

# Submission Worksheet

**CLICK TO GRADE**

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-3-chatroom-2024-m24/grade/mcp62>

Course: IT114-003-F2024

Assignment: [IT114] Milestone 3 Chatroom 2024 M24

Student: Michael P. (mcp62)

## Submissions:

Submission Selection

1 Submission [submitted] 12/9/2024 7:14:33 PM

## Instructions

[^ COLLAPSE ^](#)

Implement the Milestone 3 features from the project's proposal document:

<https://docs.google.com/document/d/10NmveI97GTFPGfVwwQC96xSobbSbk56145XizQG4/view>

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

Group



Group: Basic UI

Tasks: 1

Points: 2

[^ COLLAPSE ^](#)

Task



Group: Basic UI

Task #1: UI Panels

Weight: ~100%

Points: ~2.00

**i Details:**

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.



Columns: 1

**Sub-Task**

Group: Basic UI

Task #1: UI Panels

Sub Task #1: Show the ConnectionPanel by running the app (should have host/port)

**Task Screenshots**

Gallery Style: 2 Columns

4 2 1



Connection Panel

**Caption(s) (required) ✓**Caption Hint: *Describe/highlight what's being shown***Sub-Task**

Group: Basic UI

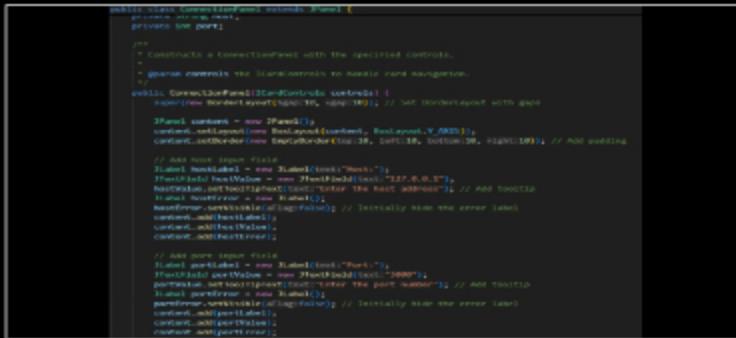
Task #1: UI Panels

Sub Task #2: Show the code related to the ConnectionPanel

**Task Screenshots**

Gallery Style: 2 Columns

4 2 1



ConnectionPanel.java

**Caption(s) (required) ✓**

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

## Task Response Prompt

*Briefly explain how it works and how it's used*

Response:

- Uses BorderLayout for the main panel
- Contains text fields for host (default: "127.0.0.1") and port (default: "3000")
- Has a "Next" button to proceed
- Includes error labels for validation feedback
- User enters host and port values
- When "Next" is clicked, it validates the port number
- If valid, stores the values and moves to the next panel
- If invalid, displays an error message

**Sub-Task**

Group: Basic UI

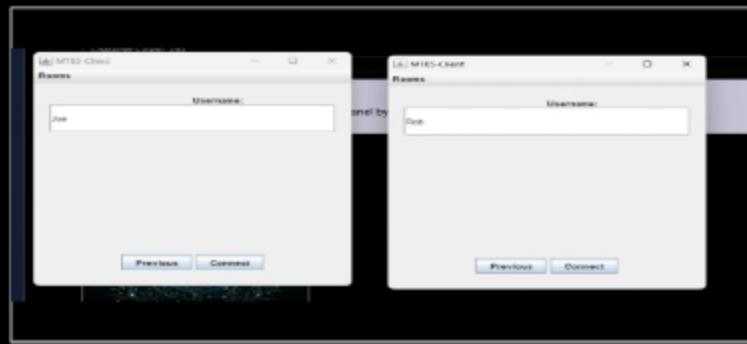
100%

Task #1: UI Panels

Sub Task #3: show the UserDetailsPanel by running the app (should have username)

## Task Screenshots

Gallery Style: 2 Columns



UserDetailsPanel

**Caption(s) (required) ✓**

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

**Sub-Task**

Group: Basic UI

100%

Task #1: UI Panels

Sub Task #4: Show the code related to the UserDetailsPanel

## Task Screenshots

Gallery Style: 2 Columns

4 2 1

```

public class UserDetailsPanel extends JPanel {
    private String username;
}

/*
 * Constructor to create the UserDetailsPanel UI.
 */
@Focusable(true)
implements ICardControls<UserDetails> {
    JButton connectButton = new JButton("Connect");
    JTextField usernameField = new JTextField("username");
    JTextField passwordField = new JTextField("password");
    JLabel errorMessage = new JLabel("");
    JButton previousButton = new JButton("Previous");
    JButton nextButton = new JButton("Next");
    JButton cancelButton = new JButton("Cancel");
}

UserDetailsPanel() {
    JPanel contentPanel = new JPanel();
    contentPanel.setLayout(new BorderLayout());
    contentPanel.add(usernameField, BorderLayout.NORTH);
    contentPanel.add(passwordField, BorderLayout.CENTER);
    contentPanel.add(errorMessage, BorderLayout.SOUTH);

    JPanel buttonPanel = new JPanel();
    buttonPanel.add(connectButton);
    buttonPanel.add(previousButton);
    buttonPanel.add(nextButton);
    buttonPanel.add(cancelButton);

    contentPanel.add(buttonPanel, BorderLayout.EAST);
}

void setUsername(String username) {
    this.username = username;
}

String getUsername() {
    return username;
}

void setErrorMessage(String errorMessage) {
    this.errorMessage.setText(errorMessage);
}

String getErrorMessage() {
    return errorMessage.getText();
}

void setConnectEnabled(boolean enabled) {
    connectButton.setEnabled(enabled);
}

boolean isConnectEnabled() {
    return connectButton.isEnabled();
}

void setPreviousEnabled(boolean enabled) {
    previousButton.setEnabled(enabled);
}

boolean isPreviousEnabled() {
    return previousButton.isEnabled();
}

void setNextEnabled(boolean enabled) {
    nextButton.setEnabled(enabled);
}

boolean isNextEnabled() {
    return nextButton.isEnabled();
}

void setCancelButtonEnabled(boolean enabled) {
    cancelButton.setEnabled(enabled);
}

boolean isCancelButtonEnabled() {
    return cancelButton.isEnabled();
}

void setFocusable(boolean focusable) {
    super.setFocusable(focusable);
}

@Override
public void keyTyped(KeyEvent e) {
    if (e.getKeyChar() == ' ') {
        e.consume();
    }
}

@Override
protected void processWindowEvent(WindowEvent event) {
    super.processWindowEvent(event);
}
}

```

### UserDetailsPanel

#### Caption(s) (required) ✓

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

## ≡, Task Response Prompt

*Briefly explain how it works and how it's used*

Response:

Users enter their desired username in a text field. The panel prevents spaces in usernames through a KeyListener. Error messages appear if the user tries to connect with an empty username. Navigation Control will allow the "Previous" button to return to the connection details screen. The "Connect" button initiates the connection process when clicked with a valid username. Navigation is managed through the ICardControls interface

Validation and other features added aswell as the panel set up and dimesions.

#### Sub-Task

Group: Basic UI



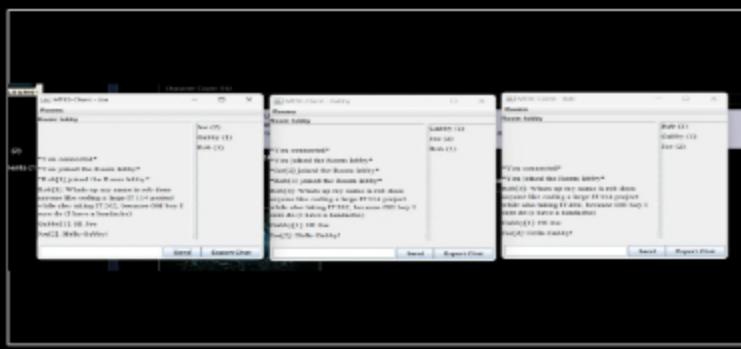
Task #1: UI Panels

Sub Task #5: Show the ChatPanel (there should be at least 3 users present and some example messages)

## ❑ Task Screenshots

Gallery Style: 2 Columns

4 2 1



### 3 Clients within chatPanel

#### Caption(s) (required) ✓

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

#### Sub-Task

Group: Basic UI

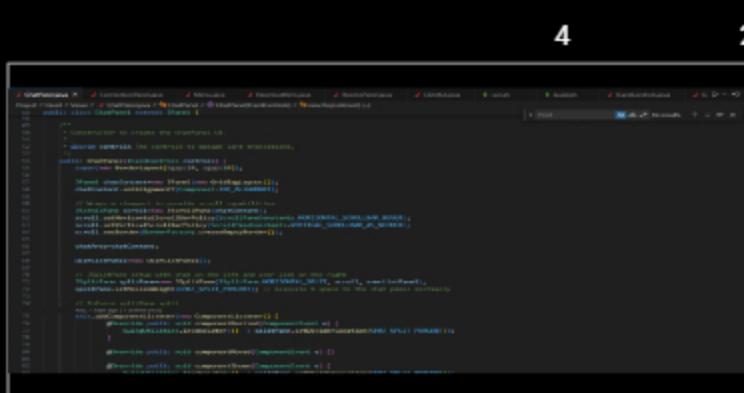
Task #1: UI Panels

100%

Sub Task #6: Show the code related to the ChatPanel

## Task Screenshots

Gallery Style: 2 Columns



A screenshot of a Java IDE showing the code for ChatPanel.java. The code implements a split-screen interface for a chat application. It uses BorderLayout and GridBagLayout. Key features include a scrollable chat area, a user list panel on the right, and an input area at the bottom. The code handles message sending via a text field and "Send" button, and supports HTML formatting in messages. It also includes a user list management section and a timestamped export function.

```
1 package com.chatapp;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.*;
6 import java.util.*;
7 import java.io.*;
8
9 public class ChatPanel extends JPanel {
10     // Implementation details...
11 }
```

ChatPanel.java code

**Caption(s) (required)** ✓**Caption Hint:** *Describe/highlight what's being shown*

## Task Response Prompt

**Briefly explain how it works and how it's used (note the important parts of the ChatPanel)****Response:**

ChatPanel.java serves as the main interface for the chat application, implementing a split-screen design where 70% of the window displays the chat messages and 30% shows the user list. The panel uses BorderLayout and contains three key components, a scrollable chat area for messages, a user list panel on the right, and an input area at the bottom. Users can send messages through a text field using either the "Send" button or the Enter key, and messages are displayed in the chat area with HTML formatting support. The panel includes features like automatic scrolling to new messages, user list management (adding/removing users), and a chat export function that saves the conversation history to a timestamped text file. All UI updates are handled through SwingUtilities.invokeLater to maintain thread safety, and the panel uses GridBagLayout to properly arrange and display messages while maintaining proper spacing and formatting.

End of Task 1

End of Group: Basic UI

Task Status: 1/1

**Group**

Group: Build-up

Tasks: 2

Points: 3

100%

▲ COLLAPSE ▲

**Task**

Group: Build-up

Total: 100% | Last updated: 5/17/2024 11:45:00 AM | Last modified: 5/17/2024 11:45:00 AM

100%

Task #1: Results of /flip and /roll appear in a different format than regular chat text  
Weight: ~50%  
Points: ~1.50

▲ COLLAPSE ▾

### ● Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.



Columns: 1

Sub-Task

100%

Group: Build-up

Task #1: Results of /flip and /roll appear in a different format than regular chat text

Sub Task #1: Show examples of it printing on screen

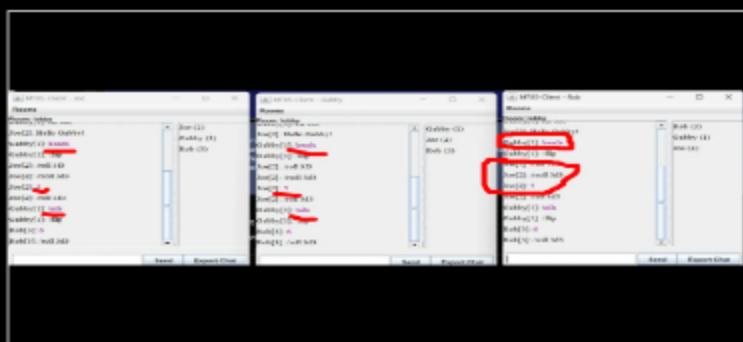
## ▣ Task Screenshots

Gallery Style: 2 Columns

4

2

1



you can see results are a different color

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Build-up

Task #1: Results of /flip and /roll appear in a different format than regular chat text

Sub Task #2: Show the code on the Room side that changes this format

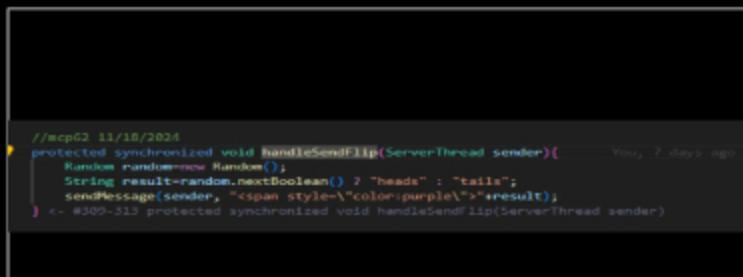
## ▣ Task Screenshots

Gallery Style: 2 Columns

4

2

1



server thread handles flip, while the output is adjusted

### Caption(s) (required) ✓

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

## Task Response Prompt

*Explain what you did and how it works*

Response:

Targeted by altering the style of the message using html. The results variable what is being modified.

End of Task 1

### Task

Group: Build-up



Task #2: Text Formatting appears correctly on the UI

Weight: ~50%

Points: ~1.50

[▲ COLLAPSE ▲](#)

### Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.



Columns: 1

### Sub-Task

Group: Build-up



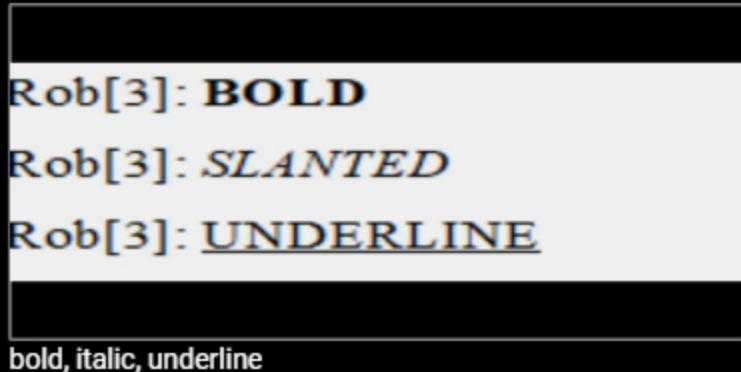
Task #2: Text Formatting appears correctly on the UI

Sub Task #1: Show examples of bold, italic, underline, each color implemented and a combination of bold, italic, underline, and one color in the same message

## Task Screenshots

Gallery Style: 2 Columns

4      2      1



bold, italic, underline

Rob[3]: Green  
Rob[3]: Red  
Rob[3]: Blue

RGB

Rob[3]: THIS IS MY MIXED MESSAGE

## Mixed Message

**Caption(s) (required)** ✓

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

Sub-Task

Group: Build-up

100%

Task #2: Text Formatting appears correctly on the UI

Sub Task #2: Show the code changes necessary to get this to work

## Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
/*msg02 11/10/2024
while (true){
String originalMessage = message;
if (message.contains("<h1>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:blue\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h2>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:red\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h3>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:green\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h4>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:orange\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h5>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:purple\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h6>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:teal\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h7>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:yellow\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h8>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:pink\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.contains("<h9>")){
    message=message.replaceFirst(regex:"\\n\\b", replacement:<span style='color:cyan\\n\\b>");}
    message=message.replaceFirst(regex:"\\n\\b", replacement:</span>");}
else if (message.equals(originalMessage)){
    break;
}
}
--#msg-210 while (true)
```

Code for applying styles

**Caption(s) (required)** ✓

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

## Task Response Prompt

*Briefly explain what was necessary and how it works*

Response:

Checks the message contents if the message contains certain string, if nothing is found then it breaks out of the loop, this is done for every message (while true). essentially we are assuming every message will contain some regex that indicates the message should be modified by the regex identifier, but if it doesn't the loop exits until the condition is met again.

End of Task 2

End of Group: Build-up

Task Status: 2/2

Group

Group: New Features

100%

Tasks: 2  
Points: 4

▲ COLLAPSE ▲

### Task

100%

Group: New Features

Task #1: Private messages via @username

Weight: ~50%

Points: ~2.00

▲ COLLAPSE ▲

#### ⓘ Details:

All code screenshots must include ucid/date.

App screenshots must have the UCID in the title bar like the lesson gave.



Columns: 1

#### Sub-Task

100%

Group: New Features

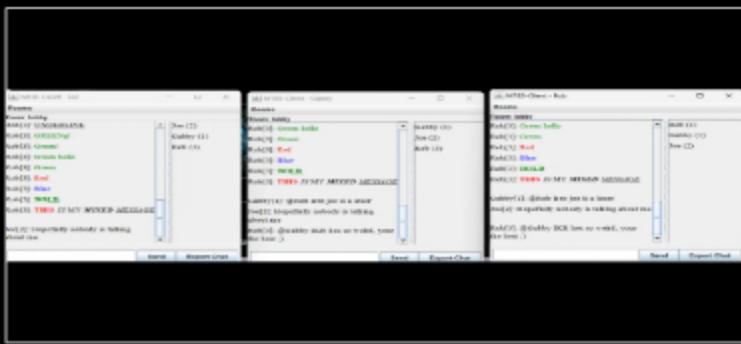
Task #1: Private messages via @username

Sub Task #1: Show a few examples across different clients (there should be at least 3 clients in the Room)

## 🖼 Task Screenshots

Gallery Style: 2 Columns

4 2 1



Example across 3 clients

#### Caption(s) (required) ✓

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

#### Sub-Task

100%

Group: New Features

Task #1: Private messages via @username

Sub Task #2: Show the client-side code that processes the text per the requirement

## 🖼 Task Screenshots

Gallery Style: 2 Columns

```
protected void sendPrivateMessage(ConcurrentHashMap<String, String> message) {
    String userName = message.split(" ")[0].substring(beginIndex-1);
    if (!isRunning) {
        return;
    }
}
```

method for sendPrivateMessage

**Caption(s) (required)** ✓

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

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

message.split(" ") creates array: ["@Bob", "hello", "there"] [0] gets first element: "@Bob" substring(1) removes the '@': "Bob"

Checks if room is running

Applies any formatting

Sends the original message to

- The recipient matching the extracted username
- The sender of the message
- Only sends if the recipient hasn't muted the sender

Sub-Task

Group: New Features

100%

Task #1: Private messages via @username

Sub Task #3: Show the ServerThread code receiving the payload and passing it to Room

## Task Screenshots

Gallery Style: 2 Columns

```
private void sendPrivateMessage(ConcurrentHashMap<String, String> message) {
    if (!isRunning) {
        return;
    }
    if (message.startsWith("@")) {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.PRIVATE);
        p.setRoomName(message.substring(1));
        send(p);
    } else if (message.startsWith("room")) {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.CHAT);
        p.setRoomName(message.substring(5));
        send(p);
    } else if (message.startsWith("leave")) {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.CHAT);
        p.setRoomName(message.substring(5));
        send(p);
    } else if (message.startsWith("muted")) {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.MUTED);
        p.setRoomName(message.substring(6));
        send(p);
    }
}
```

handling the payload

**Caption(s) (required) ✓**Caption Hint: *Describe/highlight what's being shown*

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

message payload is checked by "@" if found, sets payload type to type PRIVATE to be hanled by room.java

**Sub-Task**

Group: New Features

100%

Task #1: Private messages via @username

Sub Task #4: Show the Room code that verifies the id and sends the message to both the sender and receiver

## Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
final String modifiedMessage = message;
info(String.format("Modified message to %d recipients: %s", clientsInRoom.size(), message));
clientsInRoom.forEach(client -> {
    if (!client.equals(sender.getClientName())) {
        boolean failedToSend = client.sendMessage(modifiedMessage);
        if (failedToSend) {
            info(String.format("Removing disconnected client[%s] from list", client.getClientId()));
            disconnect(client);
        }
    }
    return failedToSend;
});
<- #220-225 IF (!client.equals(sender.getClientName()))
return false;
});
<- #227-237 clientsInRoom.values().removeIf
<- #165-180 protected synchronized void sendMessage(ServerThread sender, ...)
```

code for checking client ID and modified messages

**Caption(s) (required) ✓**Caption Hint: *Describe/highlight what's being shown*

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

Stores the content within originalmessage,checks for regex,in other method if @ is found then sent to the sender and reciver

End of Task 1

**Task**

100%

Group: New Features

Task #2: Mute and Unmute

Weight: ~50%

Points: ~2.00

▲ COLLAPSE ▲

**i Details:**

All code screenshots must include ucid/date.  
App screenshots must have the UCID in the title bar like the lesson gave.

- Client-side will implement a /mute and /unmute command (i.e., /mute Bob or /unmute Bob)

Columns: 1

**Sub-Task**

Group: New Features



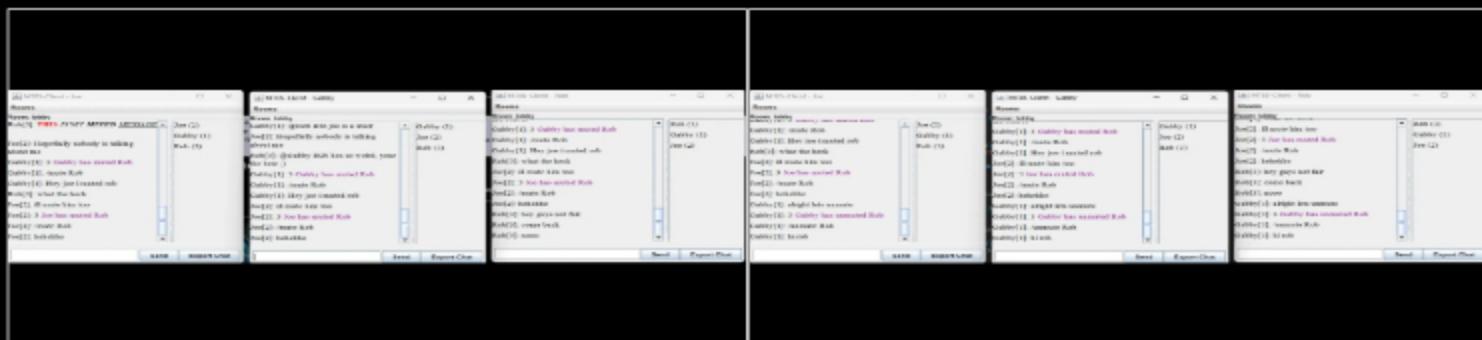
Task #2: Mute and Unmute

Sub Task #1: Show a few examples across different clients (there should be at least 3 clients in the Room)

## Task Screenshots

Gallery Style: 2 Columns

4      2      1



/mute Rob

/unmute

**Caption(s) (required) ✓**

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

**Sub-Task**

Group: New Features



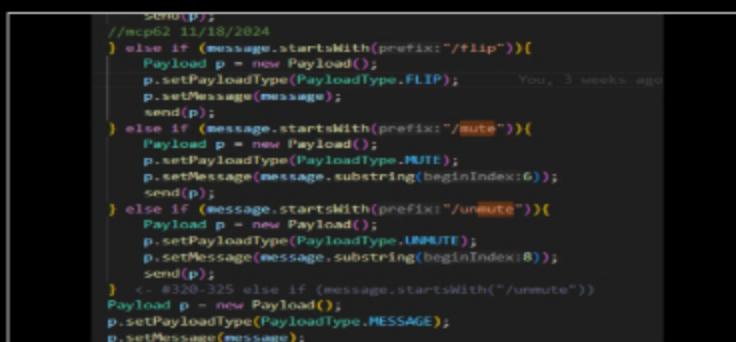
Task #2: Mute and Unmute

Sub Task #2: Show the client-side code that processes the text per the requirement

## Task Screenshots

Gallery Style: 2 Columns

4      2      1



/mute and /unmute client side

**Caption(s) (required) ✓**

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

## Task Response Prompt

*Explain in one/some steps how this is basically working*

*Explain in concise steps how this logically works*

**Response:**

checks message if the message contains item /mute or /unmute by a regex patter then a new payload is created the handles the mute within room.

**Sub-Task**

Group: New Features

Task #2: Mute and Unmute

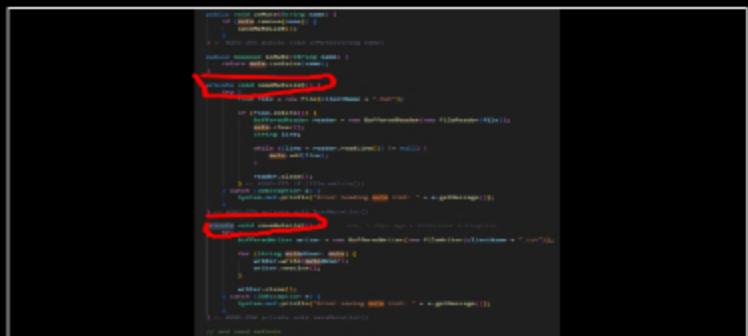
Sub Task #3: Show the ServerThread code receiving the payload and passing it to Room

100%

## Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
protected void handleMute(Command command) {
    String[] args = command.getArgs();
    if (args.length > 1) {
        String targetName = args[1];
        // Logic to handle mute
    }
}

protected void handleUnmute(Command command) {
    String[] args = command.getArgs();
    if (args.length > 1) {
        String targetName = args[1];
        // Logic to handle unmute
    }
}

protected void handleConnect(Command command) {
    String[] args = command.getArgs();
    if (args.length > 1) {
        String targetName = args[1];
        // Logic to handle connection
    }
}
```

payload handler

**Caption(s) (required)** ✓

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

## Task Response Prompt

*Explain in concise steps how this logically works*

**Response:**

Each client (user) has their own list of people they've muted, stored in the mute ArrayList. When a user mutes someone: The muted person's name is added to the list. The list is saved to a file named [username].txt. When a user unmutes someone: The person's name is removed from the list. The list is saved to the file again. When a user connects: The system looks for a file with their username (like "john.txt"). If it exists, it loads all the muted names from the file. These names are added to their mute list.

**Sub-Task**

Group: New Features

Task #2: Mute and Unmute

Sub Task #4: Show the Room code that verifies the id and add/removes the muted name to/from the ServerThread's list

100%

## Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
protected void handleMute(Command command) {
    String[] args = command.getArgs();
    if (args.length > 1) {
        String targetName = args[1];
        // Logic to handle mute
    }
}

protected void handleUnmute(Command command) {
    String[] args = command.getArgs();
    if (args.length > 1) {
        String targetName = args[1];
        // Logic to handle unmute
    }
}

protected void handleConnect(Command command) {
    String[] args = command.getArgs();
    if (args.length > 1) {
        String targetName = args[1];
        // Logic to handle connection
    }
}
```

```
private void mute(String name) {
    if (!mute.contains(name)) {
        mute.add(name);
        saveMuteList();
    }
}

private void unMute(String name) {
    if (mute.remove(name)) {
        saveMuteList();
    }
}

private boolean isMute(String name) {
    return mute.contains(name);
}
```

Mute and unmute room

**Caption(s) (required)** ✓

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

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

The code shows two main functions that handle muting and unmuting users in a chat room system. The mute function starts by searching through all the clients currently in the room. When it finds the person that matches the name the user wants to mute, it tells the client's system to add that person to their mute list. It then sends a message to the room in purple text to let everyone know that someone was muted. The unmute function works in a similar way, but does the opposite. It looks through the room for someone who is currently muted by the requesting user. When it finds that person, it removes them from the user's mute list. Just like with muting, it sends a purple message to everyone in the room announcing that someone was unmuted.

Sub-Task

Group: New Features

100%

Task #2: Mute and Unmute

Sub Task #5: Show the Room code that checks the mute list during send message, private message, and any other relevant location

## Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
public void mute(String name) {
    if (!mute.contains(name)) {
        mute.add(name);
        saveMuteList();
    }
}

public void unMute(String name) {
    if (mute.remove(name)) {
        saveMuteList();
    }
}

public boolean isMute(String name) {
    return mute.contains(name);
}
```

checks muted

**Caption(s) (required)** ✓

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

## Task Response Prompt

*Explain in concise steps how this logically works*

Response:

This code is part of the ServerThread class which handles individual client connections on the server side. The isMute(String name) checks whether a specific user (identified by their name) is in the mute list of the current client. It works by using the contains() method on the mute ArrayList, which stores all the names of users that the current

client has chosen to mute. The method returns true if the specified name exists in the mute list, and false otherwise. It allows the server to quickly determine whether messages from certain users should be filtered out before being sent to the client who has muted them. The method is called whenever a message is being processed to determine if it should be delivered to a particular client based on their mute preferences.

### Sub-Task

100%

Group: New Features

Task #2: Mute and Unmute

Sub Task #6: Show terminal supplemental evidence per the requirements (refer to the details of this task)

## Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
> Received Payload: Payload[MESSAGE] Client Id [1] Message: [/unmute Bob]
12/05/2024 18:00:40 [Project.Client.Client] [1] [2mDPhw](me)
> Received Payload: Payload[MESSAGE] Client Id [1] Message: (hi mom)
> Received Payload: Payload[MESSAGE] Client Id [1] Message: (I'm home)
> Received Payload: Payload[MESSAGE] Client Id [1] Message: (I've been having trouble with my work)
12/05/2024 18:00:45 [Project.Client.Client] [1] [2mDPhw](me)
> Received Payload: Payload[MESSAGE] Client Id [1] Message: [/mute Bob]
12/05/2024 18:00:46 [Project.Client.Client] [1] [2mDPhw](me)
> Received Payload: Payload[MESSAGE] Client Id [1] Message: (Bob is muted)
12/05/2024 18:00:47 [Project.Client.Client] [1] [2mDPhw](me)
> Received Payload: Payload[MESSAGE] Client Id [1] Message: (Lalala)
```

terminal evidence

Caption(s) (required) ✓

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

End of Task 2

End of Group: New Features

Task Status: 2/2

### Group

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/17>

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:

The main issue was unable to access the clients name through the rooms name so i asked the professor and was able to come up with a solution/work-around

End of Task 2

### Task



Group: Misc

Task #3: WakaTime Screenshot

Weight: ~33%

Points: ~0.33

[▲ COLLAPSE ▲](#)

### ⓘ Details:

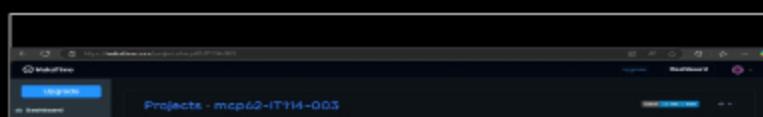
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved



## 🖼 Task Screenshots

Gallery Style: 2 Columns

4      2      1





Waka, no way this is accurate

End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment