Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-002-S2024/it114-chatroom-milestone-3-2024/grade/mth39

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

Submissions:

Submission Selection

1 Submission [active] 4/14/2024 3:51:37 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

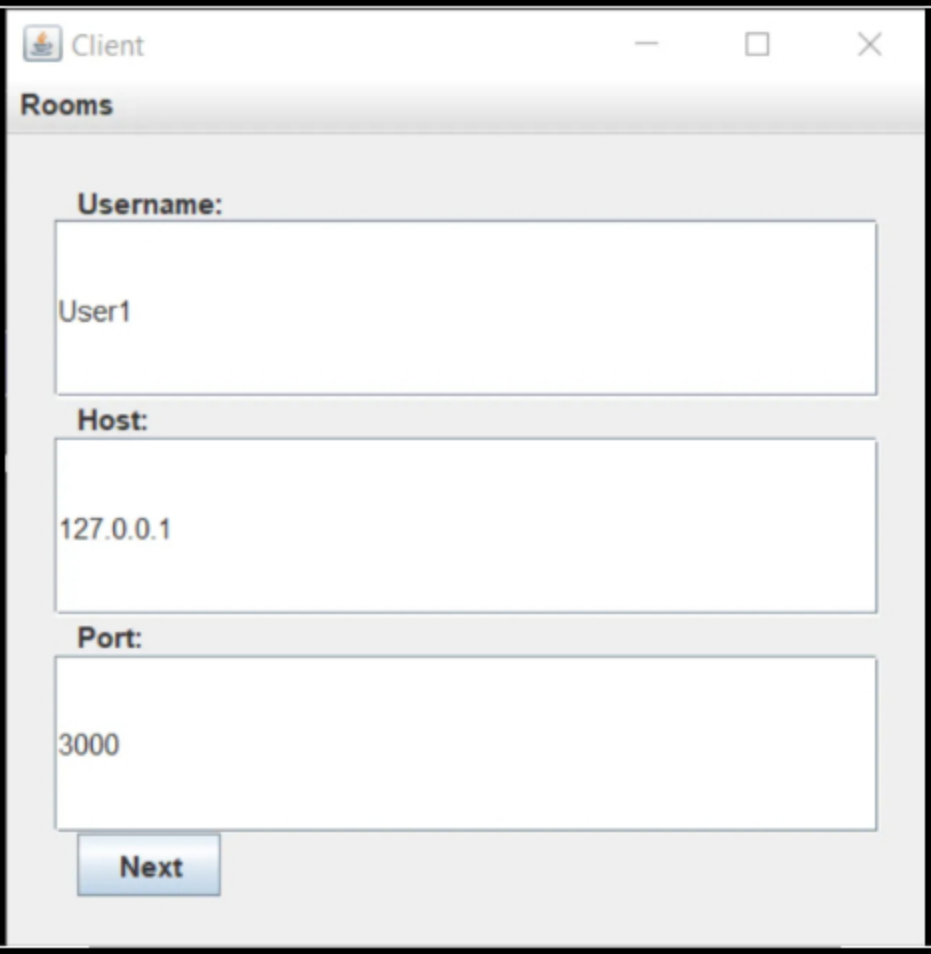
#	Points	Details

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

Task Screenshots:

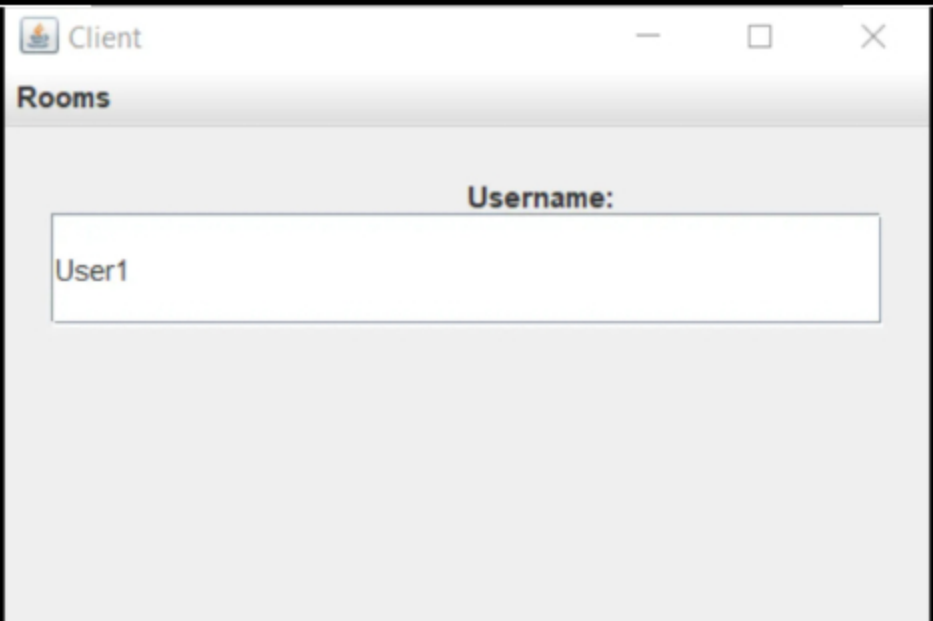
Gallery Style: Large View

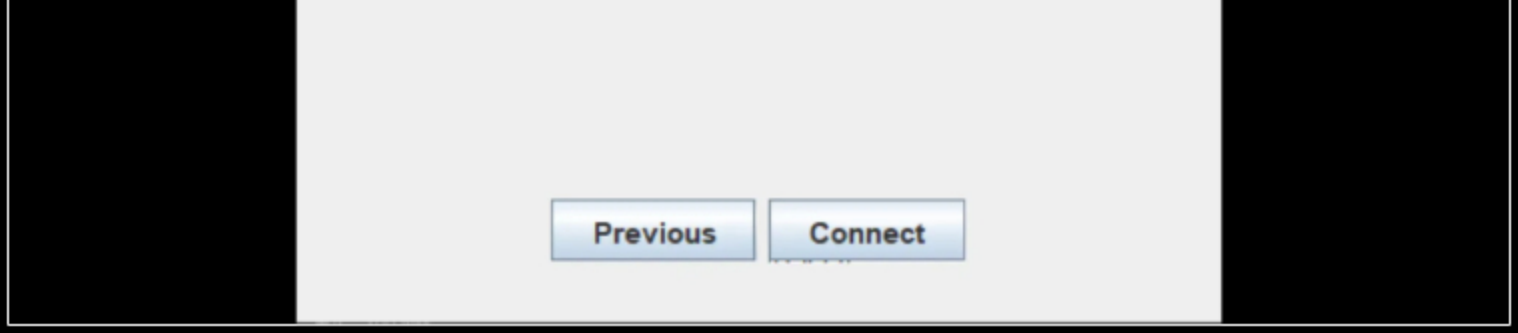
Small Medium Large



Screenshot showing Connection Panel

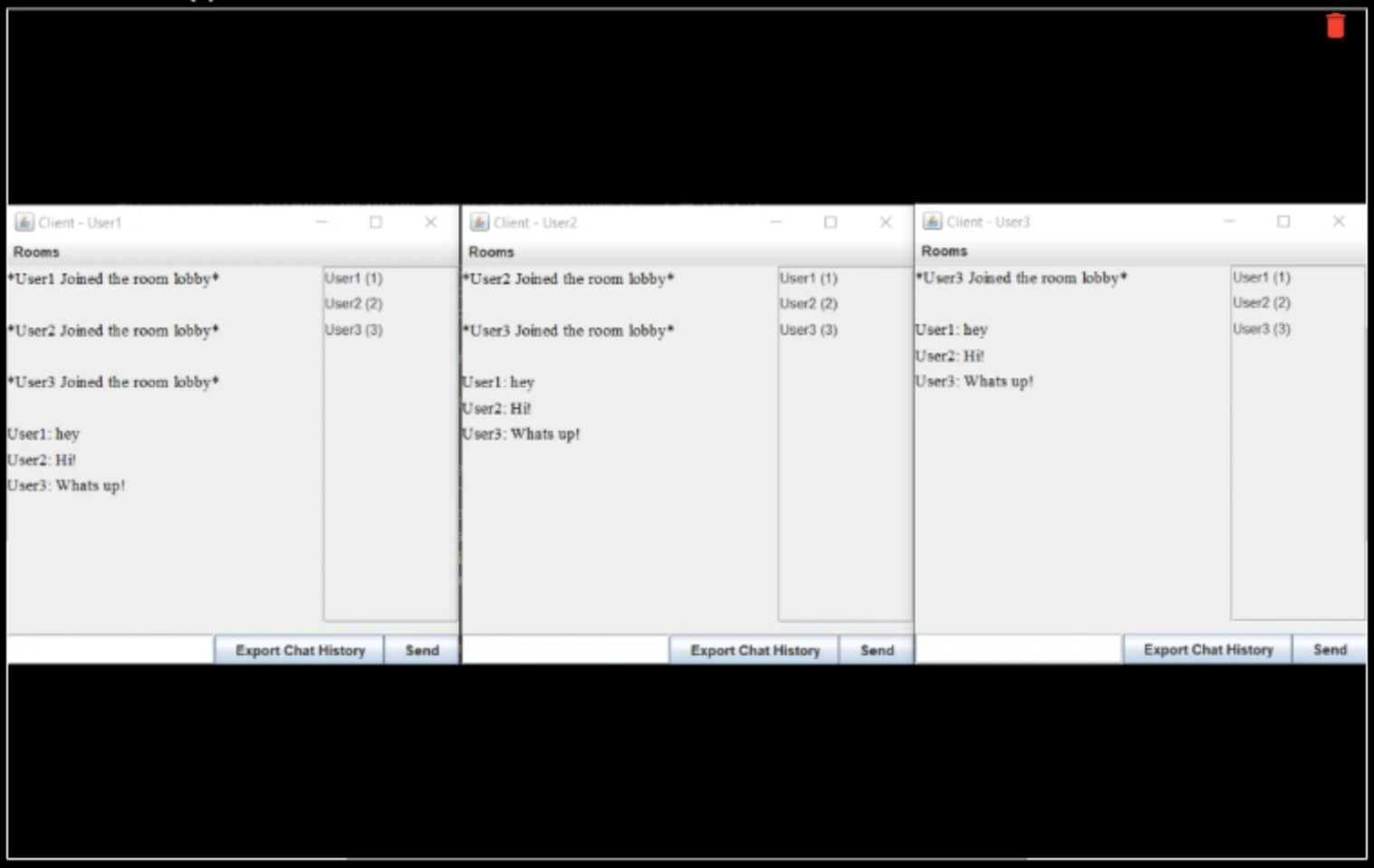
Checklist Items (0)





Screenshot showing User Details Panel

Checklist Items (0)



Screenshot showing Chat Panel

Checklist Items (0)

Formatting (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing flip and roll commands

Checklist

*The checkboxes are for your own tracking

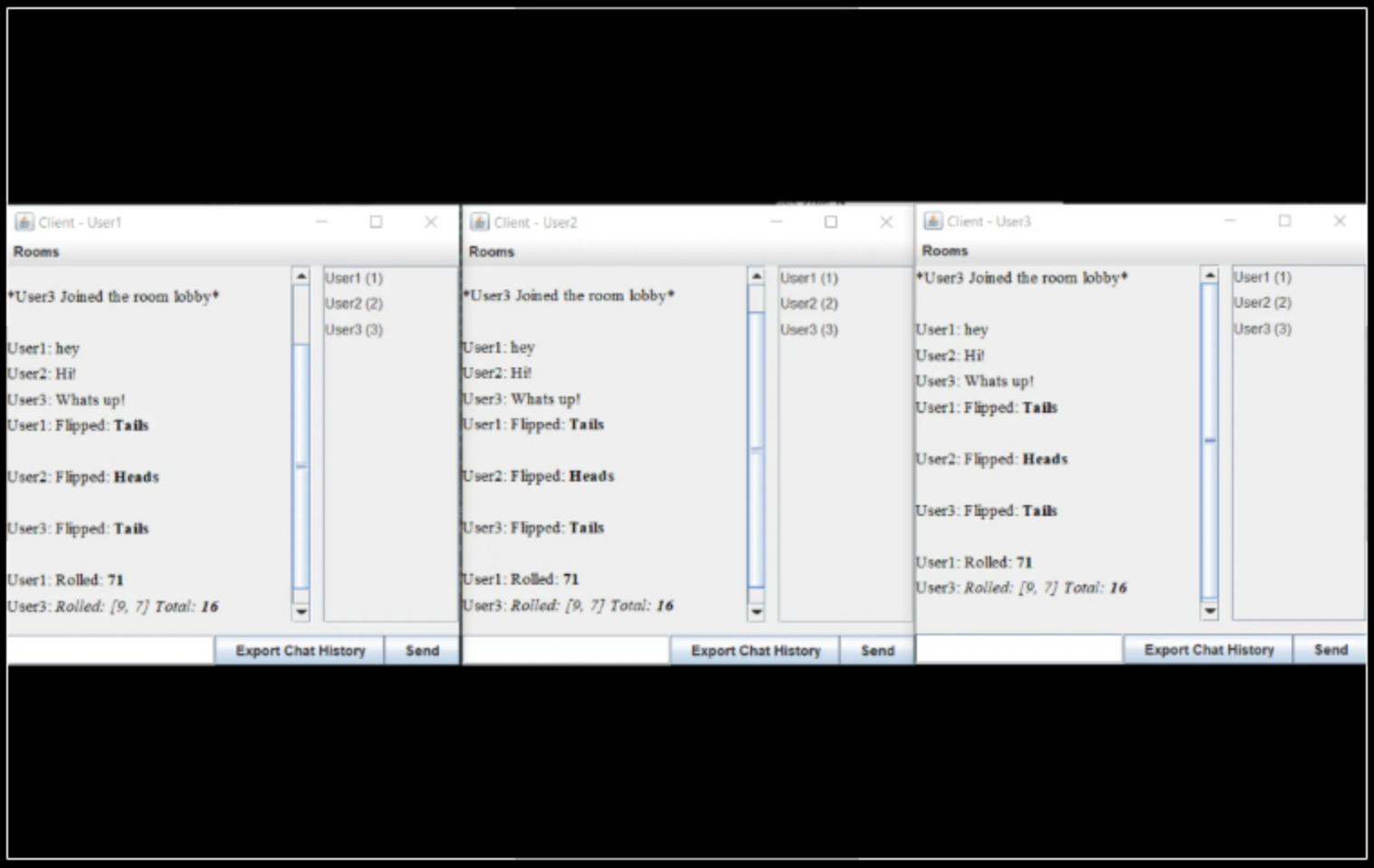
#	Points	Details
☐ #1	1	Flip output in a different format than normal messages

☐ #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



Screenshot showing "/flip" command output, "/roll 100" command output, and "/roll 2d10" output that differ from plain text.

Checklist Items (0)

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

Screenshot showing the output for underlined, bold, italic, red, green, blue formatted text and combinations of all four formats applied at once.

Checklist Items (0)

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

```
private void processRollCommand(ServerThread sender, String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    // Remove the command prefix
    String rollCommand = message;


    // Check if it's in Format 1: /roll # or Format 2: /roll #d#
    Pattern format1Pattern = Pattern.compile("(\\d+)");
    Pattern format2Pattern = Pattern.compile("(\\d+)d(\\d+)");

    Matcher format2Matcher = format2Pattern.matcher(rollCommand);
    Matcher format1Matcher = format1Pattern.matcher(rollCommand);

    if (format2Matcher.matches()) {
        int numberOfDice = Integer.parseInt(format2Matcher.group(1));
        int sides = Integer.parseInt(format2Matcher.group(2));
        int total = 0;
        StringBuilder result = new StringBuilder("Rolled: [");

        for (int i = 0; i < numberOfDice; i++) {
            int roll = (int) (Math.random() * sides) + 1;
            result.append(roll);

            if (i < numberOfDice - 1) {
                result.append(", ");
            }

            total += roll;
        } <- #284-293 for (int i = 0; i < numberOfDice; i++)

        result.append("] Total: <b>").append(total).append("</b>");
        sendMessage(sender, ("<i>" + result.toString() + "<i>"));
    } else if (format1Matcher.matches()) {
        int upperBound = Integer.parseInt(format1Matcher.group(1));
        int result = (int) (Math.random() * upperBound) + 1;
        sendMessage(sender, "Rolled: <b>" + result + "</b>");
    } else {
        sendMessage(sender, "<i>Invalid roll command. Please use /roll # or /roll #d#.</i>");
    }
} <- #266-304 private void processRollCommand(ServerThread sender, String m...
private void processFlipCommand(ServerThread sender) { // UCID: mth39, Date: 04/17/24, Milestone 3
    // Simulate a coin flip
    String result = (Math.random() < 0.5) ? "Heads" : "Tails";
    sendMessage(sender, "Flipped: <b>" + result + "</b>");
} // UCID: mth39, Date: 04/17/24, Milestone 3 <- #305-309 private void processFlipCommand(ServerThread sender)
```

Screenshot of code for flip and roll generation

Checklist Items (0)

```
private String processBold(String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    System.out.println("Source Message (Bold): " + message);
    // Print the original message for debugging


    // Define the regular expression pattern for detecting bold formatting
    Pattern pattern = Pattern.compile(BOLD_REGEX);
    // Create a matcher object to find matches in the message
    Matcher matcher = pattern.matcher(message);


    // Iterate through all matches in the message
    while (matcher.find()) {
        // Create bold-formatted text using the matched content
        String boldText = "<b>" + matcher.group(1) + "</b>";
        // Replace the original matched content with the formatted text
        message = message.replace(matcher.group(0), boldText);
    } <- #373-378 while (matcher.find())


    // Print the formatted message for debugging
    System.out.println("Formatted Message (Bold): " + message);
    return message;
} // UCID: mth39, Date: 04/17/24, Milestone 3 <- #361-384 private String processBold(String message)


private String processItalics(String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    // Print the original message for debugging
    System.out.println("Source Message (Italics): " + message);
    // Define the regular expression pattern for detecting italics formatting
    Pattern pattern = Pattern.compile(ITALICS_REGEX);
    // Create a matcher object to find matches in the message
    Matcher matcher = pattern.matcher(message);


    // Iterate through all matches in the message
    while (matcher.find()) {
        // Create italics-formatted text using the matched content
        String italicText = "<i>" + matcher.group(1) + "</i>";
        // Replace the original matched content with the formatted text
        message = message.replace(matcher.group(0), italicText);
    } <- #397-402 while (matcher.find())
```

```
    // Print the formatted message for debugging
    System.out.println("Formatted Message (Italics): " + message);
    return message;
} // UCID: mth39, Date: 04/17/24, Milestone 3 <- 8387-406 private String processItalics(String message)
```

Screenshot showing text formatting logic. (Part 1)

Checklist Items (0)

```
private String processUnderline(String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    System.out.println("Source Message (Underline): " + message);
    Pattern pattern = Pattern.compile(UNDERLINE_REGEX);
    Matcher matcher = pattern.matcher(message);


    while (matcher.find()) {
        String underlineText = "<u>" + matcher.group(1) + "</u>";
        message = message.replace(matcher.group(0), underlineText);
    }


    System.out.println("Formatted Message (Underline): " + message);
    return message;
} // UCID: mth39, Date: 04/17/24, Milestone 3 <- 8409-423 private String processUnderline(String message)


    private String processColor(String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    System.out.println("Source Message (Color): " + message);


    message = message.replace("[r", "<font color=red>").replace("r]","</font>");
    message = message.replace("[g", "<font color=green>").replace("g]","</font>");
    message = message.replace("[b", "<font color=blue>").replace("b]","</font>");

    System.out.println("Formatted Message (Color): " + message);
    return message;
} // UCID: mth39, Date: 04/17/24, Milestone 3 <- 8426-436 private String processColor(String message)


private String formatMessage(String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    System.out.println("Source Message: " + message);
    message = processBold(message);
    message = processItalics(message);
    message = processUnderline(message);
    message = processColor(message);


    System.out.println("Formatted Message: " + message);
    return message;
} // UCID: mth39, Date: 04/17/24, Milestone 3 <- 8439-449 private String formatMessage(String message)


public void processTextFormatting(ServerThread sender, String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    String formattedMessage = formatMessage(message);
    sendMessage(sender, formattedMessage);
} // UCID: mth39, Date: 04/17/24, Milestone 3
```

Screenshot showing text formatting logic. (Part 2)

Checklist Items (0)

COLLAPSE

Task #4 - Points: 1

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

Details:
Note each scenario

Response:

- Clients send commands such as roll, flip, or formatted messages to the server.
- The server processes these commands or messages, performs the necessary logic (e.g., generating random numbers or applying formatting), and then broadcasts the results to all clients in the room.
- The clients receive the broadcasted results and update their displays accordingly.

Private Message with @ (2 pts.)

^COLLAPSE ^

^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshots demoing private message

Checklist

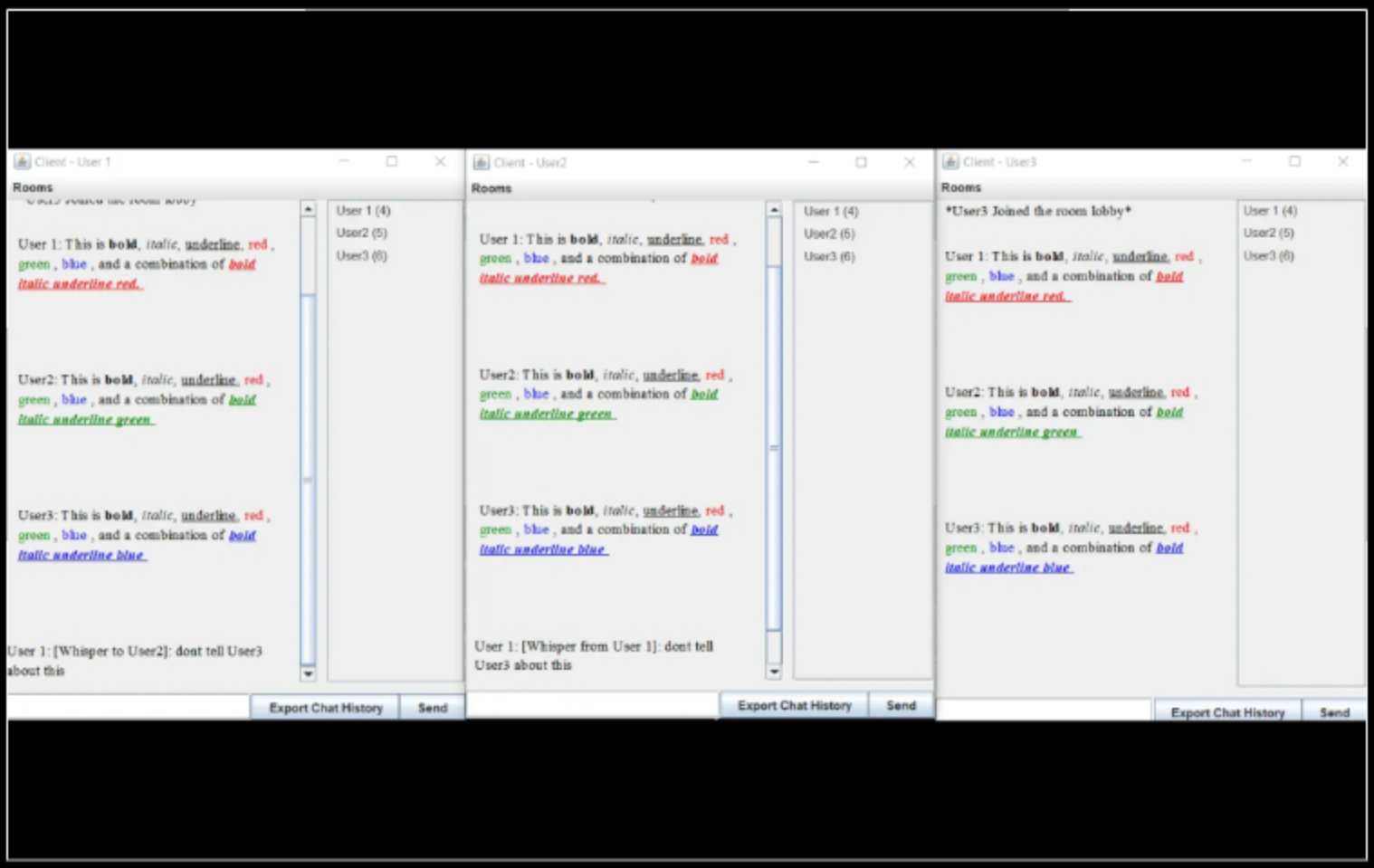
*The checkboxes are for your own tracking

#	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



Screenshot showing private message demo where User1 successfully whispers to User2 and User3 does not see the private message.

Checklist Items (0)

^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

```
        } else if (message.startsWith("@")) { // UCID: mth39, Date: 04/17/24, Milestone 3
            // Whisper command
            String[] parts = message.split(" ", 2);
            if (parts.length == 2) {
                String targetUsername = parts[0].substring(1); // Remove the "@" symbol
                String whisperMessage = parts[1];
                sendWhisperMessage(client, targetUsername, whisperMessage);
                wasCommand = true;
            } else {
                // Invalid whisper command
                client.sendMessage(Constants.DEFAULT_CLIENT_ID, "Invalid whisper command format.");
            }
        } <- #175-187 else if (message.startsWith("@"))
    } catch (Exception e) {
        e.printStackTrace();
    }

    return wasCommand;
}// UCID: mth39, Date: 04/17/24, Milestone 3 <- #130-193 private boolean processCommands(String message, ServerThread ...
```

Screenshot showing the code relevant to the functionality of the whisper command (Part 1)

Checklist Items (0)

```
private void sendWhisperMessage(ServerThread sender, String targetUsername, String message) { // UCID: mth39, Date: 04/17/24, Milestone 3
    // Find the target ServerThread
    ServerThread targetClient = findClientByUsername(targetUsername);

    if (targetClient != null) {
        // Send the whisper message to both sender and target
        sender.sendMessage(sender.getClientId(), "[Whisper to " + targetUsername + "]: " + message);
```

```
        } else {
            // Target user not found
            sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "User '" + targetUsername + "' not found.");
        }
    } // UCID: mth39, Date: 04/17/24, Milestone 3 <- #242-254 private void sendWhisperMessage(ServerThread sender, String t...

    private ServerThread findClientByUsername(String username) { // UCID: mth39, Date: 04/17/24, Milestone 3
        for (ServerThread client : clients) {
            if (client.getClientName().equalsIgnoreCase(username)) {
                return client;
            }
        } <- #257-261 for (ServerThread client : clients)
        return null; // Target user not found
    } // UCID: mth39, Date: 04/17/24, Milestone 3 <- #256-263 private ServerThread findClientByUsername(String username)
```

Screenshot showing the code relevant to the functionality of the whisper command (Part 2)

Checklist Items (0)

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:

In the provided code, the whisper feature checks if a message starts with the "@" symbol. If it does, the message is treated as a whisper command. The command is then split into two parts: the target username and the actual whisper message. The target username is extracted by removing the "@" symbol, and the message is sent using the sendWhisperMessage method.

The sendWhisperMessage method first attempts to find the target user's ServerThread by calling the findClientByUsername method. If the target user is found, the sender and target both receive a formatted whisper message, indicating the source and destination of the message. If the target user is not found, an error message is sent to the sender, informing them that the specified user was not found.

This implementation ensures that only the sender and the intended whisper target receive the message, maintaining privacy for the whisper functionality. The findClientByUsername method iterates through the list of clients in the room to locate the target user based on the provided username, and the sendWhisperMessage method facilitates the communication between the sender and the sender's target by sending appropriately formatted messages.

Mute/Unmute Users (3 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots demoing feature working

Checklist

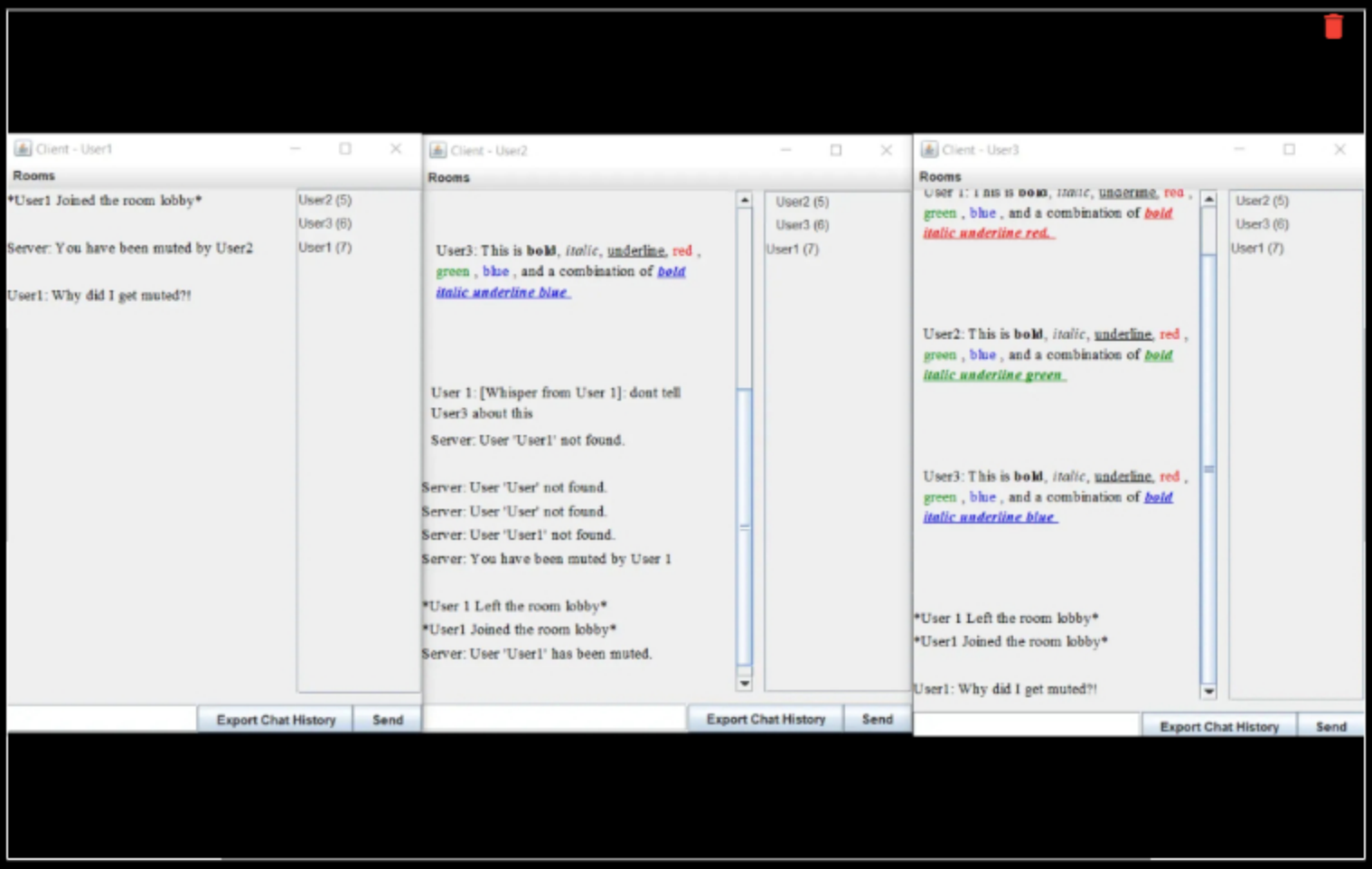
*The checkboxes are for your own tracking

#	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:

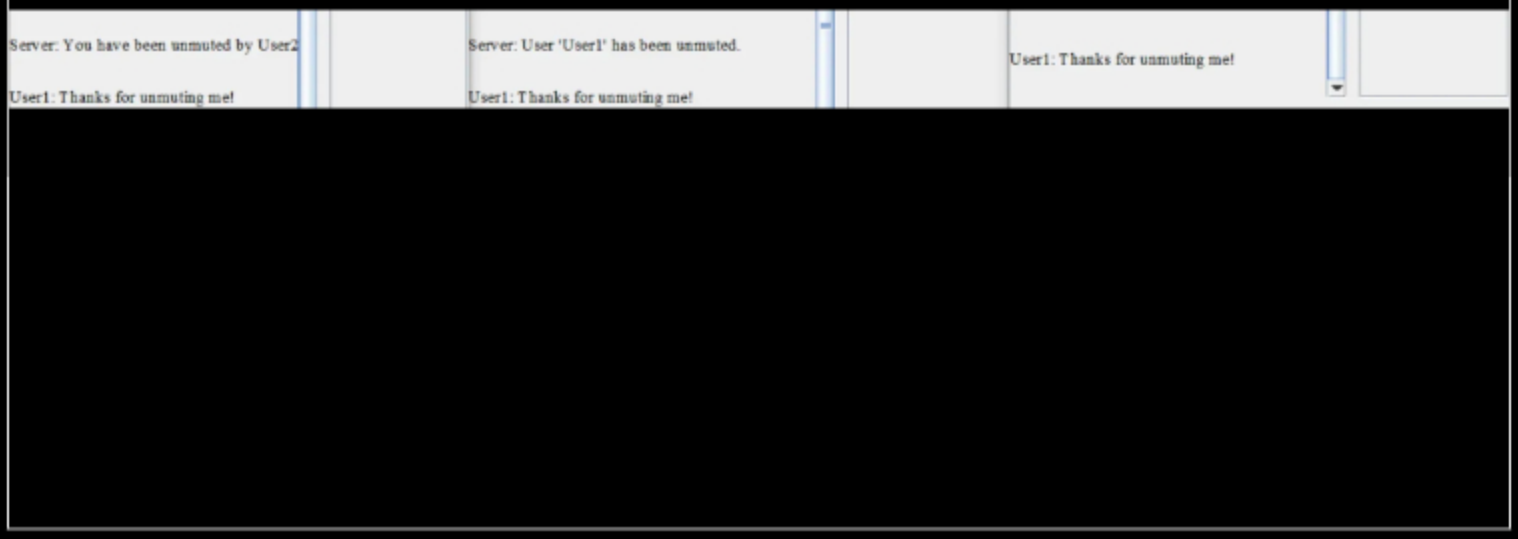
Gallery Style: Large View

Small Medium Large



Screenshot showing User2 muting User1 and both users getting a confirmation that User1 was muted by User2. User1 then sends a message that User2 does not see but User3 does.

Checklist Items (0)



Screenshot showing User2 unmuting User1 and User1's messages going through to User2 again.

Checklist Items (0)

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

```
public boolean isMuted(String username) { // UCID: mth39, Date: 04/17/24, Milestone 3
    return muteList.contains(username);
}

public void mute(String username) {
    if (!muteList.contains(username)) {
```

```
        muteList.add(username);
        saveMuteListToFile();
    }
} <- #170-175 public void mute(String username)

public void unmute(String username) {
    muteList.remove(username);
    saveMuteListToFile();
}
```

Screenshot showing the relevant code/methods within the ServerThread.java file.

Checklist Items (0)

```
                case "mute":
                    String targetUsername = comm2[1];
                    processMuteCommand(client, targetUsername);
                    break;
                case "unmute":
                    targetUsername = comm2[1];
                    processUnmuteCommand(client, targetUsername);
                    break;
                default:
                    wasCommand = false;
                    break;
            } <- #142-174 switch (command)
        } else if (message.startsWith("@")) { // UCID: mth39, Date: 04/17/24, Milestone 3
            // Whisper command
            String[] parts = message.split(" ", 2);
            if (parts.length == 2) {
                String targetUsername = parts[0].substring(1); // Remove the "@" symbol
                String whisperMessage = parts[1];
                sendWhisperMessage(client, targetUsername, whisperMessage);
                wasCommand = true;
            } else {
                // Invalid whisper command
                client.sendMessage(Constants.DEFAULT_CLIENT_ID, "Invalid whisper command format.");
            }
        } <- #175-187 else if (message.startsWith("@"))
    } catch (Exception e) {
        e.printStackTrace();
    }

    return wasCommand;
}// UCID: mth39, Date: 04/17/24, Milestone 3 <- #130-193 private boolean processCommands(String message, ServerThread ...
```

Screenshot showing the relevant code/methods within the Room.java file (Part 1).

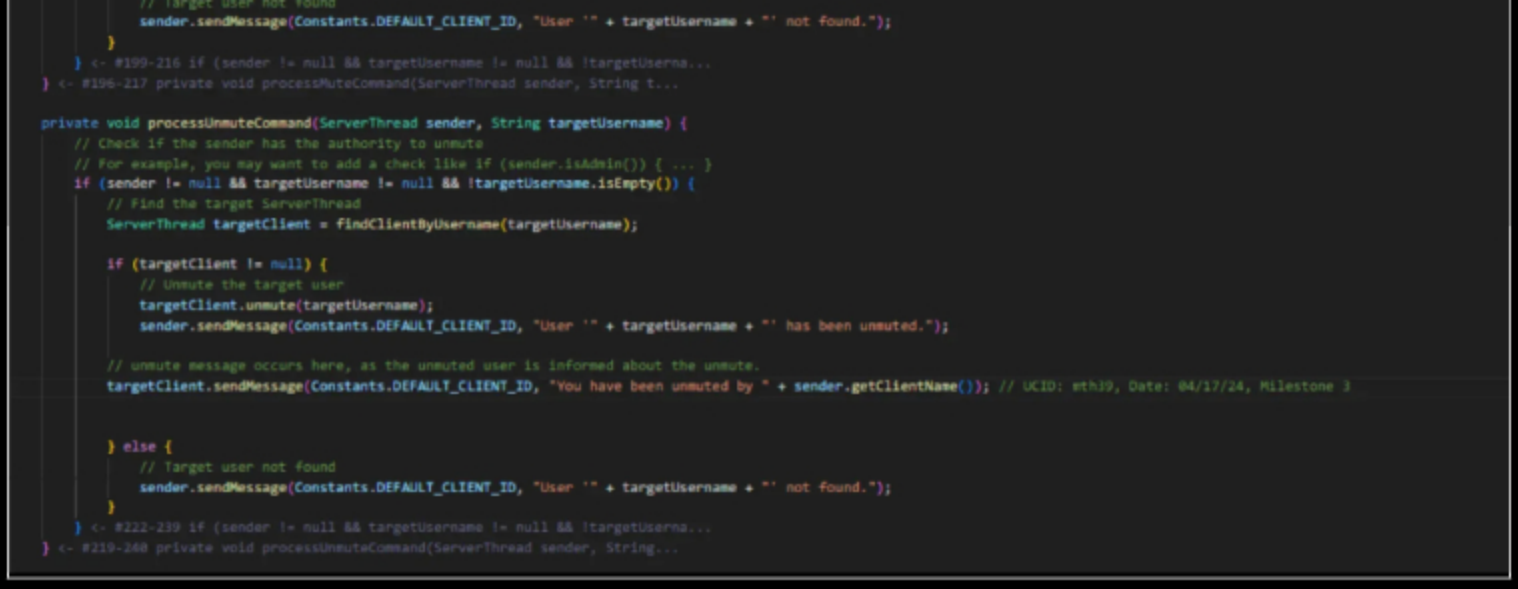
Checklist Items (0)

```
private void processMuteCommand(ServerThread sender, String targetUsername) {
    // Check if the sender has the authority to mute
    // For example, you may want to add a check like if (sender.isAdmin()) { ... }
    if (sender != null && targetUsername != null && !targetUsername.isEmpty()) {
        // Find the target ServerThread
        ServerThread targetClient = findClientByUsername(targetUsername);

        if (targetClient != null) {
            // Mute the target user
            targetClient.mute(targetUsername);
            sender.sendMessage(Constants.DEFAULT_CLIENT_ID, "User '" + targetUsername + "' has been muted.");

        // mute message occurs here, as the muted user is informed about the mute.
        targetClient.sendMessage(Constants.DEFAULT_CLIENT_ID, "You have been muted by " + sender.getClientName()); // UCID: mth39, Date: 04/17/24, Milestone 3


        } else {
```



Screenshot showing the relevant code/methods within the Room.java file (Part 2).

Checklist Items (0)

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/unmute functionality is implemented in both the Room.java and ServerThread.java classes. In the Room.java class, the relevant code processes mute and unmute commands received through messages. When a message is received, it checks if it starts with the "/mute" or "/unmute" command, if so, it extracts the target username and calls the appropriate methods in the ServerThread.java class to handle muting or unmuting.

In the ServerThread.java class, the isMuted, mute, and unmute methods manage the mute list. The isMuted method checks whether a given username is in the mute list. The mute method adds a username to the mute list, and the unmute method removes a username from the mute list.

The actual implementation of preventing muted users from sending messages is in the Room.java class. The sendMessage method is modified to check if the sender is muted before broadcasting a message. If the sender is muted, the message is not broadcasted to them, ensuring that muted users cannot send messages to the room. Additionally, this modification ensures that the muted sender still receives their own message, allowing them to see what they sent even if others cannot.

This collaboration between the two classes results in a comprehensive mute/unmute feature where commands are processed, and the mute list is managed in the ServerThread.java class, while the actual handling of message broadcasting to muted users is implemented in the Room.java class.

Misc (1 pt.)

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

Missing URL

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I did not encounter any issues during this assignment. This assignment was the hardest assignment yet for this class. Overall, I had a fun but tedious time implementing coding logic and debugging code to make my roll, flip, text formatting, muting, unmuting, and whispering commands work. I feel accomplished for the chatroom so far.

COLLAPSE

Task #3 - Points: 1

Text: WakaTime Screenshot

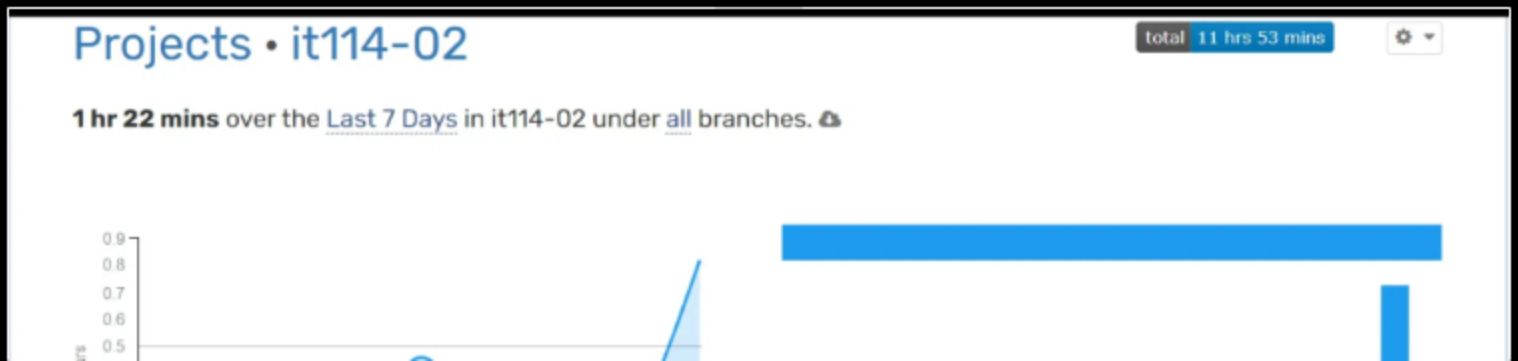
Details:

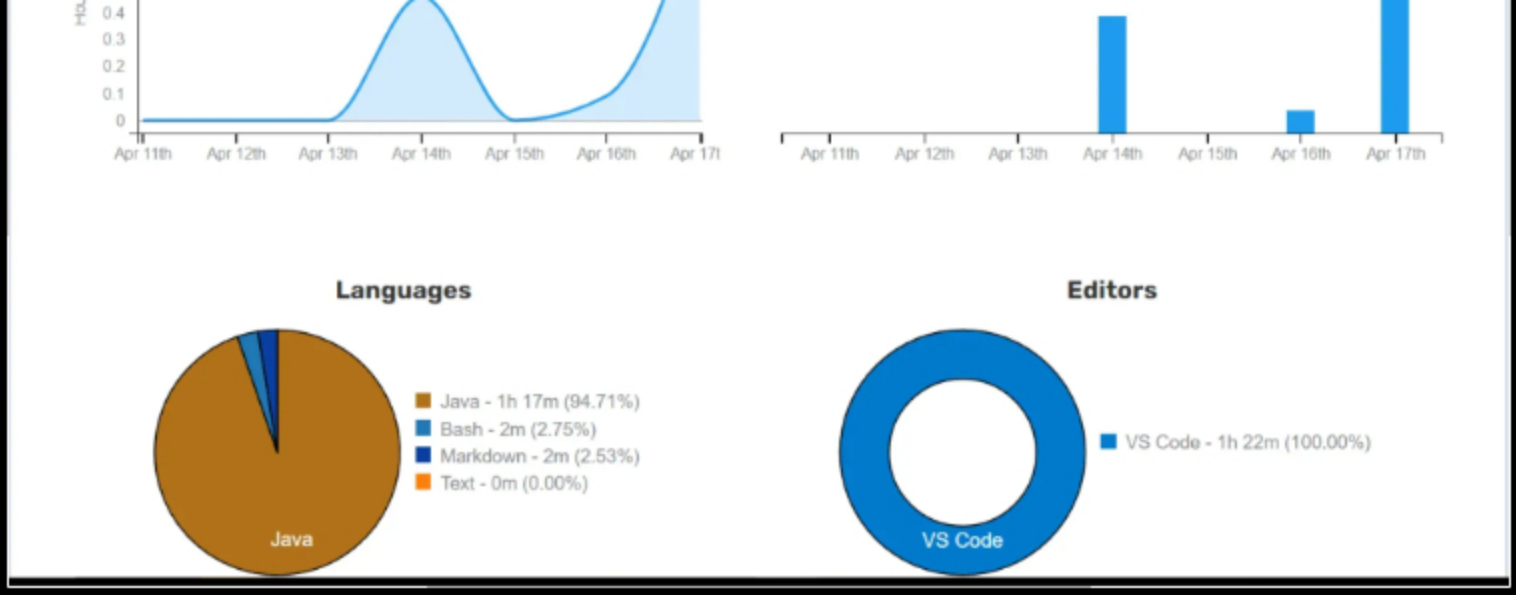
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved.

Task Screenshots:

Gallery Style: Large View

Small Medium Large





WakaTime screenshot showing time spent on "it114-002" repository in the past week.

End of Assignment