Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-450-M2024/it114-milestone-4-chatroom-2024-m24/grade/jah89

IT114-450-M2024 - [IT114] Milestone 4 Chatroom 2024 M24

Submissions:

Submission Selection

1 Submission [active] 7/27/2024 7:57:17 AM

Instructions

COLLAPSE

- Implement the Milestone 4 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145XizQG4/view
- Make sure you add your ucid/date as code comments where code changes are done
- All code changes should reach the Milestone4 branch
- Create a pull request from Milestone4 to main and keep it open until you get the output PDF from this assignment.
- Gather the evidence of feature completion based on the below tasks.
- Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
- Run the necessary git add, commit, and push steps to move it to GitHub
- Complete the pull request that was opened earlier
- Upload the same output PDF to Canvas

Branch name: Milestone4

Tasks: 7 Points: 10.00

● Features (9 pts.)

COLLAPSE

● COLLAPSE

Task #1 - Points: 3

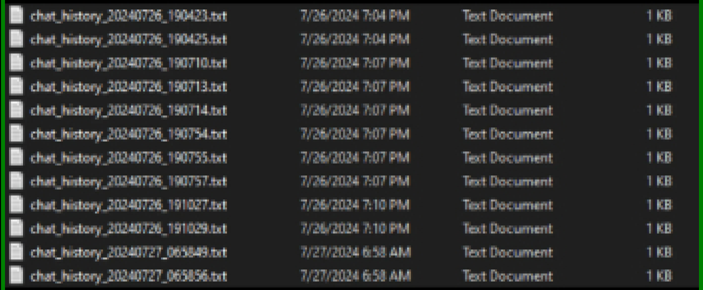
Text: Client can export chat history of their current session (client-side)

Details:

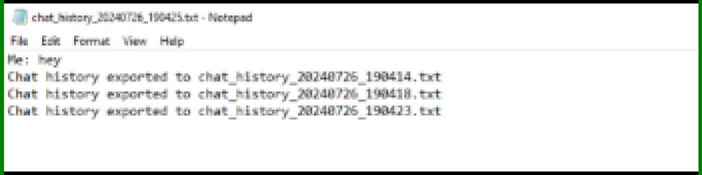
For this requirement it's not valid to have another list keep track of messages. The goal is to utilize the location where messages are already present.

This must be a client-side implementation.
A StringBuilder must be used for consolidation.
Screenshots of editors must have the frame title visible with your ucid and the client name.
Code screenshots must have ucid/data comments.

#1) Show a few examples of exported chat history (include the filename showing that there are multiple copies)



```
chat_history_20240726_190423.txt    7/26/2024 7:04 PM    Text Document    1 KB
chat_history_20240726_190425.txt    7/26/2024 7:04 PM    Text Document    1 KB
chat_history_20240726_190710.txt    7/26/2024 7:07 PM    Text Document    1 KB
chat_history_20240726_190713.txt    7/26/2024 7:07 PM    Text Document    1 KB
chat_history_20240726_190714.txt    7/26/2024 7:07 PM    Text Document    1 KB
chat_history_20240726_190754.txt    7/26/2024 7:07 PM    Text Document    1 KB
chat_history_20240726_190755.txt    7/26/2024 7:07 PM    Text Document    1 KB
chat_history_20240726_190757.txt    7/26/2024 7:07 PM    Text Document    1 KB
chat_history_20240726_191027.txt    7/26/2024 7:10 PM    Text Document    1 KB
chat_history_20240726_191029.txt    7/26/2024 7:10 PM    Text Document    1 KB
chat_history_20240727_065849.txt    7/27/2024 6:58 AM    Text Document    1 KB
chat_history_20240727_065856.txt    7/27/2024 6:58 AM    Text Document    1 KB
```



chat_history_20240726_190425.txt - Notepad

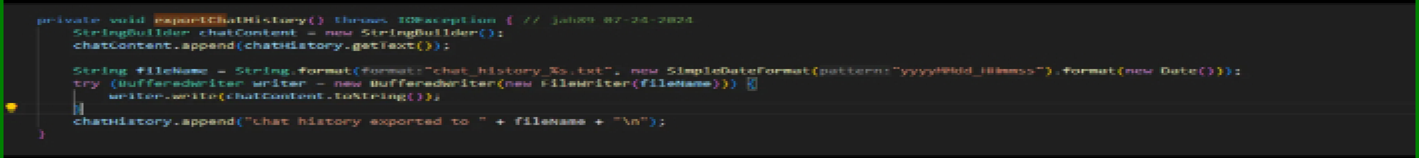
```
Me: hey
Chat history exported to chat_history_20240726_190414.txt
Chat history exported to chat_history_20240726_190418.txt
Chat history exported to chat_history_20240726_190423.txt
```

Caption (required) ✓

Describe/highlight what's being shown

Showing exported chat history

#2) Show the code related to building the export data (where the messages are gathered from, the StringBuilder, and the file generation)



```
private void exportChatHistory() throws IOException { // jah89 07-24-2024
    StringBuilder chatContent = new StringBuilder();
    chatContent.append(chatHistory.getText());

    String fileName = String.format(format:"chat_history_%s.txt", new SimpleDateFormat(pattern:"yyyyMMdd_HHmmss").format(new Date()));
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
        writer.write(chatContent.toString());
    }
    chatHistory.append("Chat history exported to " + fileName + "\n");
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing code related to building the export data

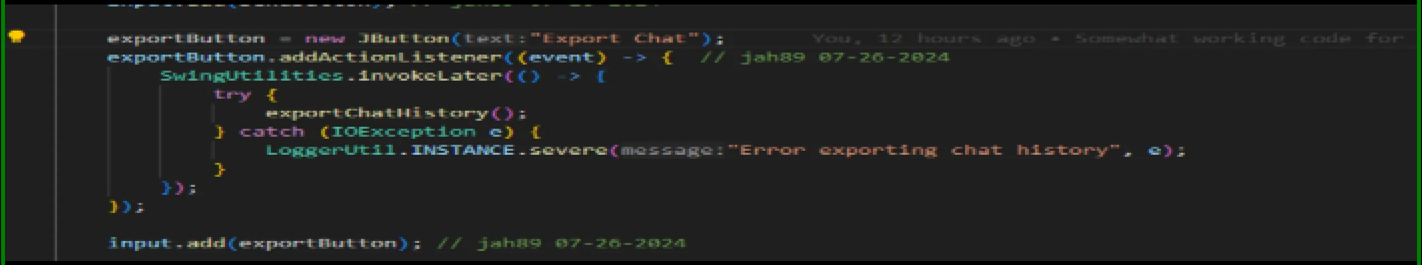
Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The first thing the code does is initialize StringBuilder to an object "chatContent" to handle the chat history. Then the text located in the jtextarea that contains the chat messages is appended to the chatContent stringbuilder. Then it creates a filename with the format with the name "chat_history" and then date with year month and day and it is saved to a .txt file. I used a BufferedWriter around the fileWriter to write the chat history into the generated file, this is done by putting it inside a try block to handle error and exceptions. After the chat history is exported there is a message to notify it was exported along with the file name it was saved under.

#3) Show the UI interaction that will trigger an export



```
exportButton = new JButton(text:"Export Chat");        You, 12 hours ago • Somewhat working code for
exportButton.addActionListener((event) -> {  // jah89 07-26-2024
    SwingUtilities.invokeLater(() -> {
        try {
            exportChatHistory();
        } catch (IOException e) {
            LoggerUtil.INSTANCE.severe(message:"Error exporting chat history", e);
        }
    });
});

input.add(exportButton); // jah89 07-26-2024
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the UI interaction that will trigger an export

Explanation (required) ✓

Explain where you put it any why

PREVIEW RESPONSE

I put this code inside the chatpanel class because it is responsible for creating and managing the chat interface and in this case the export button and the action listener. The line "exportButton = new JButton(text: "Export Chat");" created the button with the label export chat to clearly state the intention of the button. The actionlistener is attached to the button so when it is clicked the code inside the action listener is executed.

COLLAPSE

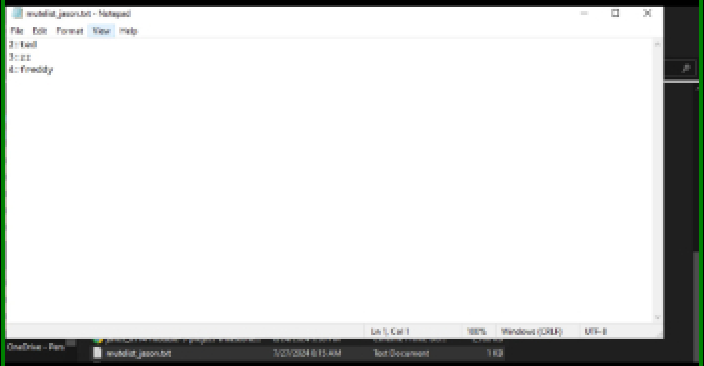
Task #2 - Points: 3

Text: Client's Mute List will persist across sessions (server-side)

Details:
This must be a server-side implementation.

Screenshots of editors must have the frame title visible with your ucid and the client name.
Code screenshots must have ucid/data comments.

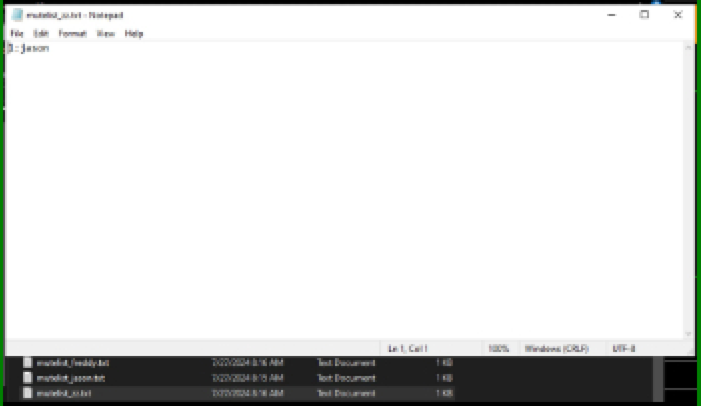
#1) Show multiple examples of mutelist files and their content (their names should have/include the user's client name)



mutelist_jason.txt - Notepad

```
2::Neil
3::zz
4::freddy
```

mutelist_jason.txt   7/27/2024 8:15 AM   Text Document   1 KB



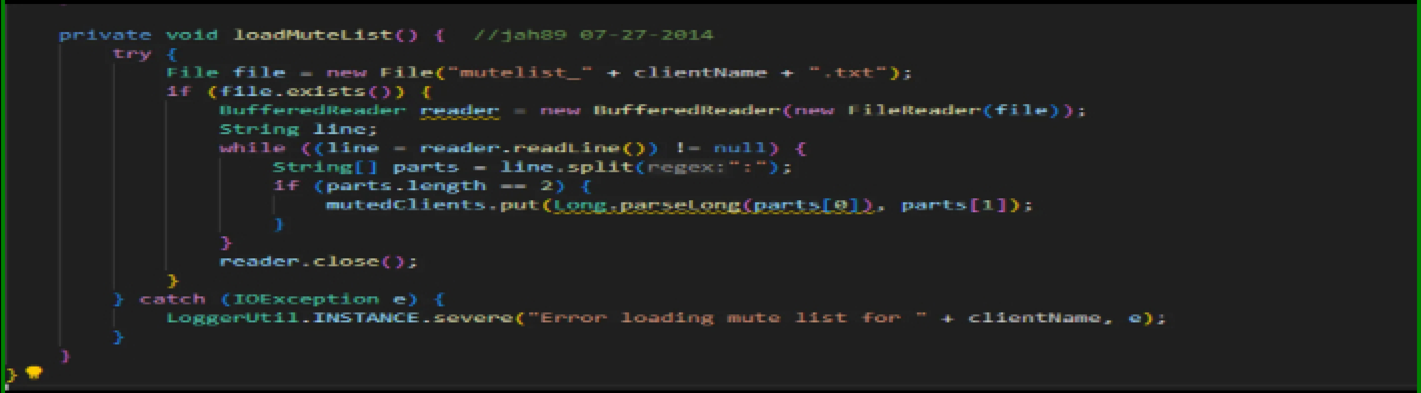
mutelist_zz.txt - Notepad

```
1::jason
```

mutelist_freddy.txt   7/27/2024 8:16 AM   Text Document   1 KB
mutelist_jason.txt   7/27/2024 8:15 AM   Text Document   1 KB
mutelist_zz.txt   7/27/2024 8:16 AM   Text Document   1 KB

Caption (required) ✓
Describe/highlight what's being shown
Showing mutelist files

#2) Show the code related to loading the mutelist for a connecting client (and logic that handles if there's no file)



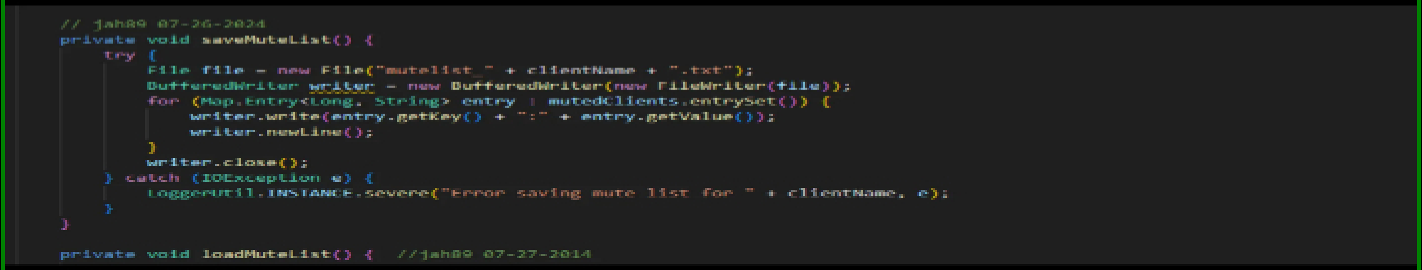
```
    private void loadMuteList() {  //jah89 07-27-2014
        try {
            File file = new File("mutelist_" + clientName + ".txt");
            if (file.exists()) {
                BufferedReader reader = new BufferedReader(new FileReader(file));
                String line;
                while ((line = reader.readLine()) != null) {
                    String[] parts = line.split(regex:":");
                    if (parts.length == 2) {
                        mutedClients.put(Long.parseLong(parts[0]), parts[1]);
                    }
                }
                reader.close();
            }
        } catch (IOException e) {
            LoggerUtil.INSTANCE.severe("Error loading mute list for " + clientName, e);
        }
    }
}
```

Caption (required) ✓

Describe/highlight what's being shown
Showing the code related to loading the mutelist

Explanation (required) ✓
Explain in concise steps how this logically works
PREVIEW RESPONSE
The first thing that happens in this code is a file object gets created with the name "mutelist + clientName + txt. This makes the file and makes sure its unique enough so each client has their own file which is easily found. Then it checks if the file exists use "file.exists() and if it doesnt exist it will exit the method. I then used BufferedReader to read the contents of the file with the FileReader inside to ready the specific file. Then a while loop reads the file using reader.readline and is stored in the variable "line". Then I split it into parts using split and if this array has a length of 2, it indicates the clientid and a client name and then it gets added to the muted list. If an exception occurs during this an error message is sent and logged.

#3) Show the code related to saving the mutelist whenever the list changes for a client



```
// jah89 07-26-2024
private void saveMuteList() {
    try {
        File file = new File("mutelist_" + clientName + ".txt");
        BufferedWriter writer = new BufferedWriter(new FileWriter(file));
        for (Map.Entry<Long, String> entry : mutedClients.entrySet()) {
            writer.write(entry.getKey() + ":" + entry.getValue());
            writer.newLine();
        }
        writer.close();
    } catch (IOException e) {
        LoggerUtil.INSTANCE.severe("Error saving mute list for " + clientName, e);
    }
}

private void loadMuteList() {  //jah89 07-27-2014
```

Caption (required) ✓
Describe/highlight what's being shown
Showing the code related to saving the mutelist

Explanation (required) ✓
Explain in concise steps how this logically works
PREVIEW RESPONSE
The first thing that happens in this code is a file object gets created with the name "mutelist + clientName + txt. This makes the file and makes sure its unique enough so each client has their own file which is easily found. Then it checks if the file exists use "file.exists() and if it doesnt exist it will exit the method. I then used BufferedReader to read the contents of the file with the FileReader inside to ready the specific file. Then a while loop reads the file using reader.readline and is stored in the variable "line". Then I split it into parts using split and if this array has a length of 2, it indicates the clientid and a client name and then it gets added to the muted list. If an exception occurs during this an error message is sent and logged.

COLLAPSE

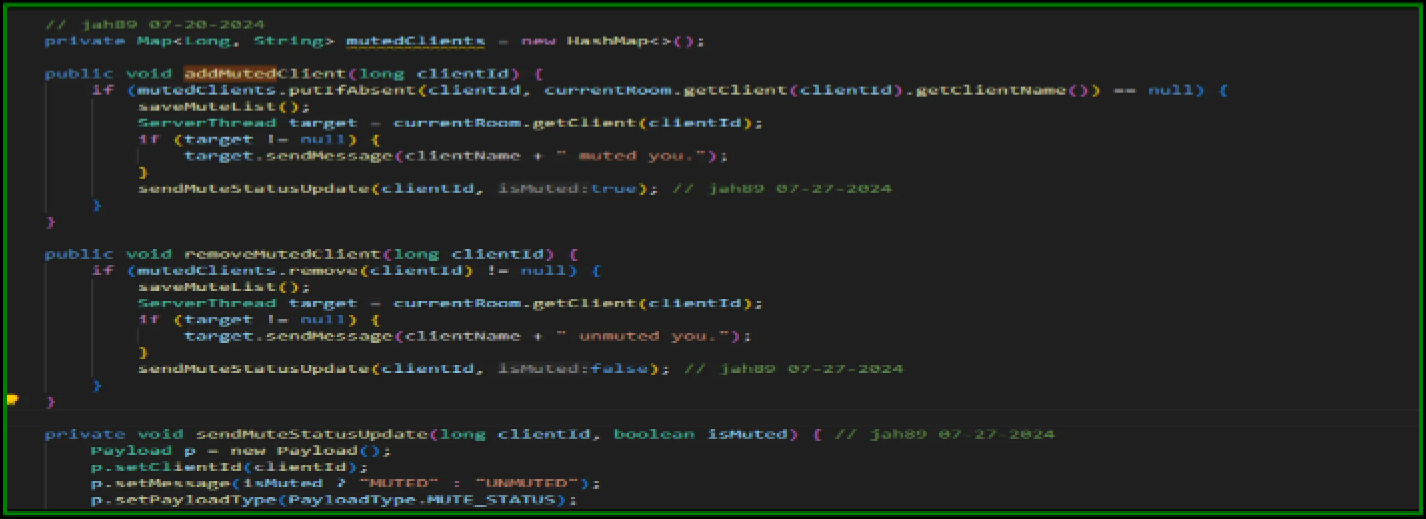
Task #3 - Points: 1

Text: Clients will receive a message when they get muted/unmuted by another user

Details:
Screenshots of editors must have the frame title visible with your ucid and the client name.
Code screenshots must have ucid/data comments.

I.e., /mute Bob followed by a /mute Bob should only send one message because Bob can only be muted once until they're unmuted. Similarly for /unmute Bob

#1) Show the code that generates the well formatted message only when the mute state changes (see notes in the details above)



```
// jah89 07-20-2024
private Map<Long, String> mutedClients = new HashMap<>();

public void addMutedClient(long clientId) {
    if (mutedClients.putIfAbsent(clientId, currentRoom.getClient(clientId).getClientName()) == null) {
        saveMuteList();
        ServerThread target = currentRoom.getClient(clientId);
        if (target != null) {
            target.sendMessage(clientName + " muted you.");
        }
        sendMuteStatusUpdate(clientId, isMuted:true); // jah89 07-27-2024
    }
}

public void removeMutedClient(long clientId) {
    if (mutedClients.remove(clientId) != null) {
        saveMuteList();
        ServerThread target = currentRoom.getClient(clientId);
        if (target != null) {
            target.sendMessage(clientName + " unmuted you.");
        }
        sendMuteStatusUpdate(clientId, isMuted:false); // jah89 07-27-2024
    }
}

private void sendMuteStatusUpdate(long clientId, boolean isMuted) { // jah89 07-27-2024
    Payload p = new Payload();
    p.setClientId(clientId);
    p.setMessage(isMuted ? "MUTED" : "UNMUTED");
    p.setPayloadType(PayloadType.MUTE_STATUS);
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the code that generates the well formatted message only when the mute state changes

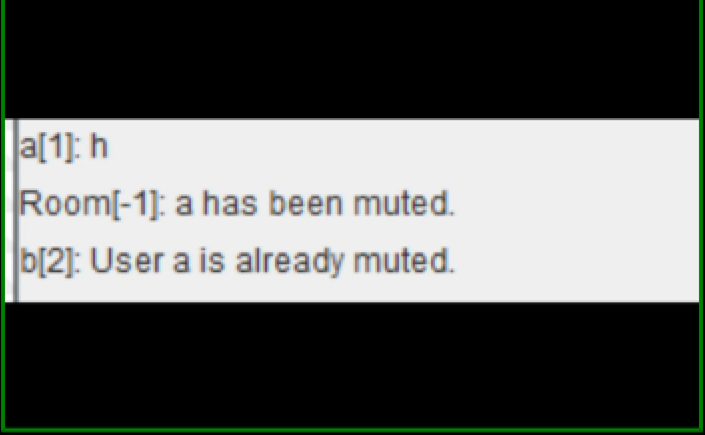
Explanation (required) ✓

Explain in concise steps how this logically works

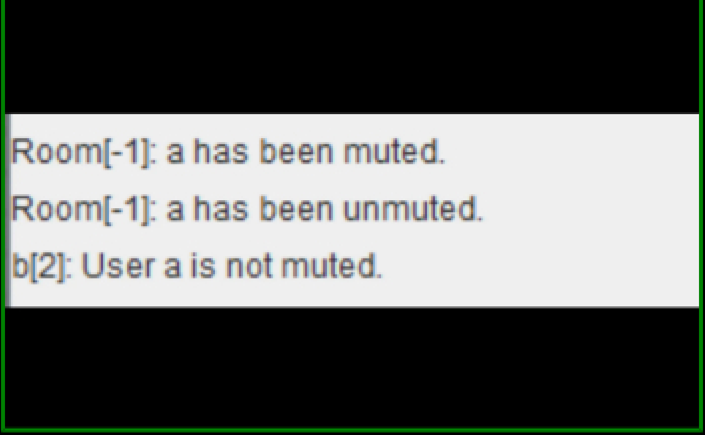
PREVIEW RESPONSE

The part of the code uses a hashmap to store the muted clients with their id as they key and the clients name as the value. Next it checks if the clientid is already in the mute clients map and if not it adds it to the map. Then it saves the mutelist to update the list to the file to make sure its up to date. If the target client is found in the current room it will send a message to notify that they have been muted. Then I use "sendMuteStatusUpdate" to update the clients mute status. This has the same process for unmuting a client as well. The last part of the code creates a payload object and sets the client id and message with muted or unmuted based on the ismuted flag.

#2) Show a few examples of this occurring and demonstrate that two mutes of the same user in a row generate only one message, do the same for unmute)



```
a[1]: h
Room[-1]: a has been muted.
b[2]: User a is already muted.
```



```
Room[-1]: a has been muted.
Room[-1]: a has been unmuted.
b[2]: User a is not muted.
```

Caption (required) ✓

Describe/highlight what's being shown

Showing examples of clients getting muted and unmuted

COLLAPSE

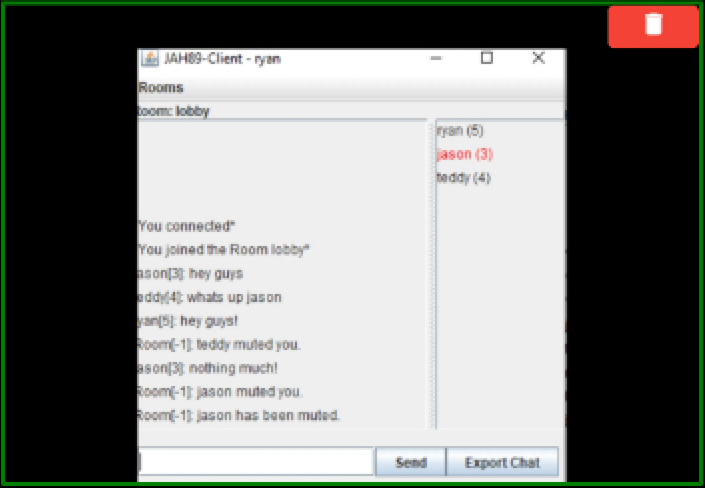
Task #4 - Points: 3

Text: The user list on the Client-side should update per the status of each user

Details:

Screenshots of editors must have the frame title visible with your ucid and the client name.
Code screenshots must have ucid/data comments.

#1) Show the UI for Muted users appear grayed out (or similar indication of your choosing) include a few examples showing it updates correctly when changing from mute/unmute and back



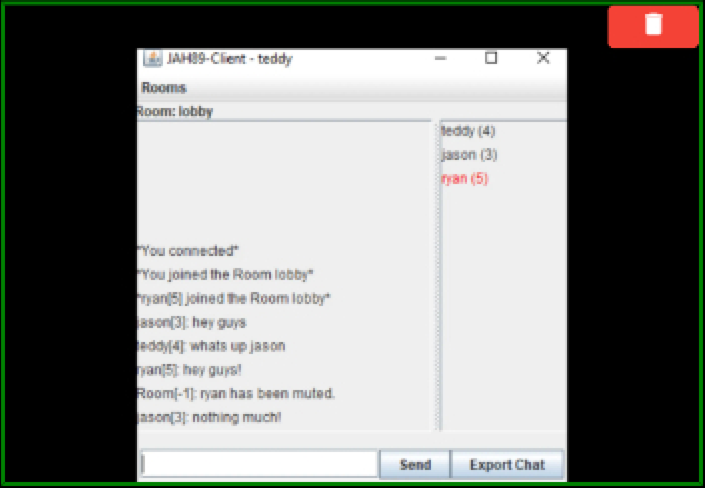
JAH89-Client - ryan

Rooms

Room: lobby

```
ryan (5)
jason (3)
teddy (4)
```

```
You connected*
You joined the Room lobby*
ason[3]: hey guys
eddy[4]: whats up jason
yan[5]: hey guys!
Room[-1]: teddy muted you.
ason[3]: nothing much!
Room[-1]: jason muted you.
Room[-1]: jason has been muted.
```



JAH89-Client - teddy

Rooms

Room: lobby

```
teddy (4)
jason (3)
ryan (5)
```

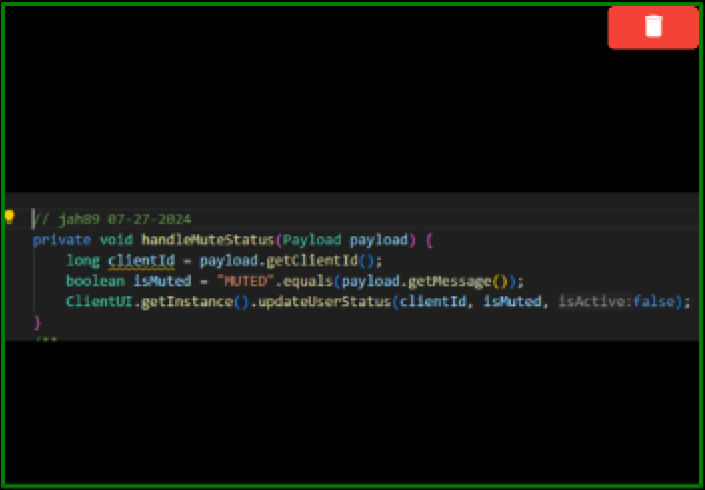
```
*You connected*
*You joined the Room lobby*
*ryan[5] joined the Room lobby*
jason[3]: hey guys
teddy[4]: whats up jason
ryan[5]: hey guys!
Room[-1]: ryan has been muted.
jason[3]: nothing much!
```

Caption (required) ✓

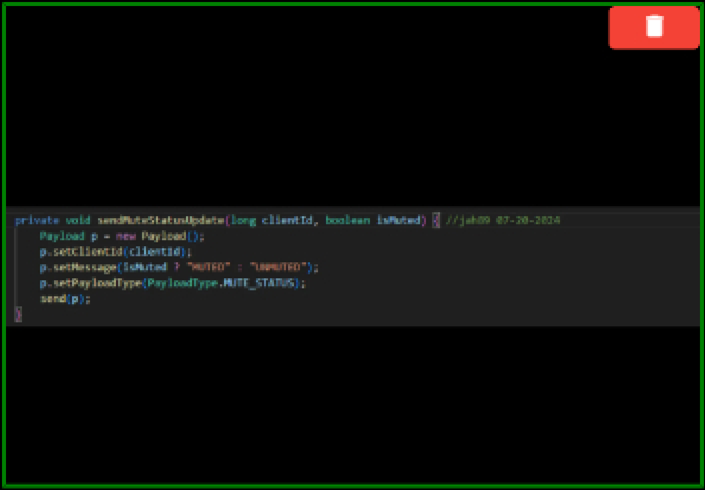
Describe/highlight what's being shown

Showing muted users appear red in ui.

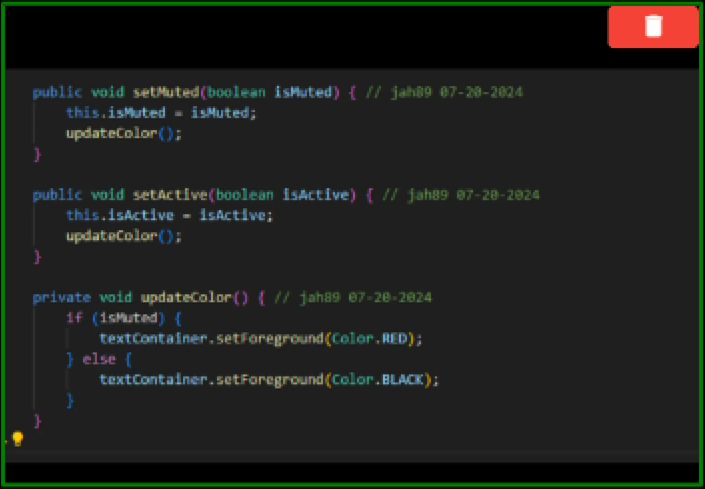
#2) Show the code flow (client receiving -> UI) for Muted users appear grayed out (or similar indication of your choosing)



```
// jah89 07-27-2024
private void handleMuteStatus(Payload payload) {
    long clientId = payload.getClientId();
    boolean isMuted = "MUTED".equals(payload.getMessage());
    ClientUI.getInstance().updateUserStatus(clientId, isMuted, isActive:false);
}
```



```
private void sendMuteStatusUpdate(long clientId, boolean isMuted) { //jah89 07-20-2024
    Payload p = new Payload();
    p.setClientId(clientId);
    p.setMessage(isMuted ? "MUTED" : "UNMUTED");
    p.setPayloadType(PayloadType.MUTE_STATUS);
    send(p);
}
```



```
public void setMuted(boolean isMuted) { // jah89 07-20-2024
    this.isMuted = isMuted;
    updateColor();
}

public void setActive(boolean isActive) { // jah89 07-20-2024
    this.isActive = isActive;
    updateColor();
}

private void updateColor() { // jah89 07-20-2024
    if (isMuted) {
        textContainer.setForeground(Color.RED);
    } else {
        textContainer.setForeground(Color.BLACK);
    }
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing code that muted users appear red in UI

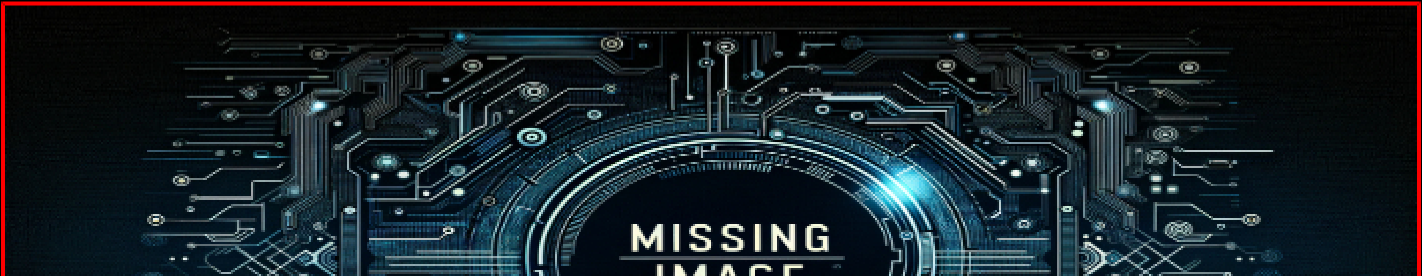
Explanation (required) ✓

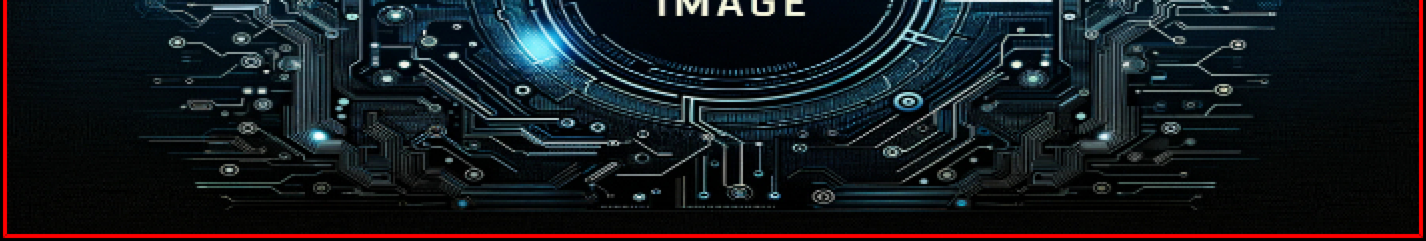
Explain in concise steps how this logically works

PREVIEW RESPONSE

In this code there is two attributes including isMuted and isActive which is used to track the mute and active status of a client or user. The setMuted method takes isMuted as a parameter and updates the isMuted attribute and calls the updateColor method to change the appearence of the user in the UI panel to red when muted. If the user is muted (isMuted is true), the text color is set to red. If the user is not muted, the text color is set to black.

#3) Show the UI for Last person to send a message gets highlighted (or similar indication of your choosing)





Caption (required)

Describe/highlight what's being shown

Missing caption

#4) Show the code flow (client receiving -> UI) for Last person to send a message gets highlighted (or similar indication of your choosing)

Caption (required)

Describe/highlight what's being shown

Missing caption

Explanation (required)

Explain in concise steps how this logically works

PREVIEW RESPONSE

Missing text

● 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/jah89/jah89-IT114-450/pull/17

URL
https://github.com/jah89/jah89-IT114-450/pull/

+ ADD ANOTHER URL

COLLAPSE

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

The only issue I really had during this assigment was time. For this final milestone I was on vacation for the week and wasn't able to actively work on the project until i got back which led me to crunch and rush most of it. Which led to me not be able to finish the last piece of the project of highlighting users.

COLLAPSE

Task #3 - Points: 1

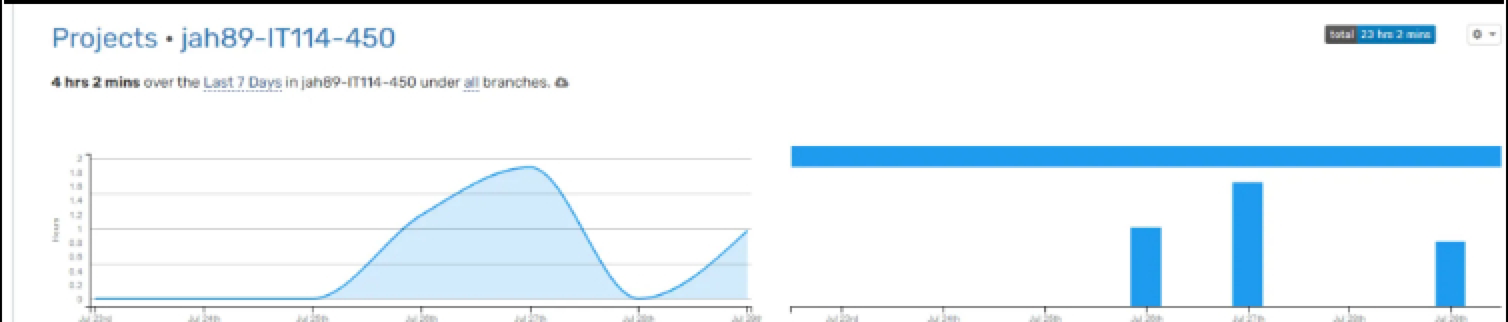
Text: WakaTime Screenshot

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


Projects • jah89-IT114-450

total 23 hrs 2 mins

4 hrs 2 mins over the Last 7 Days in jah89-IT114-450 under all branches.

Showing time spent past week

End of Assignment