Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-milestone-3-chatroom-2024-m24/grade/kr553

Course: IT114-005-F2024
Assigment: [IT114] Milestone 3 Chatroom 2024 M24
Student: Kush R. (kr553)

Submissions:

Submission Selection

1 Submission [submitted] 11/24/2024 1:10:34 AM

Instructions

COLLAPSE

Implement the Milestone 3 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 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

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: 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

ConnectionPanel on run

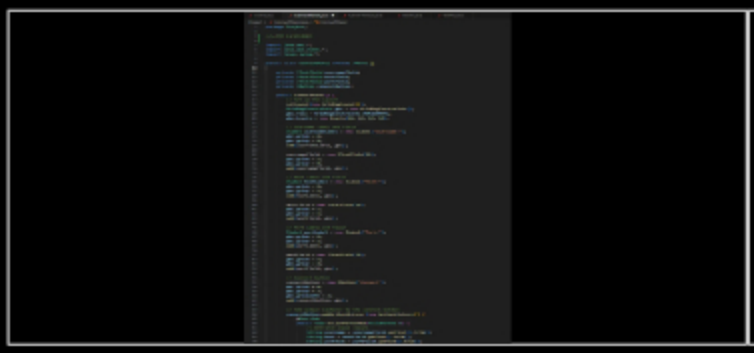
Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Sub-Task
100%
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



hald code for connectpanel.java the UI part

here is the other half code for connectpanel.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:

The ConnectionPanel is a user interface component that allows users to connect to the chat server. It consists of three main input fields: one for the server's host address (e.g., "localhost"), one for the port number (e.g., "3000"), and one for the user's desired username. Additionally, it includes a Connect button that initiates the connection process. When the Connect button is clicked, the panel retrieves the input values and validates them, ensuring the port number is numeric and the username is not empty. If the inputs are valid, it calls the connectToServer method from the Client class to establish a connection with the server using the provided host and port. Upon successful connection, it notifies the user and transitions the interface to the chat room panel. If the connection fails or inputs are invalid, it displays appropriate error messages to guide the user.

Sub-Task

100%

Group: Basic UI

Task #1: UI Panels

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

user details panel on right side of each client.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Basic UI

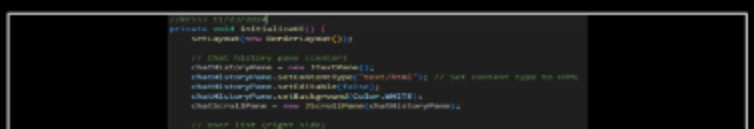
Task #1: UI Panels

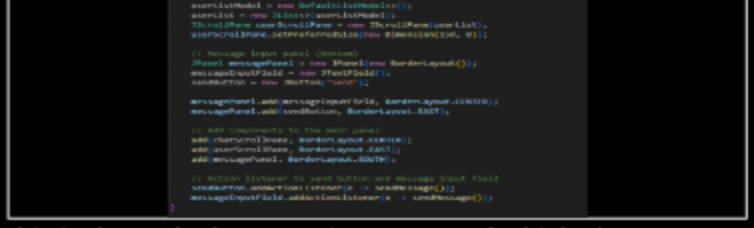
Sub Task #4: Show the code related to the UserDetailsPanel

Task Screenshots

Gallery Style: 2 Columns

4 2 1





this is the code that contains userpanel which shows users in the room

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:

The ChatRoomPanel manages the chat interface, displaying messages and an up-to-date user list. It uses a JTextPane for styled chat history and a JList backed by a DefaultListModel for usernames. Messages are sent via a text field and "Send" button, triggering client-side actions to forward them to the server. The `updateUserList` method ensures real-time updates when participants join, leave, or change names, keeping the interface synchronized. The server notifies the client of updates, allowing the user list and messages to reflect changes instantly. This design ensures smooth, real-time communication, forming a foundation for features like private messaging.

Sub-Task

100%

Group: Basic UI

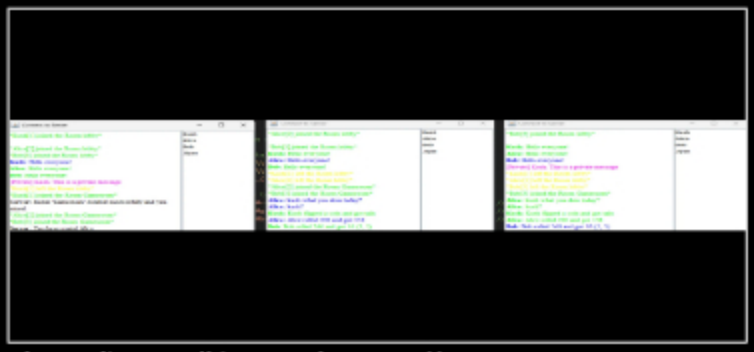
Task #1: UI Panels

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



Three clients talking on chat rom :))

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Basic UI

Task #1: UI Panels

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

code of chatpanel that displays messages in colorized format using TextFX.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 (note the important parts of the ChatPanel)

Response:

The **ChatPanel** is the core of the chat application, designed to keep everything simple and user-friendly. It focuses on two main tasks: showing messages and managing user interactions. Here's how it works:

- Chat History: The chat messages are displayed in a **JTextPane**, styled with HTML so everything looks clean and easy to read.
- Active Users: A **JList** shows who's online. It updates automatically whenever someone joins, leaves, or changes their name, thanks to a **DefaultListModel**.
- Sending Messages: There's a text box where users can type messages and a "Send" button to share them instantly.
- Real-Time Updates: The `updateUserList` method ensures the user list is always current, syncing directly with the server whenever things change.

The **ChatPanel** makes chatting seamless, whether it's keeping track of users or enabling instant communication. It's also a great base for adding cool features like private chats or status updates!

End of Task 1

End of Group: Basic UI
Task Status: 1/1

Group

100%

Group: Build-up
Tasks: 2
Points: 3

COLLAPSE

Task

100%

Group: Build-up
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

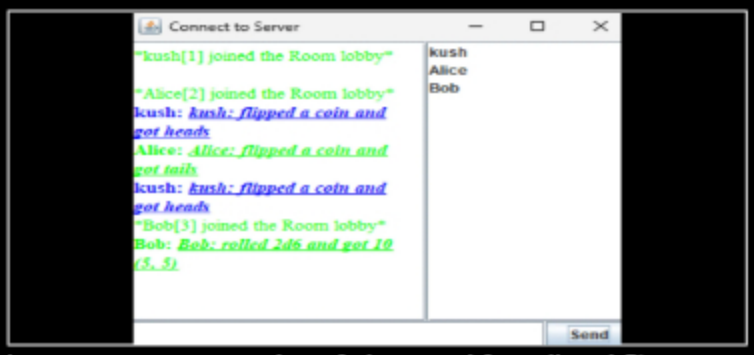
100%

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



here are some examples of chatpanel for roll and flip commands

Caption(s) (required) ✓

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

Sub-Task

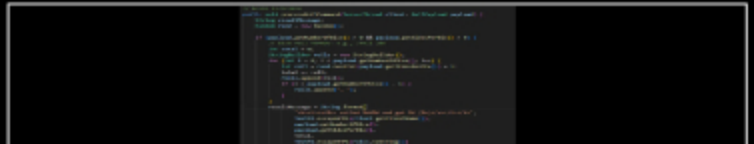
100%

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



here is the code

Caption(s) (required) ✓

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

Task Response Prompt

Explain what you did and how it works

Response:

I updated the `/flip` and `/roll` commands to display their outputs in bold, italic, and underlined styles using HTML tags (`<b><i><u>`). This visually differentiates them from regular chat messages. The formatting is applied before broadcasting, ensuring clients see the enhanced style directly in their chat panels.

End of Task 1

Task

100%

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

100%

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

here is proper formatting on chatpanel

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Build-up
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

too much work for this one specifically :) whooshh

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain what was necessary and how it works

Response:

I updated the `formatText` method in `TextFX.java` to recognize and apply combinations of bold, italic, underline, and color using special syntax like `**`, `_`, and `#color#`. It processes the text step-by-step, replacing these patterns with HTML tags to display styled text seamlessly in the chat UI.

End of Task 2

End of Group: Build-up
Task Status: 2/2

Group

100%

Group: New Features
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

Private messages

Caption(s) (required) ✓

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

Sub-Task

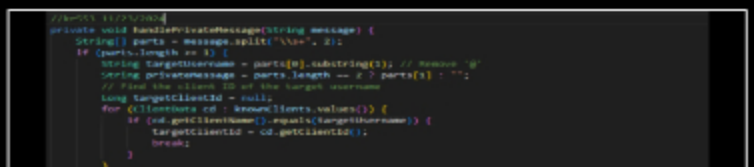
100%

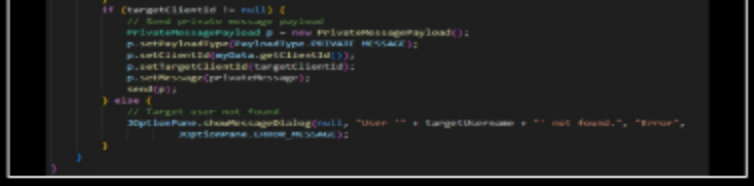
Group: New Features
Task #1: Private messages via @username
Sub Task #2: Show the client-side code that processes the text per the requirement

Task Screenshots

Gallery Style: 2 Columns

4 2 1





Handelling private messages

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The user types @username message. The program splits the input into the username and message. It searches for the username to get their clientId. A private message payload is created with the sender, receiver, and message. The payload is sent to the server. Errors show if username is invalid.

Sub-Task

100%

Group: New Features

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

4 2 1

Process payload containing all fuctionality with private messaging too

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Receive Message: The server thread gets a private message payload from the client. Check the Room: It verifies if the client is in a room. Pass to Room: The private message, sender ID, and target ID are forwarded to the room. Room Delivers: The room finds the target client and sends the message to them

Sub-Task

Group: New Features

Task #1: Private messages via @username

100%

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

sender and reciever for private messages

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Sender Initiates Message: Sender requests a private message to a target client using sendPrivateMessage. Verify Target: The room checks if the target client exists in clientsInRoom. Format Message: The message is formatted for readability. Send to Both: The message is sent to both the sender and the target client. Handle Issues: If delivery fails, disconnected clients are removed. Log Action: The server logs the message or failure for reference.

End of Task 1

Task

100%

Group: New Features
Task #2: Mute and Unmute
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.

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

Columns: 1

Sub-Task

100%

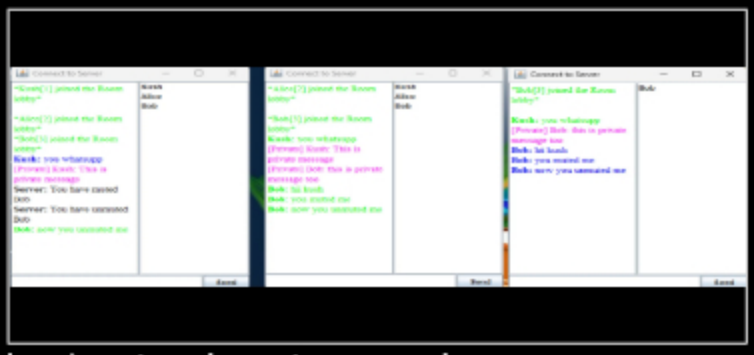
Group: New Features
Task #2: Mute and Unmute
Sub Task #1: Show a few examples across different clients (there should be at least 3 clients in the

Room)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



here is mute and unmute commands

Caption(s) (required) ✓

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

Sub-Task

100%

Group: New Features

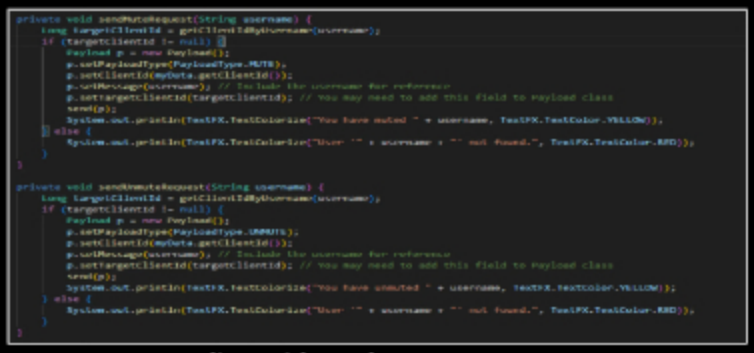
Task #2: Mute and Unmute

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
private void sendMuteRequest(String username) {
    long targetClientId = getClientIdByUsername(username);
    if (targetClientId != null) {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.MUTE);
        p.setClientId(myData.getClientId());
        p.setMessage(username); // Include the username for reference
        p.setTargetClientId(targetClientId); // You may need to add this field to Payload class
        send(p);
        System.out.println(TextFX.TextColorSize("You have muted " + username, TextFX.TextColor.YELLOW));
    } else {
        System.out.println(TextFX.TextColorSize("User '" + username + "' not found.", TextFX.TextColor.RED));
    }
}

private void sendUnmuteRequest(String username) {
    long targetClientId = getClientIdByUsername(username);
    if (targetClientId != null) {
        Payload p = new Payload();
        p.setPayloadType(PayloadType.UNMUTE);
        p.setClientId(myData.getClientId());
        p.setMessage(username); // Include the username for reference
        p.setTargetClientId(targetClientId); // You may need to add this field to Payload class
        send(p);
        System.out.println(TextFX.TextColorSize("You have unmuted " + username, TextFX.TextColor.YELLOW));
    } else {
        System.out.println(TextFX.TextColorSize("User '" + username + "' not found.", TextFX.TextColor.RED));
    }
}
```

Mute , Unmute client side code

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Find Target: The client searches for the target user's ID by username. Create Payload: It creates a payload with the MUTE or UNMUTE action, including IDs and the target username. Send Request: The payload is sent to the server. Notify User: Confirmation or error messages are displayed to the sender.

Sub-Task

Group: New Features

Task #2: Mute and Unmute

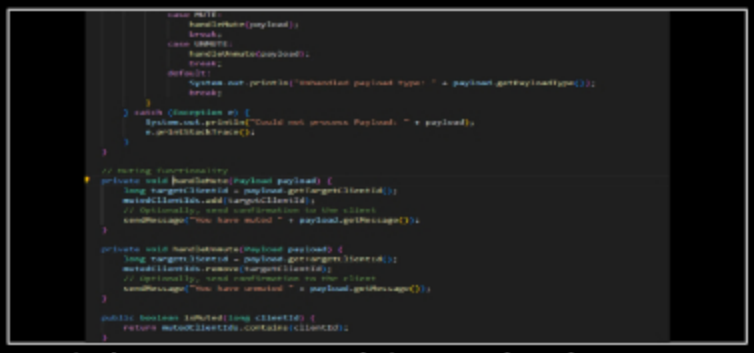
100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Here is the mute unmute code in serverthread

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Receive Payload: ServerThread gets the MUTE or UNMUTE payload from the client. Identify Target: It extracts the target client ID from the payload. Update State: Adds or removes the target ID from the muted list of the sender. Confirm: Optionally sends feedback to the sender.

Sub-Task

100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
// Muting functionality
private void handleMute(Payload payload) {
    long targetClientId = payload.getTargetClientId();
    mutedClientIds.add(targetClientId);
    // Optionally, send confirmation to the client
    sendMessage("You have muted " + payload.getMessage());
}
//KR553 11/23/2024

private void handleUnmute(Payload payload) {
    long targetClientId = payload.getTargetClientId();
    mutedClientIds.remove(targetClientId);
    // Optionally, send confirmation to the client
    sendMessage("You have unmuted " + payload.getMessage());
}
```

code that appends mute client list and unmute

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Client Sends Mute/Unmute Request: The client sends a payload to mute/unmute a target user using their ID. ServerThread Processes Request: The handleMute or handleUnmute method in ServerThread updates the mutedClientIds set. Room Sends Messages: The Room class iterates through connected clients. Check Muted Status: Before sending, Room calls isMuted on the recipient's ServerThread. Skip Muted Messages: Messages from muted users are ignored by the recipient.

Sub-Task

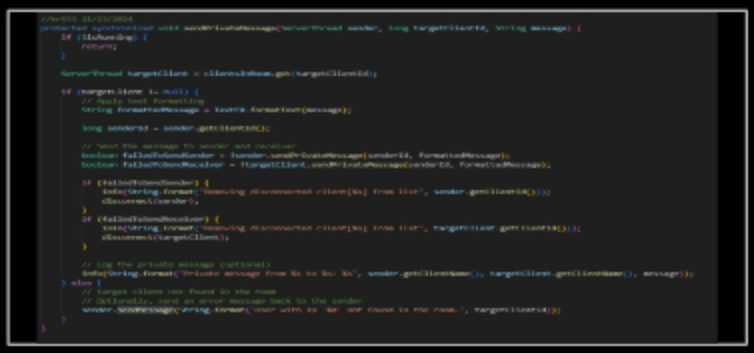
100%

Group: New Features
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



here is the code

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Public Messages: sendMessage checks if a client muted the sender using isMuted(senderId). Skips sending the message to muted clients. Sends messages only to non-muted clients. Private Messages: sendPrivateMessage checks if the recipient muted the sender. If muted, informs the sender and skips delivering the message. This ensures muted users' messages are filtered.

Sub-Task

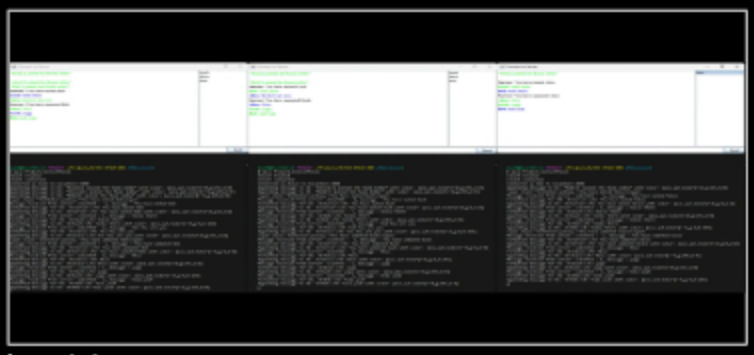
100%

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



here it is

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

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/KushDev19/kr553-IT114-005/pull/8

URL
https://github.com/KushDev19/kr553-IT114-005/

End of Task 1

Task

Group: Misc

100%

Task #2: Talk about any issues or learnings during this assignment
Weight: ~33%
Points: ~0.33

COLLAPSE

Task Response Prompt

Response:

many issue with formating messages in the Chat panel, too much researched about it and figured out the way. other issues with the chat panel being displayed and color formatting.

End of Task 2

Task

100%

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

phewww!! intense week

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment