Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-milestone-3-chatroom-2024-m24/grade/kh465

Course: IT114-005-F2024
Assigment: [IT114] Milestone 3 Chatroom 2024 M24
Student: Keven H. (kh465)

Submissions:

Submission Selection

1 Submission [submitted] 11/29/2024 6:38:55 PM

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.

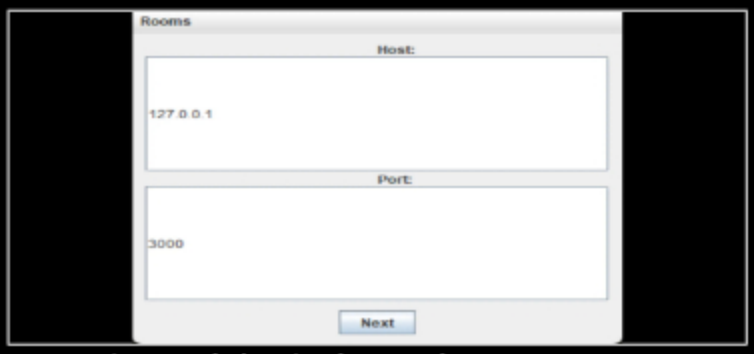
Columns: 1

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 showing host and port

Caption(s) (required) ✓

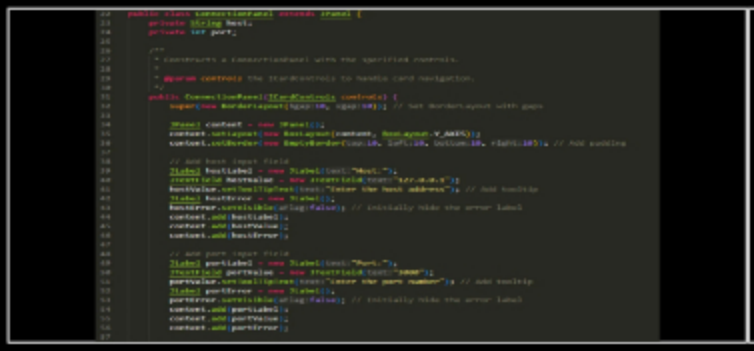
Caption Hint: *Describe/highlight what's being shown*

Sub-Task
100%
Group: Basic UI
Task #1: UI Panels
Sub Task #2: Show the code related to the ConnectionPanel

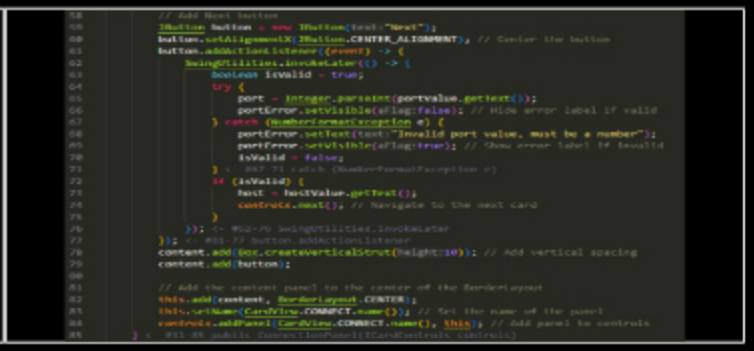
Task Screenshots

Gallery Style: 2 Columns

4 2 1



First part of the ConnectionPanel code



Second part of the ConnectionPanel 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

Response:

ConnectionPanel takes a parameter of ICardControls, which creates various parts of the ConnectionPanel UI. It creates a host and port textbox, populates them with default values, and also creates errors but hides them unless it fails a boolean check.

Sub-Task

100%

Group: Basic UI

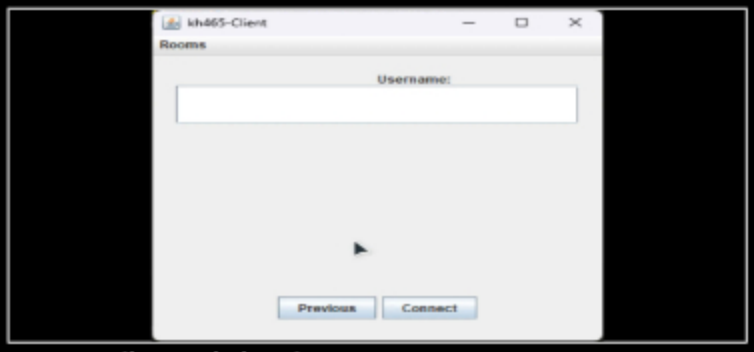
Task #1: UI Panels

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



UserDetailsPanel showing username

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Basic UI

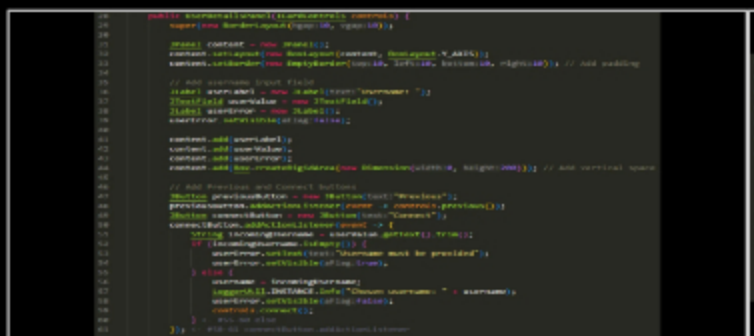
Task #1: UI Panels

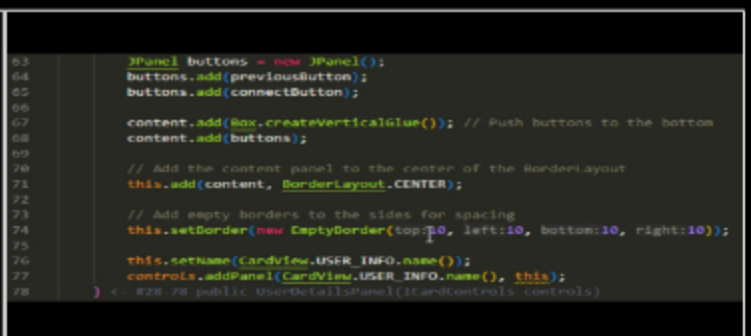
Sub Task #4: Show the code related to the UserDetailsPanel

Task Screenshots

Gallery Style: 2 Columns

4 2 1





First part of the UserDetailsPanel code

Second part of the UserDetailsPanel 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

Response:

UserDetailsPanel takes a parameter of ICardControls, which creates the UI for the username panel. It sets the layout, its alignment and its border. It creates a username textbox, waits for it to populate, and has a hidden error that only sets itself to be visible if the username field is left empty or if it contains a space

Sub-Task

100%

Group: Basic UI

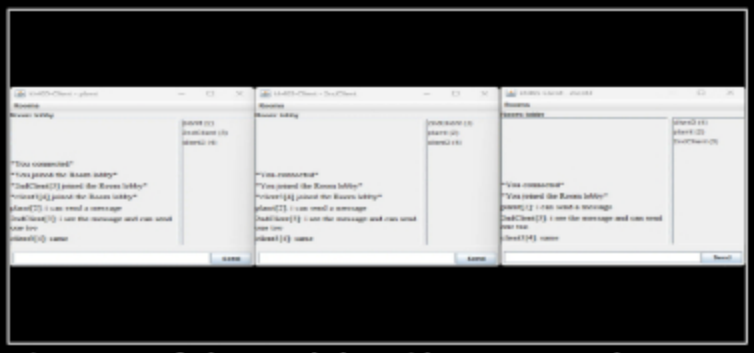
Task #1: UI Panels

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



3 instances of ChatPanel alongside some example messages

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Basic UI

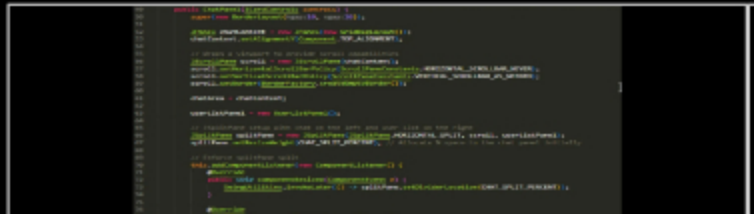
Task #1: UI Panels

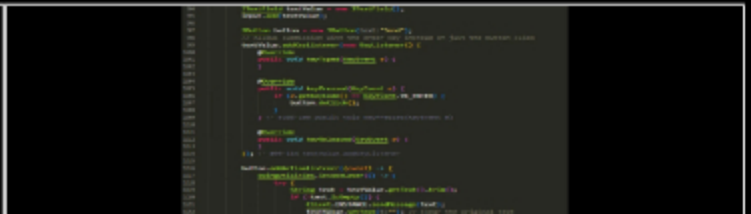
Sub Task #6: Show the code related to the ChatPanel

Task Screenshots

Gallery Style: 2 Columns

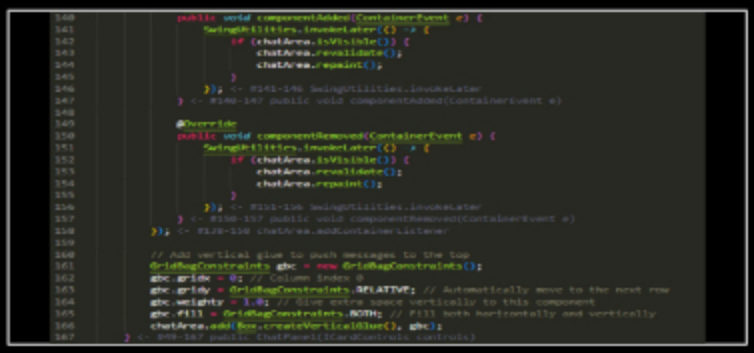
4 2 1





First part of the ChatPanel code

Second part of the ChatPanel code



Third part of the ChatPanel 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 is created with ICardControls as an argument, creating chatContent where the messages are, along with a scroll that sets the scrollbar to appear whenever items overflow off screen. An input field is also created where the user can type their message out. This can be done through the send button, or through pressing the enter key. addActionListener is responsible for sending the message off and clearing the input field. addText adds user messages to the chat area, and scrolls the messages ascending

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.

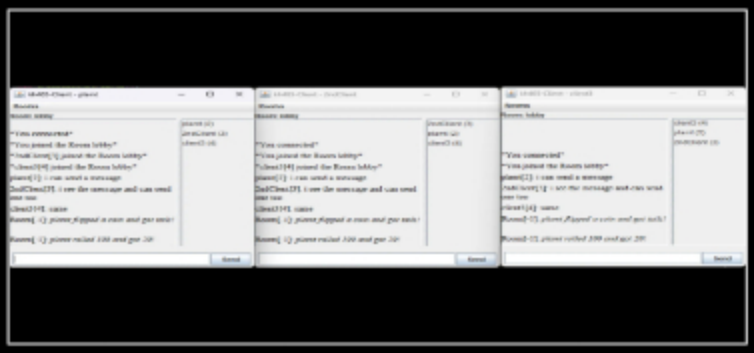
Columns: 1

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



/roll and /flip italicised. All clients can see the format and message

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

Code in Room checking for formatting. Note that the transformations are done via TextMarkup. This calls the method on message.

TextMarkup showing patterns, their replacements, and the logic for replacing them

handleRoll and handleFlip wrapping the messages in italics, which is handled by sendMessage

Caption(s) (required) ✓

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

Task Response Prompt

Explain what you did and how it works

Response:

/roll, /flip and messages all call the sendMessage method, so message is checked for specific characters (e.g. **) which then calls a method on TextMarkup (Room creates a TextMarkup object). TextMarkup has patterns for specific formats, and replacements for those formats. TMFormat loops through the MARKUP pattern array, and if it finds a match, it replaces those special characters with their HTML equivalents. handleRoll and handleFlip wrap the messages in HTML italics, so TMFormat does not need to replace it.

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

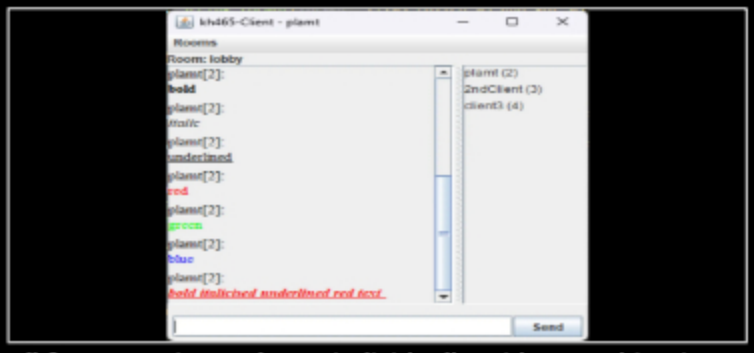
100%

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



All format variants shown individually, with a combination of all as the last message

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Build-up

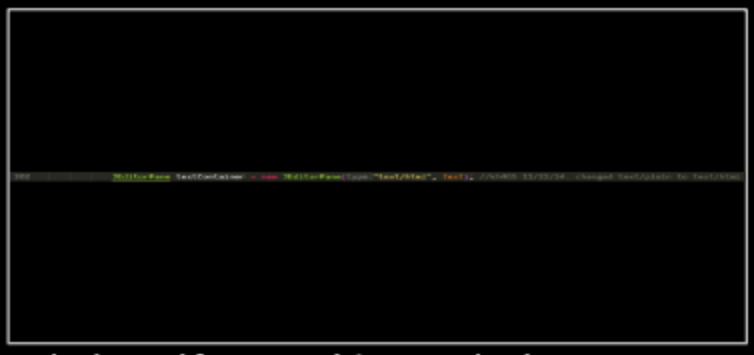
Task #2: Text Formatting appears correctly on the UI

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Code changed from text/plain to text/html

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain what was necessary and how it works

Response:

JEditorPane needed to be changed from text/plain to text/html. Text/plain does not support HTML formatting, but text/html does. Wrapping the text in HTML formatting now displays the content as intended.

End of Task 2

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

Group

75%

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

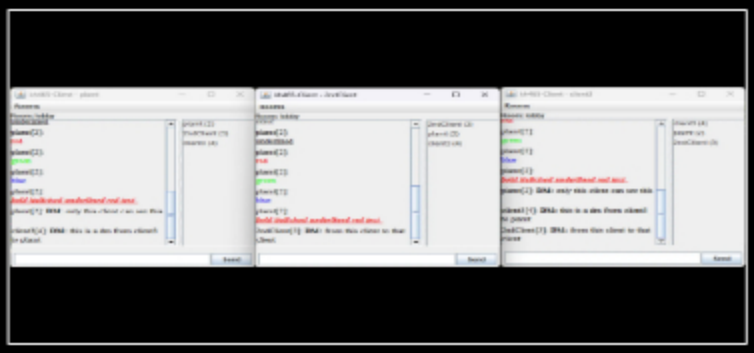
100%

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



Three instances of clients DMing individual clients

Caption(s) (required) ✓

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

Sub-Task

100%

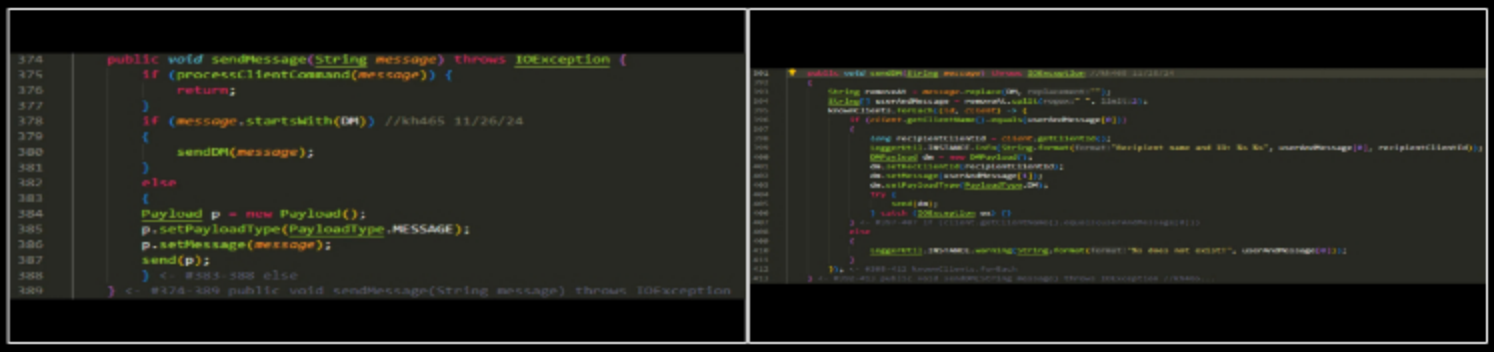
Group: New Features
Task #1: Private messages via @username
Sub Task #2: Show the client-side code that processes the text per the requirement

Task Screenshots

Gallery Style: 2 Columns

Gallery Style: 2 Columns

4 2 1



sendMessage checking if message starts with @. If it does, sendDM. Detailed explanation explained below it's redirected to sendDM

processPayload directing DM payload type to processMessage

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

sendDM takes the message, replaces the @ with "", and is passed to a String array that separates by space and has a limit of 2. A lambda expression is then used to loop through each knownClient and if their name equals the first part of the String array, their getClientId method is called and stored in recipientClientId, which is added to a DMPayload and sent along to ServerThread.

Sub-Task

100%

Group: New Features

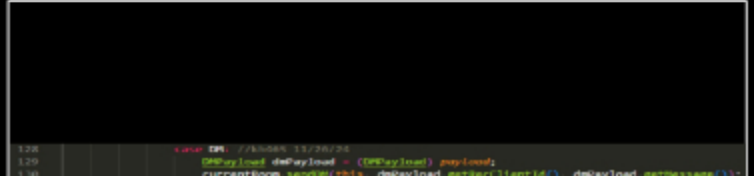
Task #1: Private messages via @username

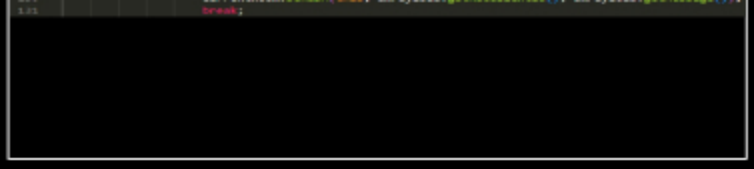
Sub Task #3: Show the ServerThread code receiving the payload and passing it to Room

Task Screenshots

Gallery Style: 2 Columns

4 2 1





ServerThread extracts the senderId, recipientClientId and message and sends it along to Room

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

When ServerThread receives a PayloadType of DM, the DM case is used, casting the current payload into a DMPayload, and calling the currentRoom.sendDM method with the senderId, recipientClientId and message extracted from the payload and sent along to the Room

Sub-Task

100%

Group: New Features

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

sendDM. Further explanation below

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

sendDM uses a lambda expression for each clientsInRoom, checking their clientId against the recipientClientId. If it matches, that clients' sendMessage method gets called with the senderId and their message, appended with **DM:** to denote that it's a DM from the specified client. This check is also done for the senderId and the same thing is done for clarity, otherwise the sender would not see their own message.

End of Task 1

Task

50%

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

0%

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

Missing Caption

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

Sub-Task

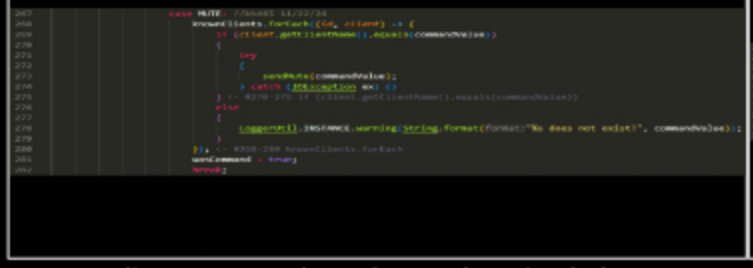
100%

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



processClientCommand. Further explanation below

sendMute assigning mutedClientId from given client name

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Using processClientCommand, the commandValue gets passed to the method sendMute. sendMute creates a new DMPayload, creates a long called mutedClientId which is assigned to the first entry found that matches the name passed along, sets the recClientId to the mutedClientId, sets the PayloadType as MUTE, and sends it to ServerThread. **Note:** I did not create a sendUnmute method since I could not create logic to actually mute a client. Sorry :(

Sub-Task

100%

Group: New Features

Task #2: Mute and Unmute

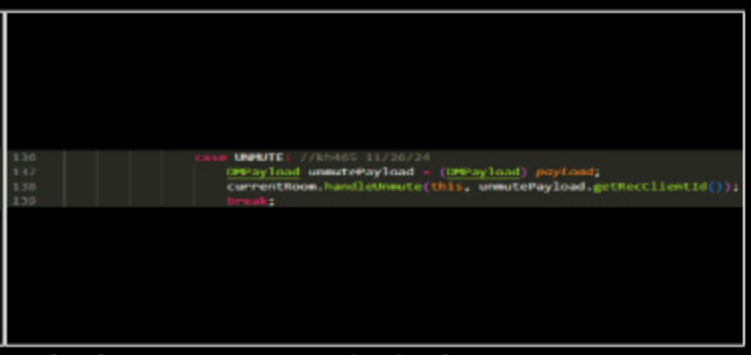
Sub Task #3: Show the ServerThread code receiving the payload and passing it to Room

Task Screenshots

Gallery Style: 2 Columns

4 2 1

PayloadType MUTE passing its data to Room



PayloadType UNMUTE passing its data to Room

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

payload is casted to DMPayload, which calls the handleMute method in currentRoom and passes the senderId alongside the recClientId (mutedClientId). The same code is used for processPayload case UNMUTE

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

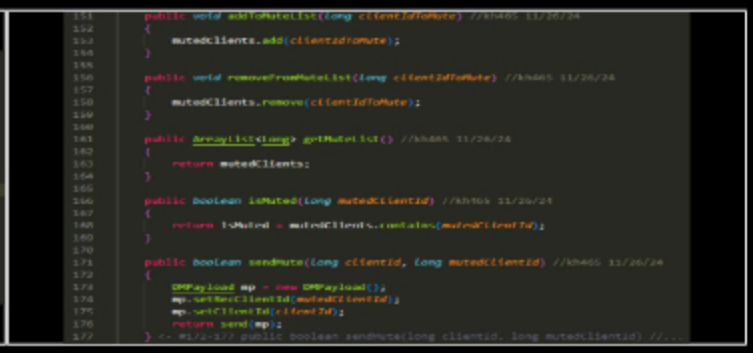
Gallery Style: 2 Columns

4 2 1

```
protected void handleMute(ServerThread sender, long clientToMute) //kh465 11/26/24
{
    clientsInRoom.forEach((id, client) -> {
        if(client.getClientId() == clientToMute)
        {
            sender.addToMuteList(clientToMute);
        }
    });
}

protected void handleUnmute(ServerThread sender, long clientToUnmute) //kh465 11/26/24
{
    clientsInRoom.forEach((id, client) -> {
        if(client.getClientId() == clientToUnmute)
        {
            sender.removeFromMuteList(clientToUnmute);
        }
    });
}
```

handleMute and handleUnmute calling methods from ServerThread



```
public void addToMuteList(long clientIdToMute) //kh465 11/26/24
{
    mutedClients.add(clientIdToMute);
}

public void removeFromMuteList(long clientIdToMute) //kh465 11/26/24
{
    mutedClients.remove(clientIdToMute);
}

public ArrayList<Long> getMuteList() //kh465 11/26/24
{
    return mutedClients;
}

public Boolean isMuted(long mutedClientId) //kh465 11/26/24
{
    return isMuted = mutedClients.contains(mutedClientId);
}

public Boolean sendMute(long clientId, long mutedClientId) //kh465 11/26/24
{
    DMPayload mp = new DMPayload();
    mp.setRecClientId(mutedClientId);
    mp.setClientId(clientId);
    return send(mp);
} <-- #1/2-1// public boolean sendMute(long clientId, long mutedClientId) //...
```

Code in ServerThread relating to the 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:

processPayload in ServerThread extracts the senderId and recClientId for both MUTE and UNMUTE and sends them to their respective methods. In Room, handleMute loops through each clientsInRoom and if their getClientId matches the long clientToMute, it calls that ServerThreads' addToMuteList method. Similar code is used for handleUnmute, except it removes clientToUnmute via removeFromMuteList

Sub-Task

0%

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

Missing Caption

Missing Caption

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

Task Response Prompt

Explain in concise steps how this logically works
Response:
Missing Response

Sub-Task
0%
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

Missing Caption

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

End of Task 2

End of Group: New Features
Task Status: 1/2

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/kh465/kh465-IT114-005/pull/14

URL
https://github.com/kh465/kh465-IT114-005/pull/

End of Task 1

Task

100%

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

COLLAPSE

Task Response Prompt

Response:

The biggest issues I encountered during the assignment was the creation of DMs and muting/unmuting users. I was able to overcome DMing users but I was unable to figure out how to create logic to actually mute and unmute users.

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

WakaTime showing the hours spent over the last 7 days

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment