

# Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-4-chatroom-2024-m24/grade/mcp62>

Course: IT114-003-F2024

Assignment: [IT114] Milestone 4 Chatroom 2024 M24

Student: Michael P. (mcp62)

## Submissions:

Submission Selection

1 Submission [submitted] 12/10/2024 6:12:25 PM

## Instructions

^ COLLAPSE ^

- Implement the Milestone 4 features from the project's proposal document:  
<https://docs.google.com/document/d/1ONmvEveI97GTFPGfVwwQC96xSsobbSbk56145XizQG4/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

Group



Group: Features

Tasks: 4

Points: 9

^ COLLAPSE ^

## Task

100%

Group: Features


Task #1: Client can export chat history of their current session (client-side)

Weight: ~0%

Points: ~0.01

^ COLLAPSE ^

### 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.

Columns: 1

### Sub-Task

100%

Group: Features

Task #1: Client can export chat history of their current session (client-side)

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

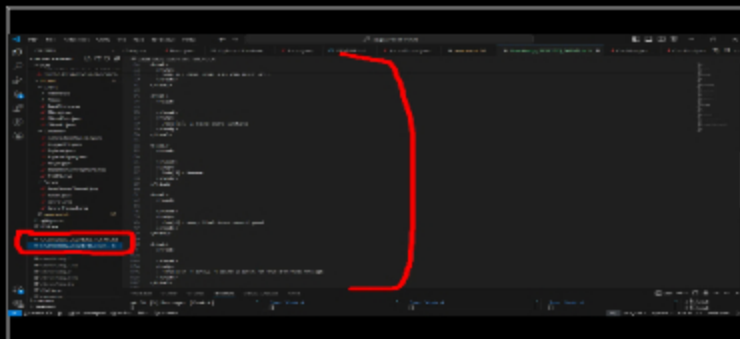
## Task Screenshots

Gallery Style: 2 Columns

4

2

1



Chat History files and context

### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

### Sub-Task

100%

Group: Features

Task #1: Client can export chat history of their current session (client-side)

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

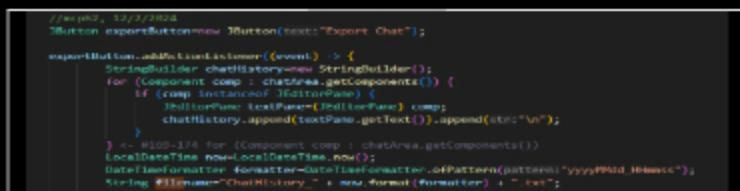
## Task Screenshots

Gallery Style: 2 Columns

4

2

1



```
FileWriter writer;
try {
    writer = new FileWriter(filename);
    writer.write(chatHistory.toString());
    writer.close();
} catch (IOException e) {
    e.printStackTrace();
}
} // #107-108 exportButton.addActionListener
import javax.swing.*;
public ChatPanel extends JPanel {
    // #104-105 public ChatPanel(JCardControls controls) {
```

Chat export w/ string builder

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

⇒ Task Response Prompt

*Explain in concise steps how this logically works*

Response:

Action Listener for Export Button: sets up an action listener for a button named exportButton. When this button is clicked, it triggers the following actions.

Collecting Chat History: creates a StringBuilder named chatHistory to store the text from the chat area. Does this by iterating through all components in chatArea.

Extracting Text from JEditorPane: For each component, if it is an instance of JEditorPane, it retrieves the text from that pane and appends it to chatHistory, followed by a newline character.

Generating Filename: It gets the current date and time using LocalDateTime.now() and formats it into a string with the pattern yyyyMMdd\_HH:mm:ss using DateTimeFormatter. Formatted string is used to create a filename.

Writing to File: It attempts to write the collected chat history to a file with the generated filename using a FileWriter. If successful, it closes the writer. If an IOException occurs, it prints the stack trace.

Adding Export Button to Input: lastly adds the exportButton to the input component.

Sub-Task

Group: Features



- Task #1: Client can export chat history of their current session (client-side)
- Sub Task #3: Show the UI interaction that will trigger an export

🖼 Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
//mcp62, 12/2/2024
JButton exportButton=new JButton(text:"Export Chat");
exportButton.addActionListener((event) -> {
```

creating button code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

T I D R

## Task Response Prompt

Explain where you put it any why

Response:

the button's size and position are managed by the layout manager of the ChatPanel which is why all i needed to do was make a new JButton and call it "export chat"

### End of Task 1

#### Task



Group: Features

Task #2: Client's Mute List will persist across sessions (server-side)

Weight: ~0%

Points: ~0.01

^ COLLAPSE ^

#### i 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.



Columns: 1

#### Sub-Task



Group: Features

Task #2: Client's Mute List will persist across sessions (server-side)

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

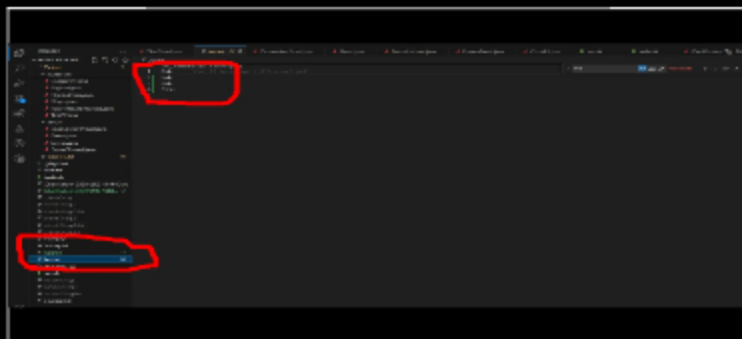
## Task Screenshots

Gallery Style: 2 Columns

4

2

1



Mute list files, named after user, list muted users

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown

#### Sub-Task



Group: Features

Task #2: Client's Mute List will persist across sessions (server-side)

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

## Task Screenshots

Gallery Style: 2 Columns

4

2

1

```
private void loadMuteList() {
    try {
        File file = new File(clientName + ".txt");

        if (file.exists()) {
            // You, last week + Milestone 4 Progress
            BufferedReader reader = new BufferedReader(new FileReader(file));
            mute.clear();
            String line;

            while ((line = reader.readLine()) != null) {
                mute.add(line);
            }

            reader.close();
        } else if (file.exists()) {
        } catch (IOException e) {
            System.out.println("Error loading mute list: " + e.getMessage());
        }
    }
} // 8261-279 private void loadMuteList()
```

load mute list

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown

### Task Response Prompt

Explain in concise steps how this logically works

Response:

File object is created using the client's name concatenated with the .txt extension. This represents the file that contains the mute list for that specific client.

Method checks if the file exists on the filesystem. If the file does not exist, the method will skip the reading process.

BufferedReader is created to read the contents of the file. This allows for efficient reading of text from the file.

Before loading the new mute list, the existing mute list is cleared to ensure that it does not contain any stale or outdated entries.

while loop used to read each line from the file until the end is reached (readLine() returning null). Each line represents a muted user, which is then added to the mute list.

After all lines have been read, the BufferedReader is closed.

#### Sub-Task

Group: Features

100%

Task #2: Client's Mute List will persist across sessions (server-side)

Sub Task #3: Show the code related to saving the mutelist whenever the list changes for a client

## Task Screenshots

Gallery Style: 2 Columns

4

2

1

```
private void saveMuteList() {
    try {
        // You, last week + Milestone 4 Progress
        BufferedWriter writer = new BufferedWriter(new FileWriter(clientName + ".txt"));

        for (String mutedUser: mute) {
            writer.write(mutedUser);
            writer.newLine();
        }

        writer.close();
    }
}
```

```
    } catch (IOException e) {
        System.out.println("Error saving mute list: " + e.getMessage());
    }
}

// end send methods
} // #20-207 public class ServerThread extends BaseServerThread
```

Save mute list method

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

## ≡ Task Response Prompt

*Explain in concise steps how this logically works*

Response:

BufferedWriter is created to write to a file named after the client's name (e.g., clientName.txt). The FileWriter is used to open the file for writing. If the file does not exist, it will be created.

For each loop to iterate over each entry in the mute list. Each entry represents a user that has been muted.

For each muted user, their name is written to the file. After writing the name, writer.newLine() is called to ensure that each entry is on a new line in the file.

After all entries have been written, the BufferedWriter is closed.

### End of Task 2

#### Task



Group: Features  
Task #3: Clients will receive a message when they get muted/unmuted by another user  
Weight: ~0%  
Points: ~0.00

^ COLLAPSE ^

#### i 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

Columns: 1

#### Sub-Task



Group: Features  
Task #3: Clients will receive a message when they get muted/unmuted by another user  
Sub Task #1: Show the code that generates the well formatted message only when the mute state changes (see notes in the details above)

## 🖼 Task Screenshots

Gallery Style: 2 Columns

4 2 1





```

private void mute(ServerThread client, String message)
{
    // Iterate over all clients currently in the room
    for (ServerThread clientInRoom : clientsInRoom)
    {
        // Check if the client's name matches the message parameter
        if (clientInRoom.getName().equals(message))
        {
            // If a match is found, the mute method of the client is called to mute the specified client.
            clientInRoom.mute();
        }
    }
    // Then the line makes a message to notify all clients in the room that the specified client has been muted. The
    // message is formatted as shown in the code.
    String msg = "User " + client.getName() + " has been muted. Muted by: " + clientInRoom.getName();
    broadcast(msg);
}

```

Message shown when the mute changes state

**Caption(s) (required)** ✓

Caption Hint: *Describe/highlight what's being shown*

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

Method iterates over all clients currently in the room, which are stored in a collection called `clientsInRoom`. Each client is represented by an instance of `ServerThread`

For each client in the room, the method checks if the client's name matches the message parameter

If a match is found, the mute method of the client is called to mute the specified client.

Then the line makes a message to notify all clients in the room that the specified client has been muted. The message is formatted as shown in the code.

**Sub-Task**

Group: Features



Task #3: Clients will receive a message when they get muted/unmuted by another user

Sub Task #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)

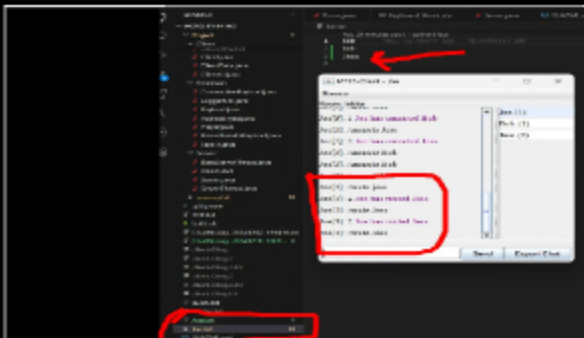
## Task Screenshots

Gallery Style: 2 Columns

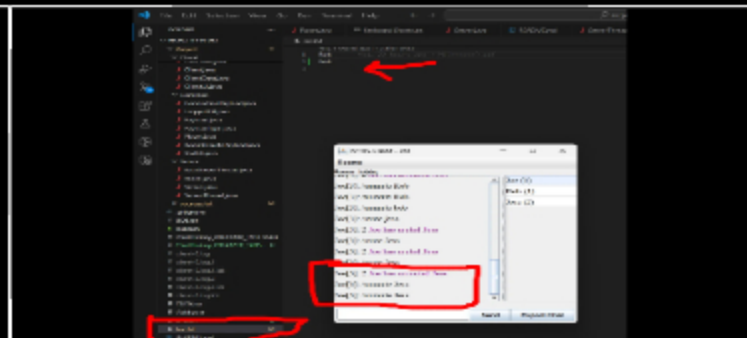
4

2

1



Jess only pops up one time after being muted twice



Jess removed from list and does not pop up again within list

**Caption(s) (required)** ✓

Caption Hint: *Describe/highlight what's being shown*



Group: Features

Task #4: The user list on the Client-side should update per the status of each user

Weight: ~0%

Points: ~0.01

^ COLLAPSE ^

### Details:

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



Columns: 1

#### Sub-Task

100%

Group: Features

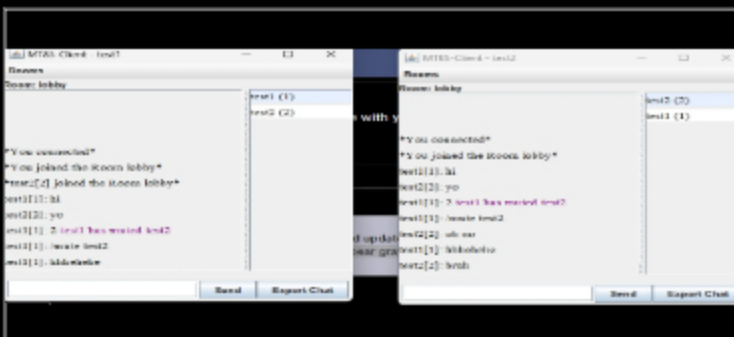
Task #4: The user list on the Client-side should update per the status of each user

Sub Task #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

## Task Screenshots

Gallery Style: 2 Columns

4 2 1



UI shown

### Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

#### Sub-Task

100%

Group: Features

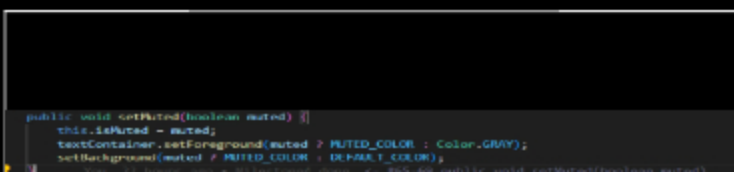
Task #4: The user list on the Client-side should update per the status of each user

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

## Task Screenshots

Gallery Style: 2 Columns

4 2 1





```

    public boolean isMuted() {
        return isMuted;
    }
}
// #17:78 public class UserListItem extends JPanel

```

Ui setForeGround

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown

## ⇒ Task Response Prompt

Explain in concise steps how this logically works

Response:

I was unable to get the UI to properly change depending on is muted. This is the code I wrote in attempt to target UserListItem, but it doesnt seem to target the usernames properly.

Sub-Task

100%

Group: Features

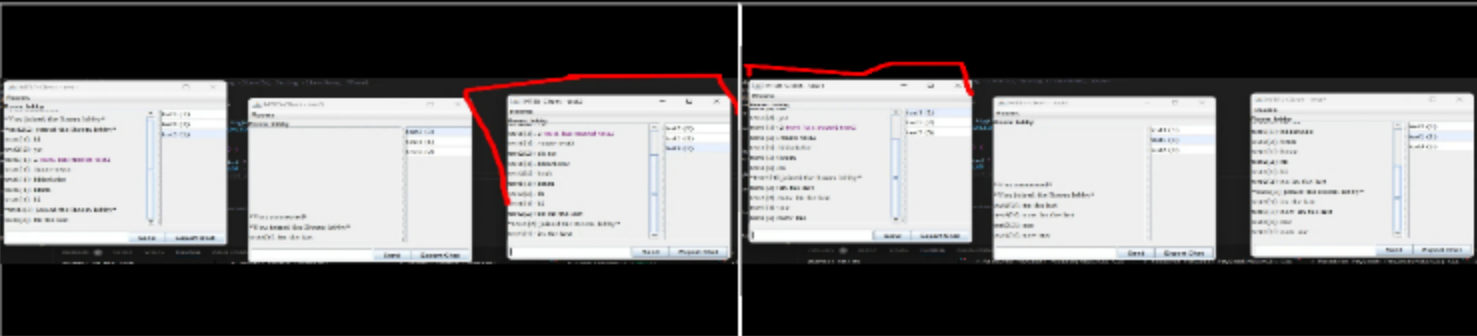
Task #4: The user list on the Client-side should update per the status of each user

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

## 🖼️ Task Screenshots

Gallery Style: 2 Columns

4 2 1



test2 is last

now test 1

Caption(s) (required) ✓

Caption Hint: Describe/highlight what's being shown

Sub-Task

100%

Group: Features

Task #4: The user list on the Client-side should update per the status of each user

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

## 🖼️ Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
userItemsMap.values().forEach((item) -> item.sethighlighted(highlighted:false));
// Highlight last sender
UserListItem item = userItemsMap.get(clientId);
if (item != null) {
    item.sethighlighted(highlighted:true);
}
});
```

last sender method

**Caption(s) (required) ✓**

Caption Hint: *Describe/highlight what's being shown*

## ≡ Task Response Prompt

*Explain in concise steps how this logically works*

Response:

When any message is received, onMessageReceive is triggered setLastSender is called with the sender's ID All users in the list have their highlight removed Only the sender of the last message gets highlighted The highlight is a light blue background (HIGHLIGHT\_COLOR)

End of Task 4

End of Group: Features  
Task Status: 4/4

Group

100%

Group: Misc

Tasks: 3

Points: 1

⌵ COLLAPSE ⌵

Task

100%

Group: Misc

Task #1: Add the pull request link for the branch

Weight: ~33%

Points: ~0.33

⌵ COLLAPSE ⌵

ⓘ

Details:

Note: the link should end with /pull/#

⌵

## ↪ Task URLs

URL #1

<https://github.com/Onervv/mcp62-IT114-003/pull/19>

URL

<https://github.com/Onervv/mcp62-IT114-003/pul>

## End of Task 1

### Task



Group: Misc

Task #2: Talk about any issues or learnings during this assignment

Weight: ~33%

Points: ~0.33

^ COLLAPSE ^

## ≡ Task Response Prompt

Response:

Only thing I wasn't able to get done was the highlight if isMuted, which is weird because I was trying to use the logic I got working from highlight last User. I just need to turn in at this point and move forward, I tried implementing it in directly too but that didn't work. Other than that everything else was okay.

## End of Task 2

### Task



Group: Misc

Task #3: WakaTime Screenshot

Weight: ~33%

Points: ~0.33

^ COLLAPSE ^

### i Details:

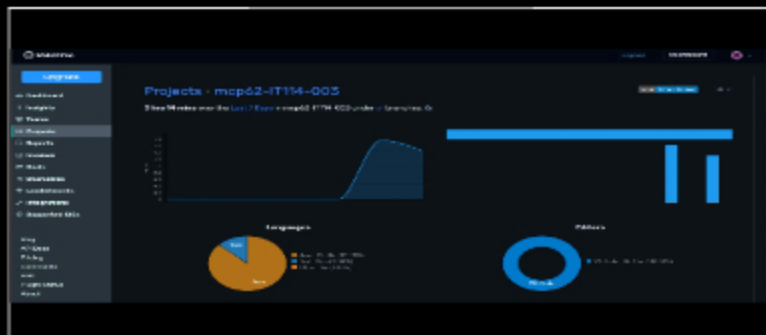
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved



## 🖼 Task Screenshots

Gallery Style: 2 Columns

4 2 1



WAKA

## End of Task 3

End of Group: Misc  
Task Status: 3/3

End of Assignment