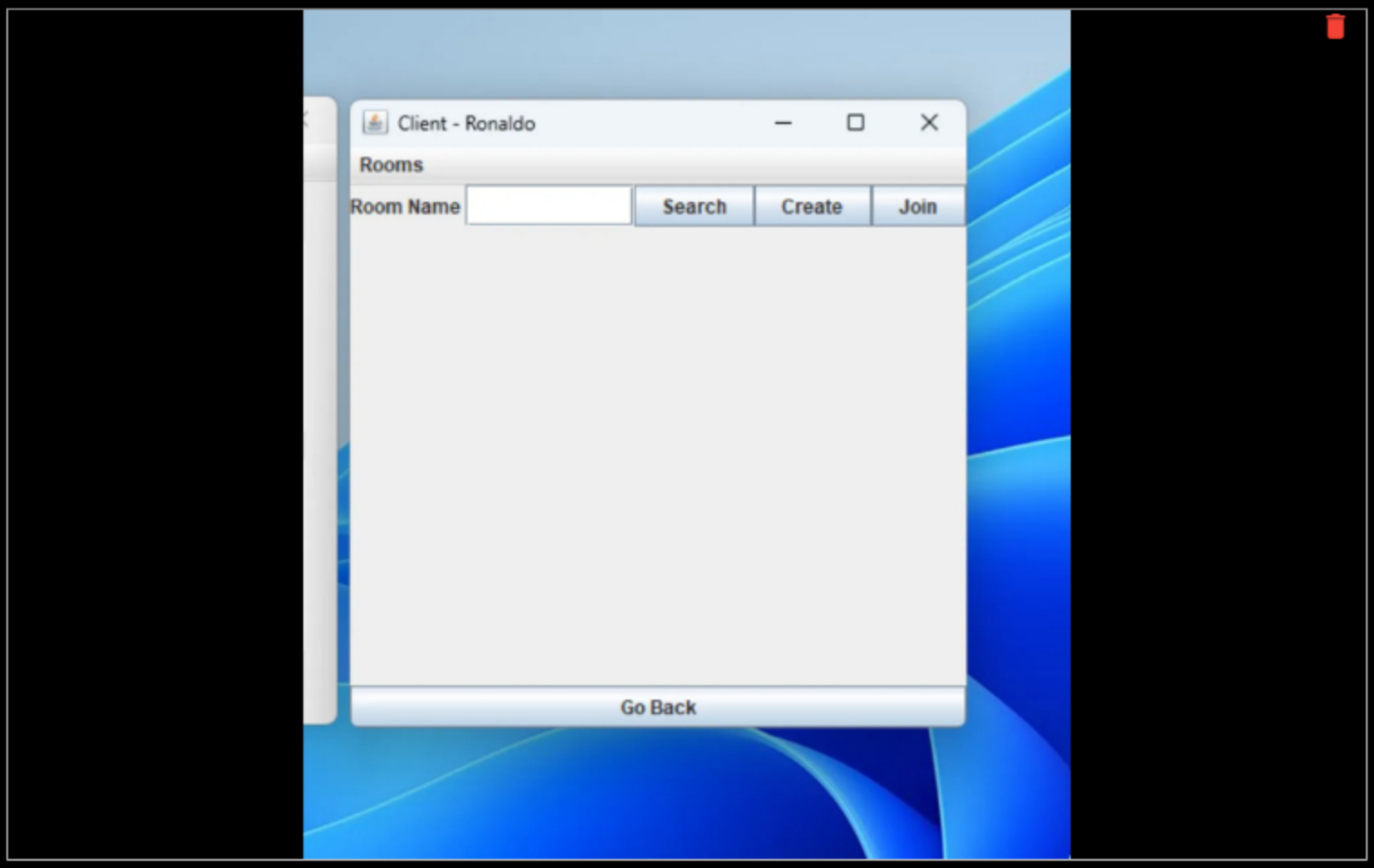


connection panel

Checklist Items (2)

#1 Connection Panel

#4 Clearly caption screenshots



details

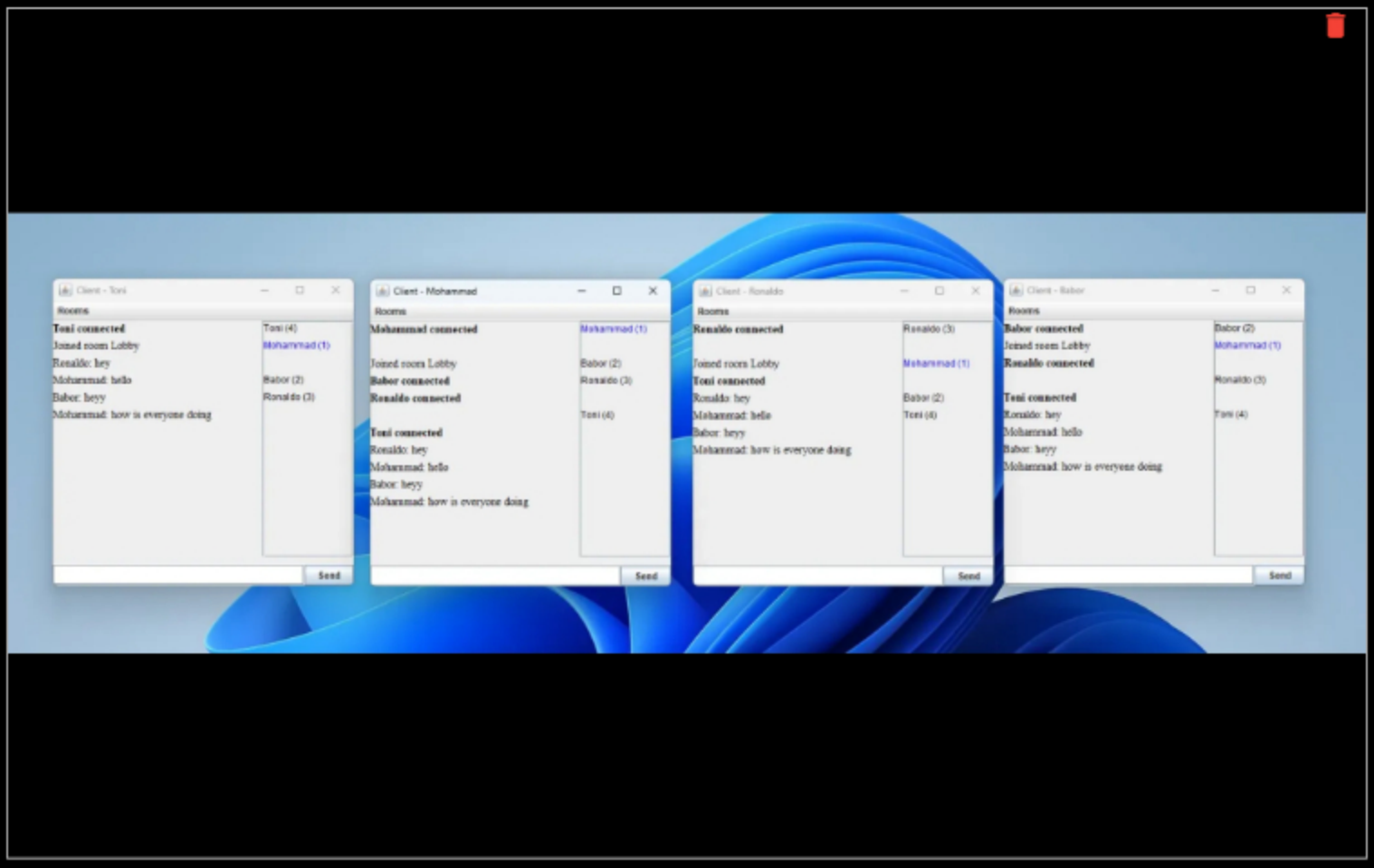
Checklist Items (4)

#1 Connection Panel

#2 User Details Panel

#3 Chat Panel

#4 Clearly caption screenshots



chat

Checklist Items (4)

#1 Connection Panel

#2 User Details Panel

#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

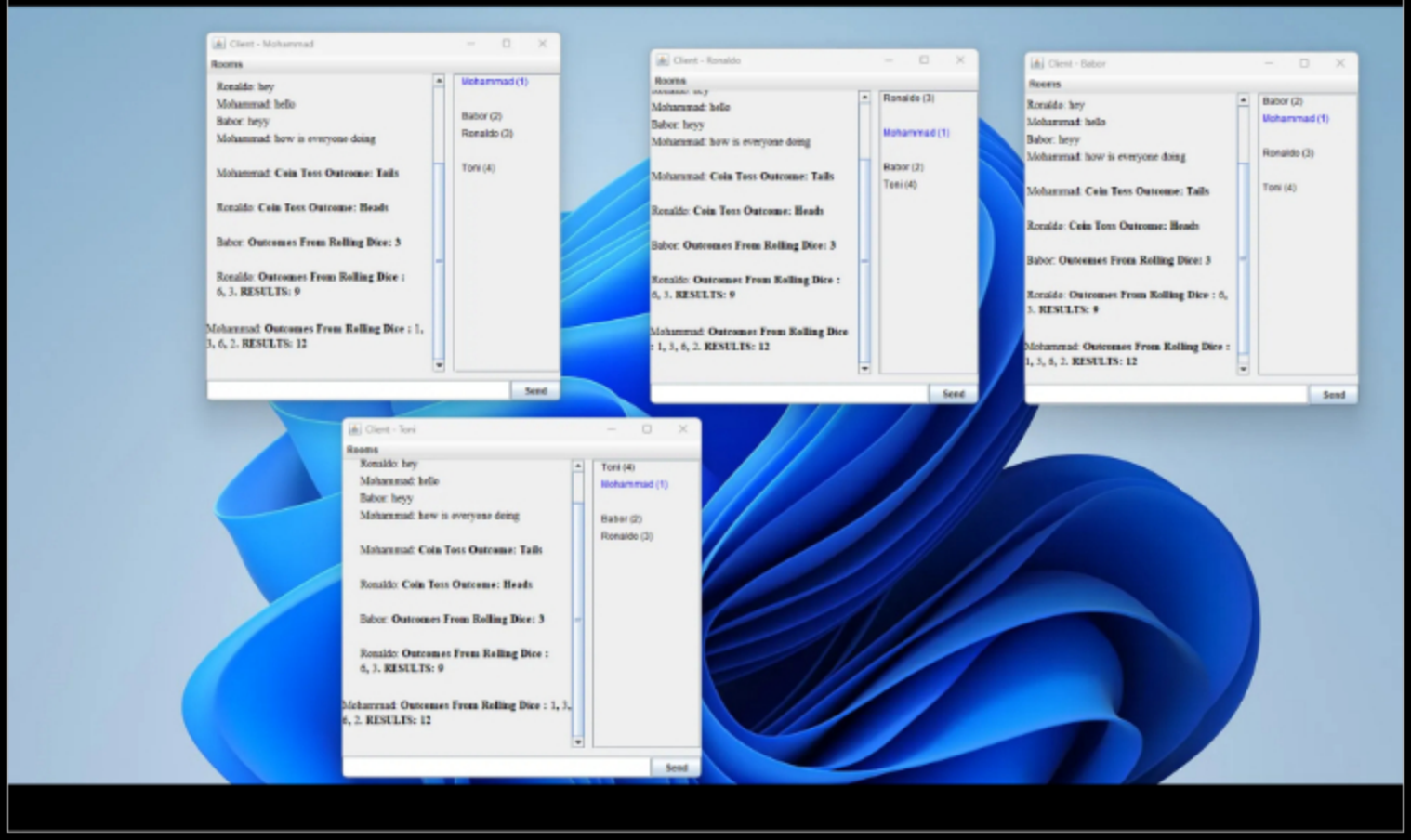
#	Points	Details
☐ #1	1	Flip output in a different format than normal messages

☐ #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 and roll method

Checklist Items (4)

#1 Flip output in a different format than normal messages

#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

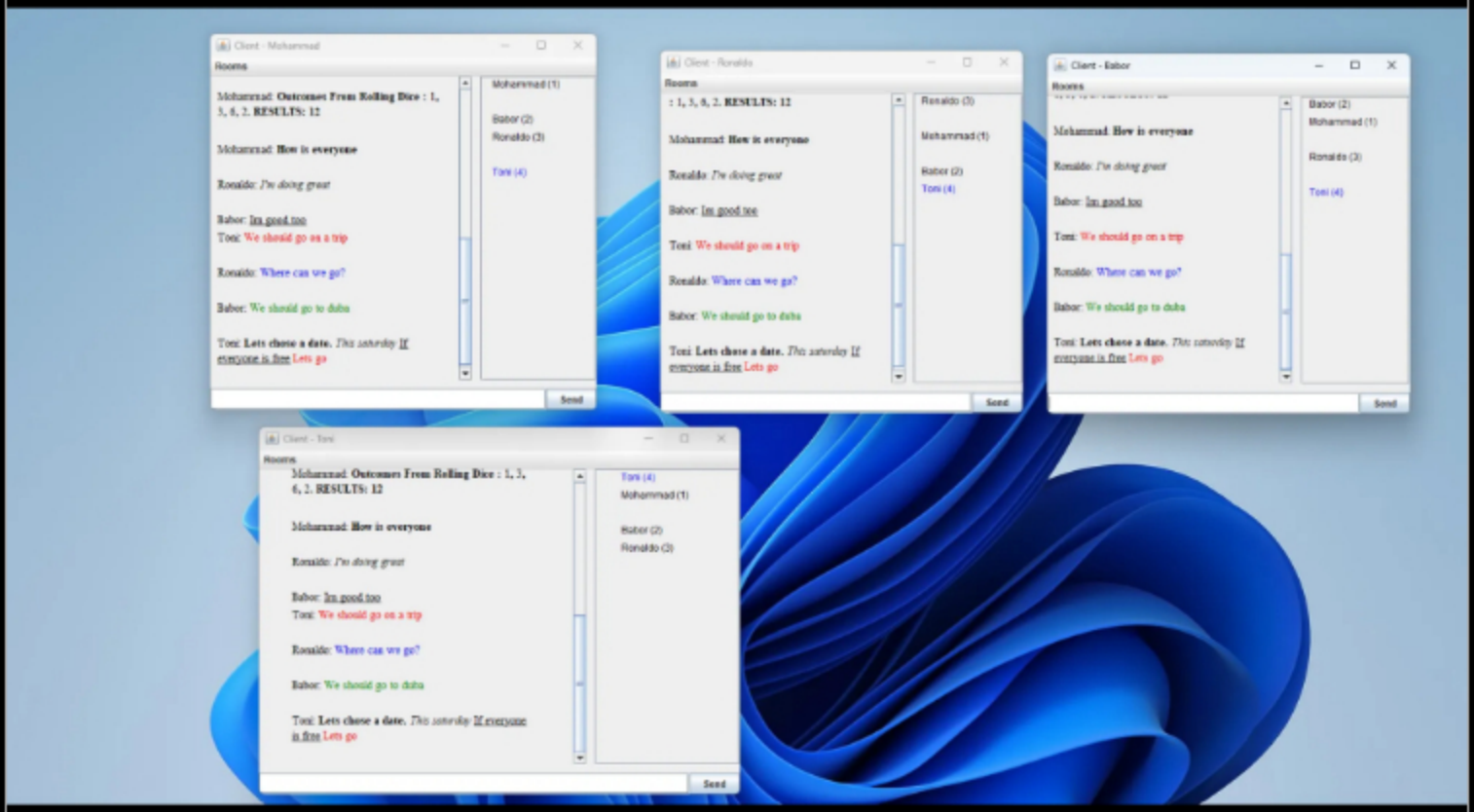
Checklist *The checkboxes are for your own tracking

#	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



Everything Together

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

COLLAPSE

Task #3 - Points: 1

Text: Screenshot of the code solving the formatting display

Checklist

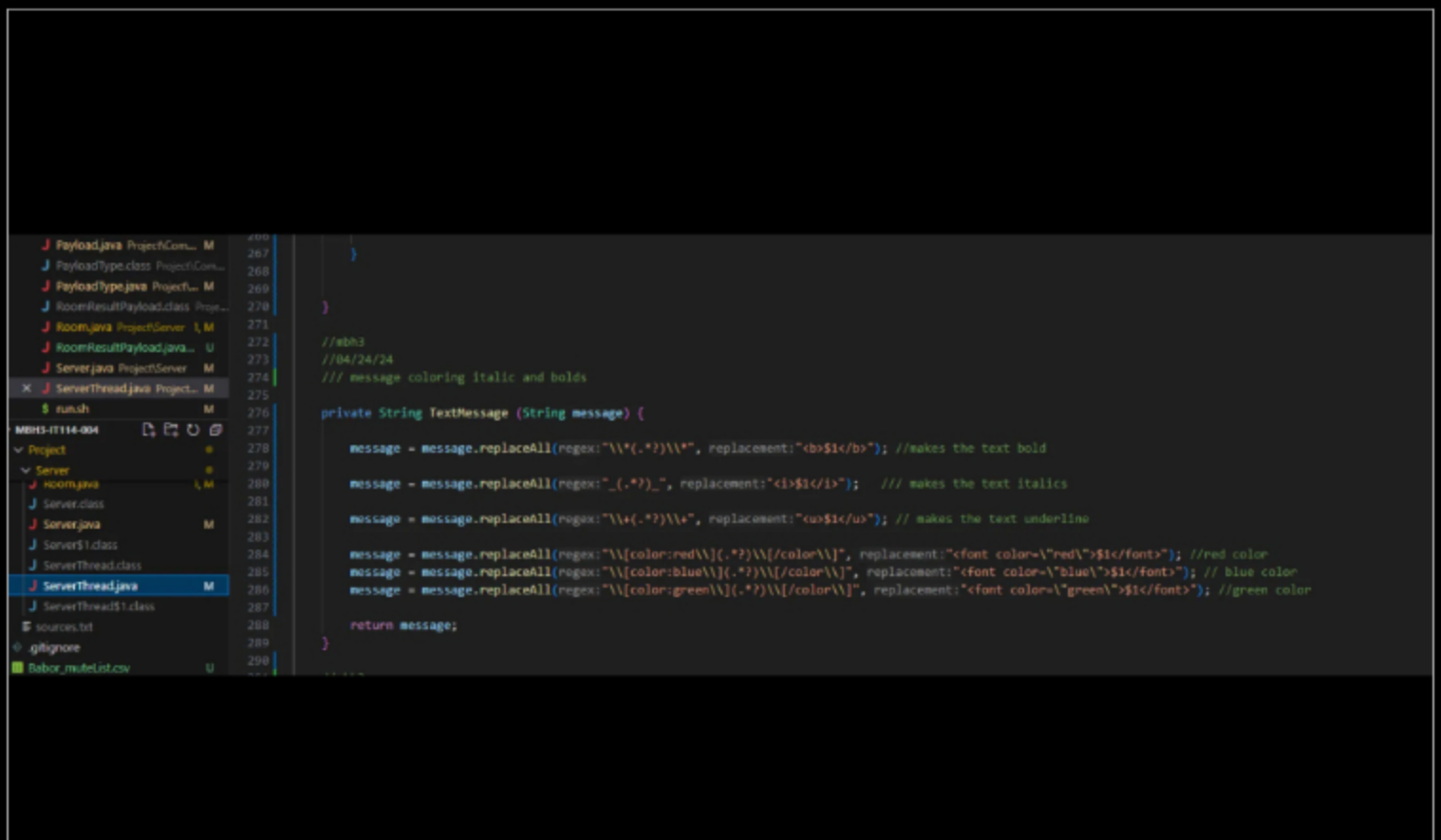
*The checkboxes are for your own tracking

#	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



code screenshot

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:

I created a method called formatMessage which takes a string and looks for certain patterns that signal different text styles. Once I detect these patterns, I replace them with HTML tags that represent the corresponding styles.

For Bold Text: I decided that any text the user encloses within asterisks should be bold. So I wrote a regular expression "(*(.*?)*)" to find any text that's wrapped in asterisks. When I find a match, I use replaceAll to surround the found text with and tags, which is the HTML notation for bold text.

For italic text, I used underscores. If a user wraps their text like _this_, it should appear italicized. The regex "(_.+?_)" captures these patterns, and I replace them with and tags using replaceAll method.

Underlined Text: To underline text, I chose to have the user wrap their text with pluses, like +this+. I detect this with the regular expression "(\\+(.*?)\\+)" and then replace the occurrences with <u> and </u> tags, which denote underlined text in HTML.

For the Colored Text: I wanted to give users the option to color their text red, blue, or green. For this, I created three separate regular expressions that look for the text wrapped within [color:red], [color:blue], and [color:green] . Using the replaceAll function, I then wrap the detected text in <font> tags with the color attribute set to the corresponding color. After applying all these transformations, the formatMessage method returns the modified message string with HTML tags in place. When this string is rendered in an HTML-supported UI, a Java component with HTML rendering capabilities like a JEditorPane, the text styles become visible just as I intended.

Private Message with @ (2 pts.)

COLLAPSE

Task #1 - Points: 1

^COLLAPSE ^

Text: Screenshots demoing private message

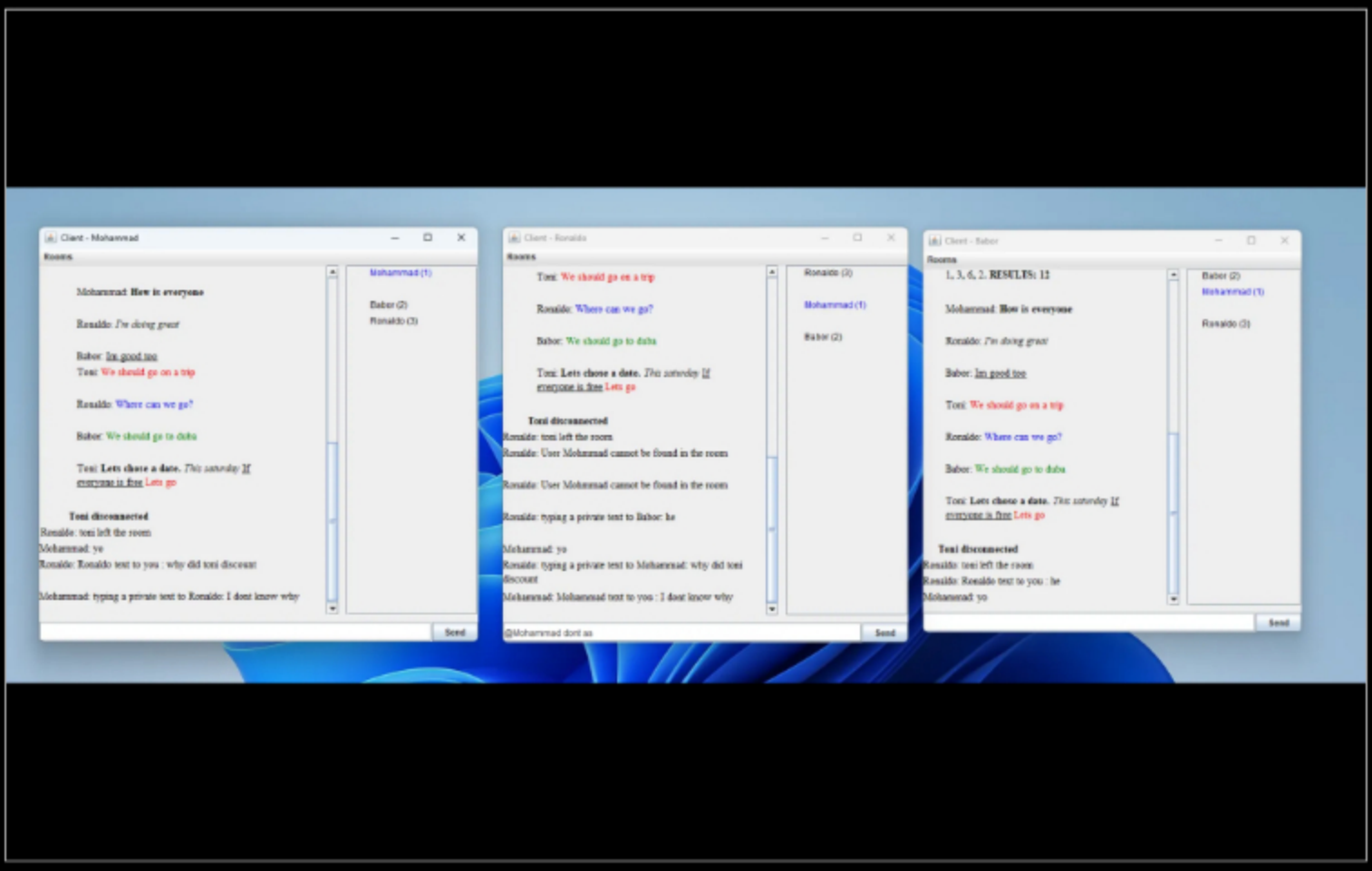
Checklist *The checkbox are for your own tracking

#	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



as you can see Mohammad and Ronaldo private message and Babor cannot see them

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

```
25  public class ServerThread extends Thread {
336
337  //mbh3
338  ///04/24/24
339  // flip game head or tails
340  //private message method
341
342
343      private void processFlipCommand() {
344
345          String result = (Math.random() < 0.5) ? "Heads" : "Tails";
346          String flipResult = "<b>Coin Toss Outcome: " + result+" </b>";
347          if (currentRoom != null) {
348              currentRoom.sendMessage(this, flipResult);
349          }
350      }
351      private void processPrivateMessage(String message) {
352          int spaceIndex = message.indexOf(str:" ");
353          if (spaceIndex != -1) {
354              String receiverName = message.substring(beginIndex:1, spaceIndex);
355              String privateMessage = message.substring(spaceIndex + 1);
356
357              if (currentRoom != null) {
358                  ServerThread receiver = currentRoom.findClientByName(receiverName);   You, 46 minutes ago • Uncommitted changes
359                  if (receiver != null) {
360                      sendMessage(getClientId(), " typing a private text to  "+ receiverName + ": " + privateMessage);
361                      receiver.sendMessage(getClientId(),getClientName()+"  text to you : " + privateMessage);
362                  } else {
363                      sendMessage(getClientId(), "User " + receiverName + " cannot be found in the room ");
364                  }
365              }
366          }
```

code 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

```
            /// TODO migrate to lobby
//mbh3
//04/24/24
        You, 1 second ago • Uncommitted changes
                String message = p.getMessage();

                if (message.startsWith(prefix:"/roll")) {
                    processRollCommand(message);

                } else if (message.startsWith(prefix:"/flip")) {
                    processFlipCommand();

                } else if (message.startsWith(prefix:"@")) {
                    processPrivateMessage(message);

                } else if (message.startsWith(prefix:"mute ")){
                    processMuteCommand(message);

                } else if (message.startsWith(prefix:"unmute ")){
                    processUnmuteCommand(message);
                } else {
                    if (currentRoom != null) {
                        currentRoom.sendMessage(this, p.getMessage());
                    } else {
                        logger.log(Level.INFO, msg:"Migrating to lobby on message with null room");
                        Room.joinRoom(roomName:"lobby", this);
                    }
                }
                break;
```

code generator

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

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:

I've designed a private messaging system in my chat application that lets users send messages directly to one another without others seeing. When I receive a message starting with "@", I extract the recipient's username and the actual message. Then, I find the recipient in the current room by their username, which gives me their unique client ID.

Once I have both the sender's and recipient's client IDs, I send the private message to the intended recipient. I also make sure to send a confirmation back to the sender, so they know the message has been delivered. This setup ensures that only the sender and the recipient can see the private message, maintaining privacy within the chat room.

Mute/Unmute Users (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing feature working

Checklist

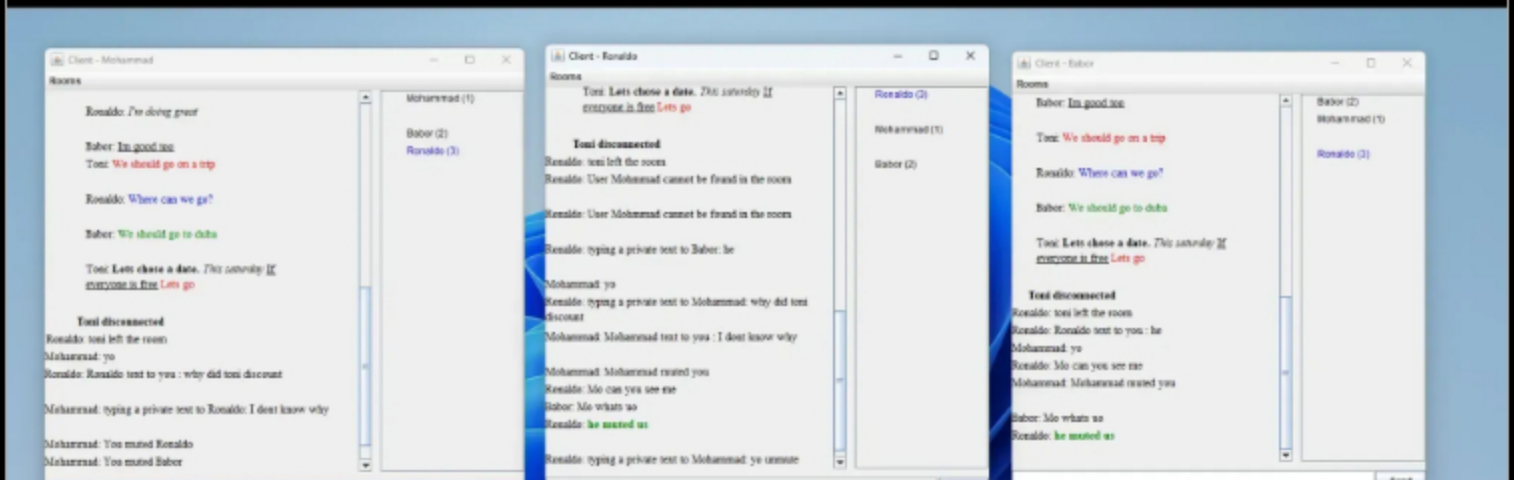
*The checkboxes are for your own tracking

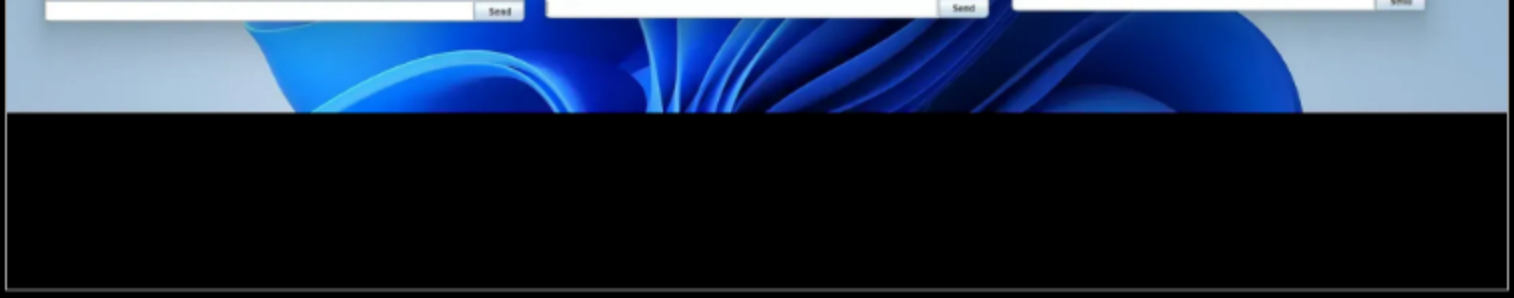
#	Points	Details
☐ #1	1	Should have 3 clients in the same room
☐ #2	1	Demo mute preventing messages between the muter and the target
☐ #3	1	Demo mute also being accounted for with private messages
☐ #4	1	Demo unmute allowing the messages again from the target to the unmuter

Task Screenshots:

Gallery Style: Large View

Small Medium Large





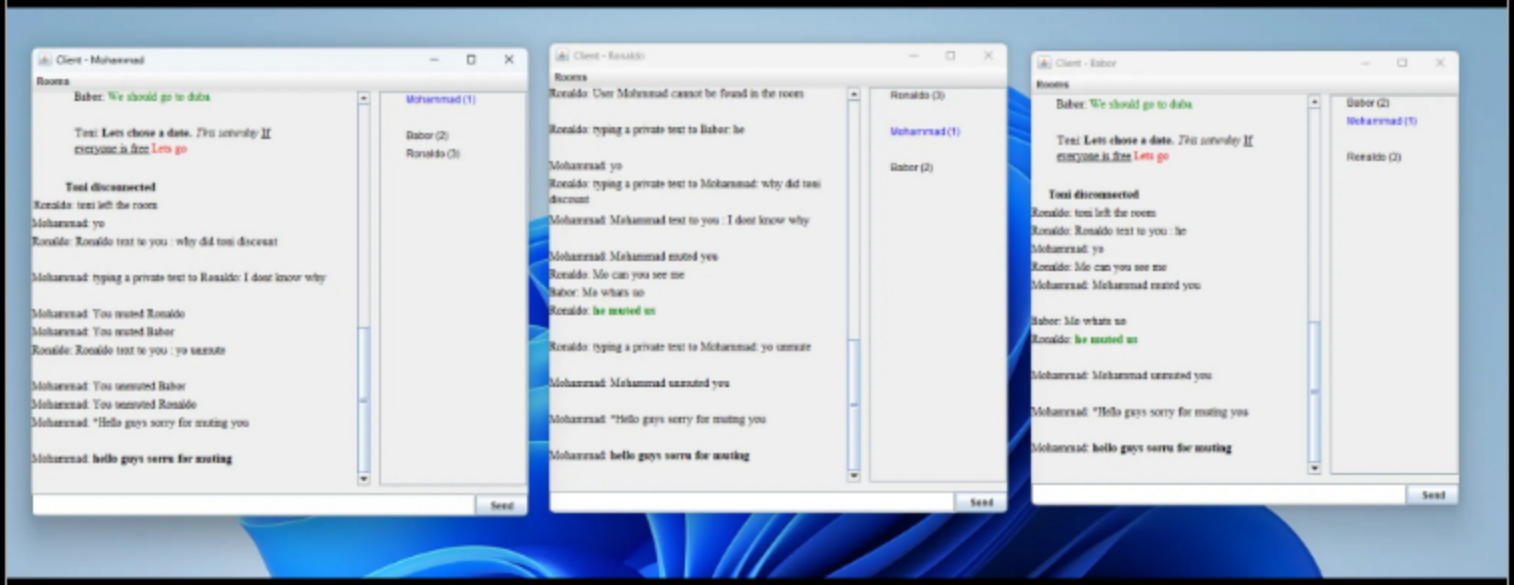
mute demo

Checklist Items (3)

#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



unmute demo

Checklist Items (2)

#1 Should have 3 clients in the same room

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

Task #2 - Points: 1

Text: Screenshots of the related code

Checklist

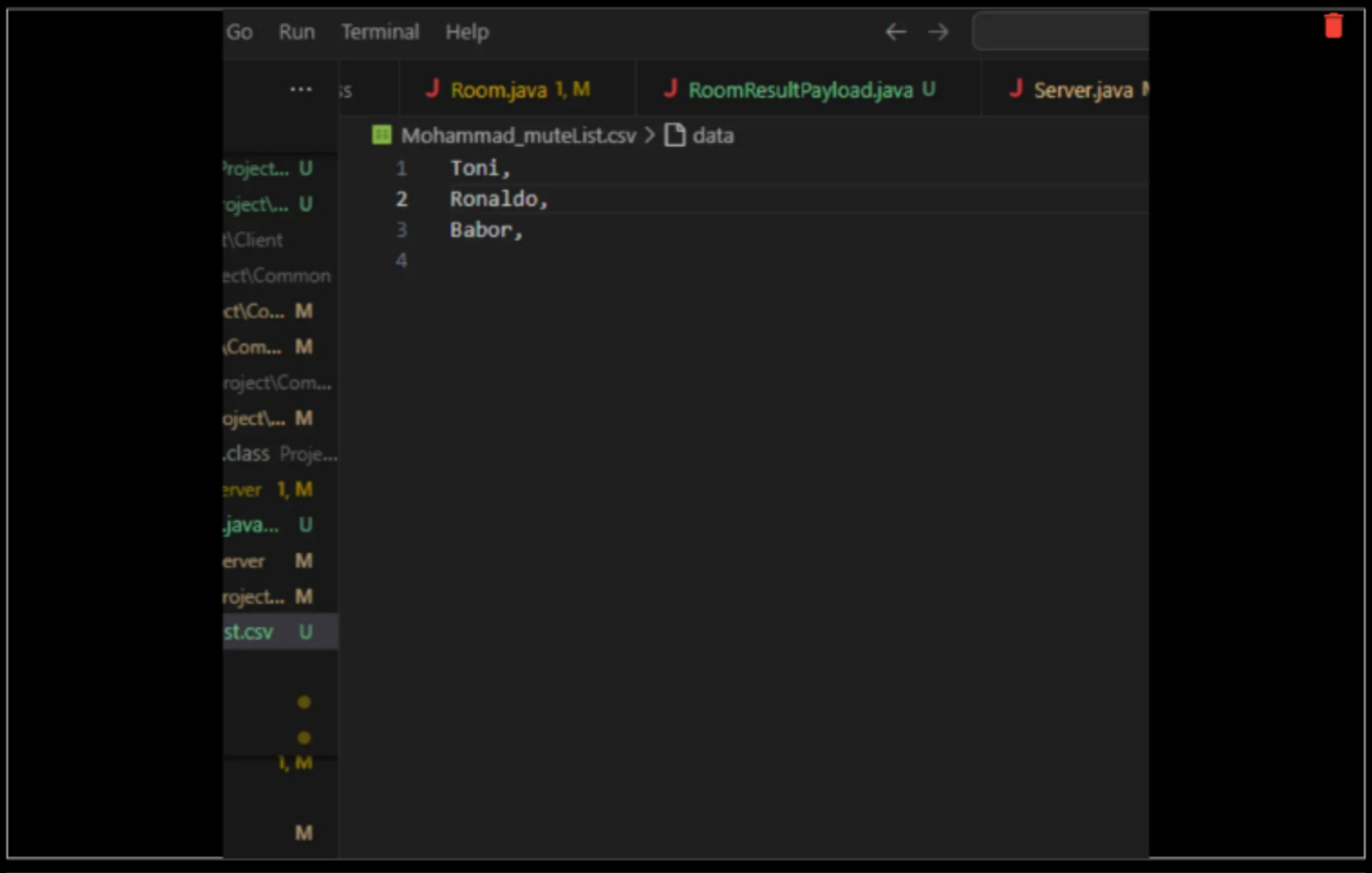
*The checkboxes are for your own tracking

#	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

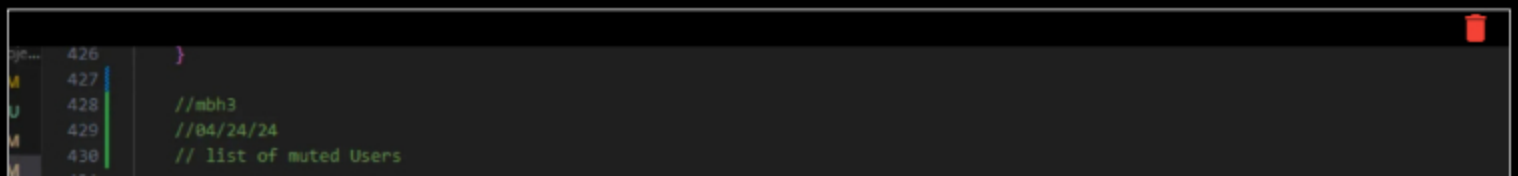
Small Medium Large



mute list

Checklist Items (1)

#1 ServerThread should have a list of who they muted



```
    private void saveMuteListToFile() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(clientName + "_muteList.csv"))) {
            for (String mutedClient : muteList) {
                writer.write(mutedClient + ",");
                writer.newLine();
            }
        } catch (IOException e) {
            logger.log(Level.SEVERE, msg:"Error writing to mute list file", e);
        }
    }

    private void loadMuteListFromFile() {
        try (BufferedReader reader = new BufferedReader(new FileReader(clientName + "_muteList.csv"))) {
            String[] mutedClients = reader.readLine().split(regex:",");
            Collections.addAll(muteList, mutedClients);
        } catch (IOException e) {
        }
    }

    public boolean isRecipientMuted(long clientId) {
        ServerThread targetClient = currentRoom.findClientById(clientId);
        if (targetClient != null) {
            String targetUsername = targetClient.getClientName();
            return muteList.contains(targetUsername);
        }
        return false;
    }
```

muted

Checklist Items (3)

#1 ServerThread should have a list of who they muted

#5 Include ucid and date comment

#6 Clearly caption screenshots

```
                sendMessage(getClientId(), message:"Currently not in a room");
            }
        }

        private void processUnmuteCommand(String message) {
            String targetUsername = message.substring(beginIndex:7).trim();

            if (currentRoom != null) {
                ServerThread targetClient = currentRoom.findClientByName(targetUsername);

                if (targetClient != null) {
                    if (!muteList.contains(targetUsername)) {
                        sendMessage(getClientId(), targetUsername + " is not muted");
                    } else {
                        muteList.remove(targetUsername);        You, 10 minutes ago • Uncommitted changes
                        saveMuteListToFile();
                        targetClient.sendMessage(getClientId(), getClientName() + " unmuted you");
                        sendMessage(getClientId(), "You unmuted " + targetUsername);
                    }
                } else {
                    sendMessage(getClientId(), "User " + targetUsername + " not found in the room");
                }
            } else {
                sendMessage(getClientId(), message:"Currently not in a room");
            }
        }

        public boolean isMuted() {
            return isMuted;
        }
        public void setMuted(boolean isMuted) {
            this.isMuted = isMuted;
        }
```

code

Checklist Items (4)

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

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

#5 Include ucid and date comment

#6 Clearly caption screenshots

```
public class ServerThread extends Thread {

    ///mhb3
    //04/24/24
    // mute and unmute

    private void processMuteCommand(String message) {
        String targetUsername = message.substring(beginIndex:5).trim();

        if (currentRoom != null) {
            ServerThread targetClient = currentRoom.findClientByName(targetUsername);

            if (targetClient != null) {
                if (muteList.contains(targetUsername)) {
                    sendMessage(getClientId(), targetUsername + " is already muted");
                } else {
                    muteList.add(targetUsername);
                    saveMuteListToFile();
                    targetClient.sendMessage(getClientId(), getClientName() + " muted you");
                    sendMessage(getClientId(), "You muted " + targetUsername);
                }
            } else {
                sendMessage(getClientId(), "User " + targetUsername + " not found in the room");
            }
        } else {
            sendMessage(getClientId(), message:"Currently not in a room");
        }
    }
```

checks

Checklist Items (5)

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

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

#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 mute and unmute features are pretty straightforward. Each ServerThread which represents a connection to a user has a list that keeps track of who is muted. When a user sends a mute command followed by a username, my code first checks if the user is in the same room. If the targeted user is there, and not already muted, I add them to the mutedList and send a confirmation message back to the sender. If the sender tries to mute someone who's already muted or not in the room, they get a message explaining what happened.

For unmuting, the process is similar but in reverse. I look up the username in the mutedList, and if found, I remove it, which effectively un-mutes that user for the sender. Just like with muting, the sender receives a confirmation message. When it comes to handling messages, before sending any message either a public chat or a private one I check the mutedList. If the sender is muted by the receiver, the message isn't sent to that particular receiver. This way, the mute feature controls whether messages are received based on who has been muted by whom.

COLLAPSE

Misc (1 pt.)

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/Mohammadh222/mbh3-IT114-004/compare/Milestone3?expand=1

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

Working on this chat application has been quite a journey, with its fair share of challenges and learning curves. Initially, things weren't going smoothly at all. Both the server and the client just wouldn't run; they kept throwing errors every time I tried to build the project, which was frustrating. No matter what I attempted, it seemed like I just couldn't join the

connection panel, leaving me stumped and a bit disheartened. Beyond just the connection issues, I was also struggling with the HTML formatting. The messages weren't displaying properly, and the HTML codes I incorporated for text styling were not being rendered as they should have been. It was like every piece of the puzzle refused to fit, and nothing was working right from the start. Despite these challenges, I kept at it, troubleshooting and learning from each error. Each issue taught me something new about how the server communicates with the clients, how connections are managed, and how important it is for the code to be precise to ensure messages are formatted correctly. It was a process, but one that was rich with lessons that made me a more capable programmer.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

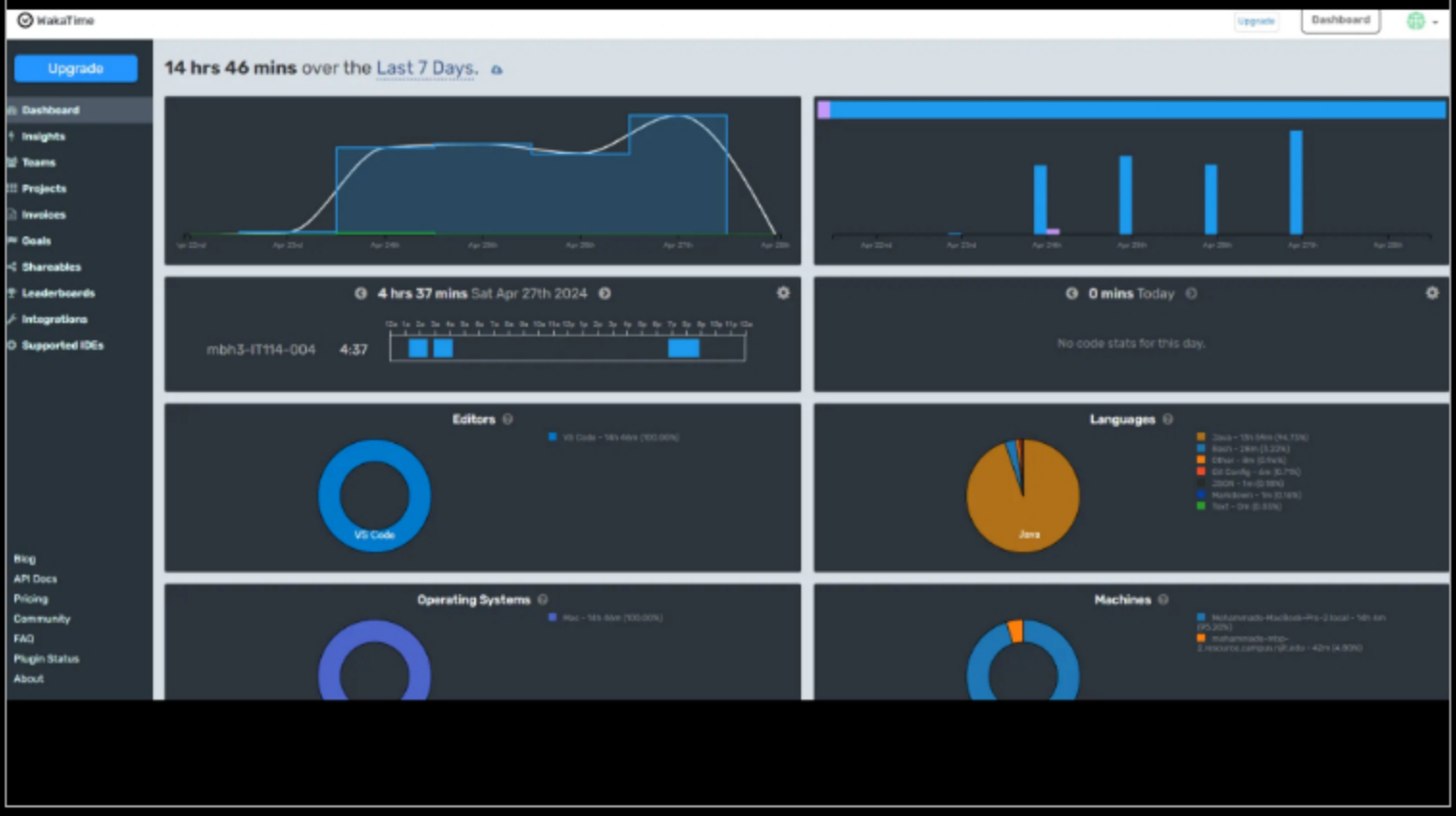
Details:

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



wakatime

End of Assignment