Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-005-F2024/it114-milestone-3-chatroom-2024-m24/grade/el286

Course: IT114-005-F2024
Assigment: [IT114] Milestone 3 Chatroom 2024 M24
Student: Erik L. (el286)

Submissions:

Submission Selection

1 Submission [submitted] 12/10/2024 9:38:21 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 by running the app (have host/port)

Caption(s) (required) ✓

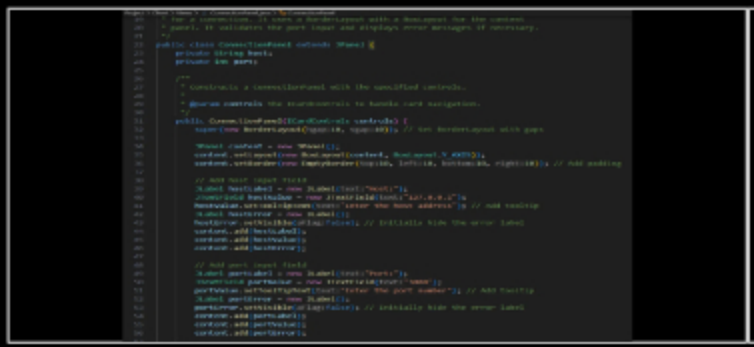
Caption Hint: *Describe/highlight what's being shown*

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

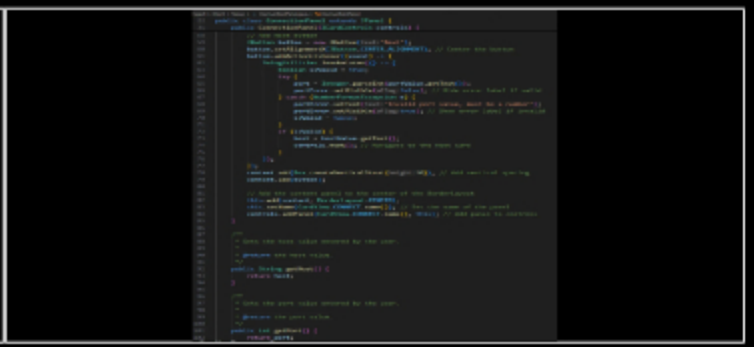
Task Screenshots

Gallery Style: 2 Columns

4 2 1



code related to the ConnectionPanel



code related to the ConnectionPanel

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:

Based on the code, a try-catch is used to set the data appropriately from both the host and the port. If everything is validated then it will proceed in doing the next() which will move on to the next card. It's then registered by using the client ui which is the addPanel(). These cards will have set names in which it can be manuevered whether going back or forth. The getHost() and getPort() are used to store the values of which it's going to be extracted by the client ui.

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 by running the app (have username)

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Basic UI

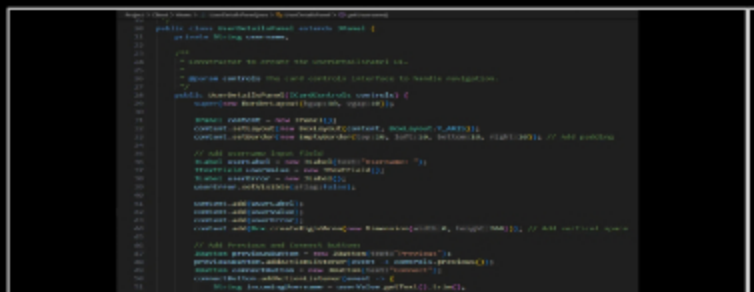
Task #1: UI Panels

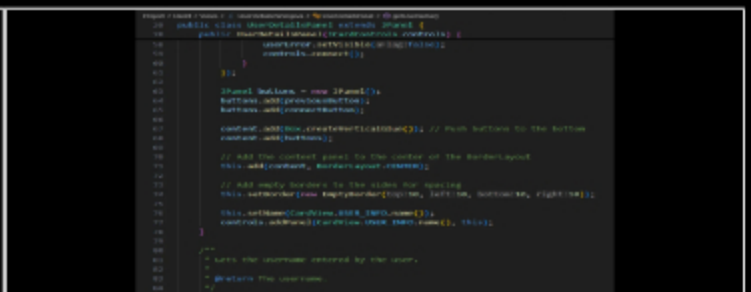
Sub Task #4: Show the code related to the UserDetailsPanel

Task Screenshots

Gallery Style: 2 Columns

4 2 1





code related to the UserDetailsPanel

code related to the 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:

Based on this code, the username is first declared and then reassigned with incomingUsername if the name is validated correctly. The connect() is then called if it all succeeds. Most of the add() and setBorder() is ismply layout towards the ui. The name is being set in the name and the card is being created by using the addPanel.

Sub-Task

100%

Group: Basic UI

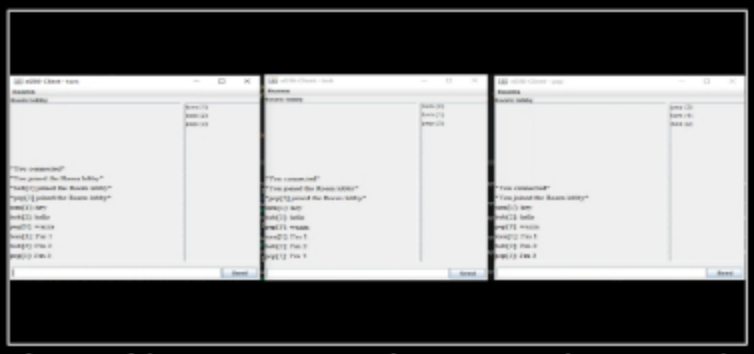
Task #1: UI Panels

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



ChatPanel (3 users present and some example messages)

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Basic UI

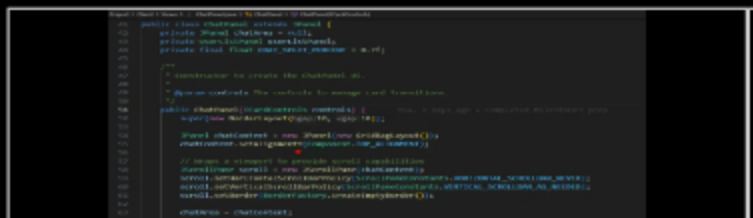
Task #1: UI Panels

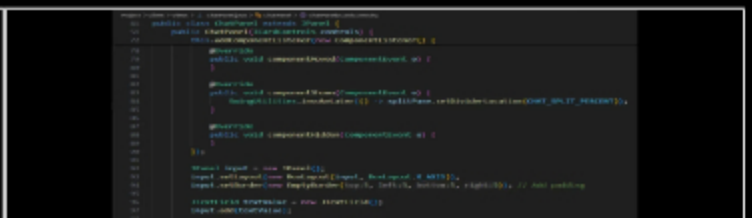
Sub Task #6: Show the code related to the ChatPanel

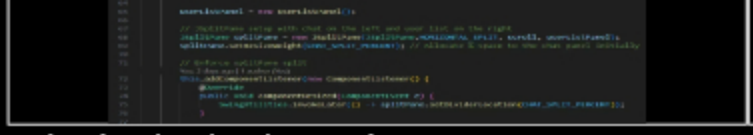
Task Screenshots

Gallery Style: 2 Columns

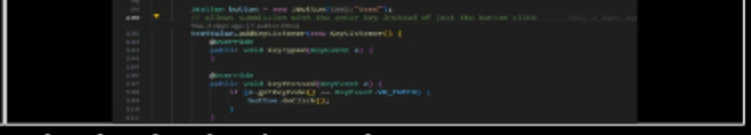
4 2 1







code related to the ChatPanel 1



code related to the ChatPanel 2

code related to the ChatPanel 3

code related to the ChatPanel 4

code related to the ChatPanel 5

code related to the ChatPanel 6

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 chat is set to a split of 70% and JSplitPane is used to add the scroll bar and the usrListPanel. The resize weight is based on the intialized chat split percentage and addComponentListener() is used so when it's resized or shown, it can promptly use SwingUtilities to set the divder location back to the original percentage so that sizing in general can be adjusted properly. When the button is clicked, the addActionListener is handles it where the text is passed using sendMessage() from the Client and if there's an exception then a logger will be performed. It then adds the splitPane where we have the client content and the userlist. Again the card will have a set name and the client ui will add the panel. When a component is added or removed it updates the ui by using revalidate() and repaint(). GridBagConstraints() is used in order to have the correct flow of the messages which will be in a vertical position. addUserListItem(), removeUserListItem(), and clearUserList() are then used to invoke the userListPanel where information will be passed on to it. Then the layout of the text is then peformed along with the content type and text based from the JEditorPane().

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

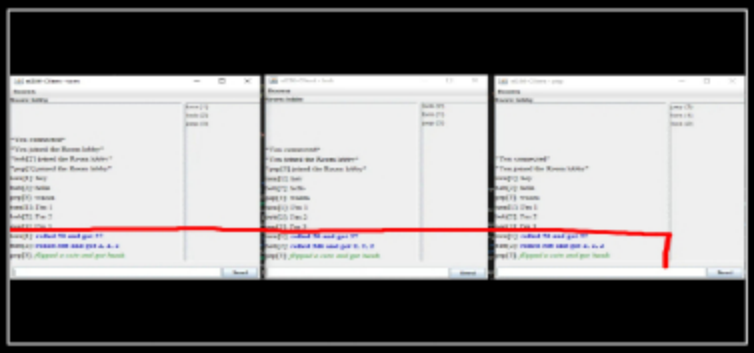
100%

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



/flip and /roll appear in a different format than regular chat text

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Roll format

Flip format

Caption(s) (required) ✓

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

Task Response Prompt

Explain what you did and how it works

Response:

Inside the sendMessage() I simply inserted the format for having color or other values like bold, italic, underline, which is in the String message part of the sendMessage(). Since it's the same format for what's in the messageFormated() then I can apply it towards the other sendMessage() where it corresponds on the roll and flip. Used String format to get the values of the rollcommand, result, and result of the flip.

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%

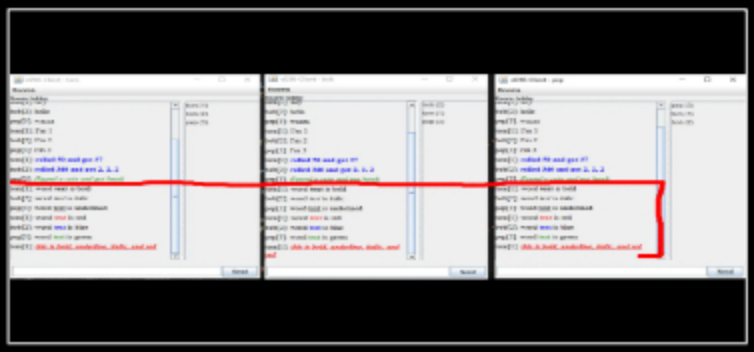
Group: Build-up
Task #2: Text Formatting appears correctly on the UI
Sub Task #1: Show examples of bold, italic, underline, each color implemented and a combination of bold, italic, underline, and one color in the same message

Task Screenshots

Gallery Style: 2 Columns

Gallery Style: 2 Columns

4 2 1



examples of bold, italic, underline, each color implemented and a combination of bold, italic, underline, and one color in the s

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Build-up

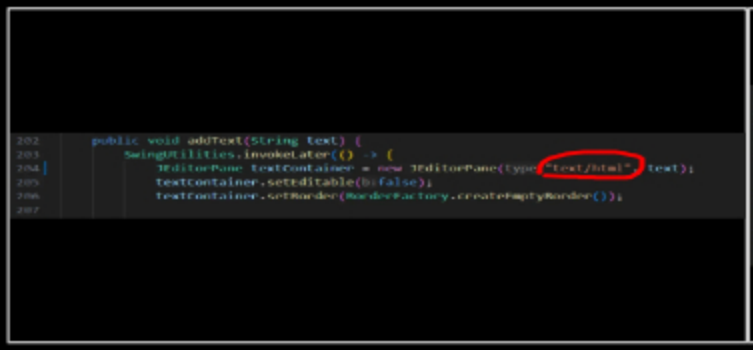
Task #2: Text Formatting appears correctly on the UI

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Change in ChatPanel

Change in Room code, messageFormated method

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain what was necessary and how it works

Response:

In order for the html tags to be registered, I had to change value inside the JEditorPane method from "text/plain" to "text/html". It will now understand the html tags and register its respective tag which is bold,italic, and underline. Another thing I changed was the color tags where it was etc. to be <span style='color:red';>. The reason for this is because tags like or any other color tag formatted like just simply won't register even in html and display nothing but default color black. Using span tag is effective for styling and in this case it's good to mark up a part of text and further use inline css inside the span tag for the specific color desired.

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

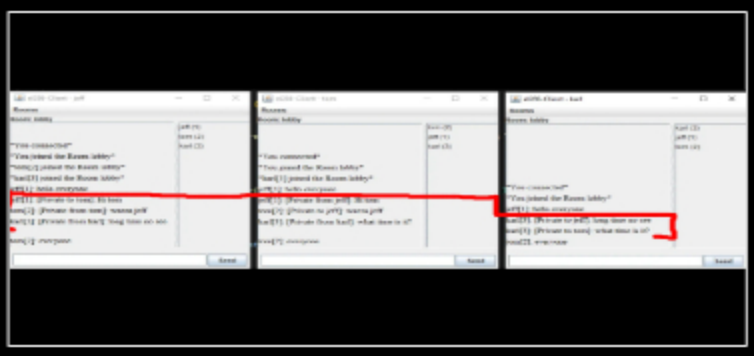
100%

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



@username examples across different 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

4 2 1

client-side code that processes the text per the requirement

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Initially the text is checked to see if it's equal to PRIVATE which is a variable set to "@" and will follow the proceeding logic. If it isn't it will log a message saying the format is "invalid" The text is split and trimeed into parts which are part[0] (the intended username) and part[1] (the message being sent). Using a for-each loop which will interate over a collection of knownClients and its values of ids. The username is then compared to the intended username and if matched, then the clientId is stored in the targetClientId variable. If there isn't a match, then it log "user not found". A Payload object is created to have access to the setters and place in the values of "PayloadType.PRIVATE_MESSAGE", clientId, and the message. Used a try-catch to send the payload using the send().

Sub-Task

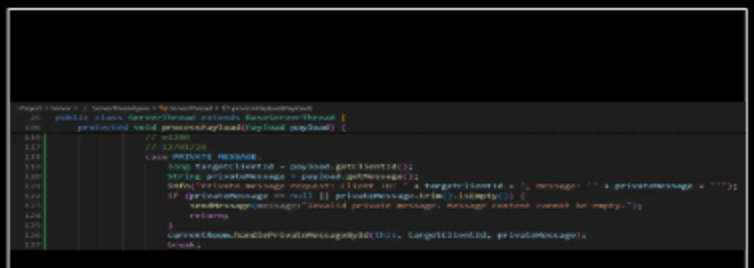
100%

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 code receiving the payload and passing it 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:

targetClientId and privateMessage are intialized with the getter values. info() is used to log the request of the private message being done. The message is then validated to make sure the message isn't null or empty. It's then passing the message to the room by using the currentRoom object which uses the handlePrivateMessage method with its respective arguments which has the sender, the intended target, and the mesage.

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

Room code that verifies the id and sends the message to both the sender and receiver

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The recipient client which is the targetClientId is looked for in the clinetsInRoom map and then determined if it's in the room or not. I intialized two strings that hold a sender format message and a recipient format mesasge that include their name and message content. Booleans are used to check if the message is actually processed/sent using send() and if it isn't it will log and info() stating that it faild. It'll then disconenct the the sender. Overall, the sender sends the private message, it makes sure that the client exists in the room and the room is valid. If everything is validated properly, it the recipient will get a private message and the sender will also get a message of the message that was sent.

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

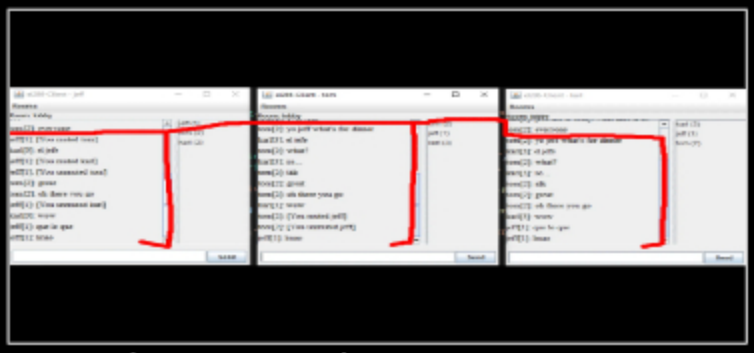
100%

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



/mute and /unmute examples

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

mute and 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:

The mute/unmute have similar structure logic like the private message where in this case, it checks to see if the value of the command is empty or not and will then be trimmed and loop through the knownClients to check if it matches with the username and retrive the id of the target. If the targetClientId isn't null, then a new Payload object is created to determine the PayloadType to be MUTE or UNMUTE and have the ClientId to be set to the value of the intended target id. It will then try to send the payload using send() and log the "muted" or "unmuted" user or "failed".

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

```
public class ServerThread extends BaseServerThread {
    protected void processPayload(Payload payload) {
                // e1286
                // 12/01/24
                case MUTE:
                    long muteTargetId = payload.getClientId();
                    currentRoom.handleMute(this, muteTargetId);
                    break;
                // e1286
                // 12/01/24        You, 14 hours ago • Uncommitted chan
                case UNMUTE:
                    long unmuteTargetId = payload.getClientId();
                    currentRoom.handleUnmute(this, unmuteTargetId);
                    break;
                default:
                    break;
            }
```

ServerThread code receiving the payload and passing it to Room

```
public class ServerThread extends BaseServerThread {
    // e1286
    // 12/01/24
    public boolean addMutedClient(String clientName) {
        boolean wasAdded = mutedClients.add(clientName);
        if (wasAdded) {
            info("Client '" + clientName + "' muted.");
        }
        return wasAdded;
    }
    // e1286
    // 12/01/24
    public boolean removeMutedClient(String clientName) {
        boolean wasRemoved = mutedClients.remove(clientName);
        if (wasRemoved) {
            info("Client '" + clientName + "' unmuted.");
        }
        return wasRemoved;
    }
    // e1286
    // 12/01/24
    public boolean isMuted(String clientName) {
        return mutedClients.contains(clientName);
    }
```

ServerThread code passing it 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:

The PayloadType of MUTE or UNMUTE is recieved, it calls the processPayload method. The target Id is obtained

from the payload and is it to the muteTargetId variable and passed to the room using the handleMute() with its arguments of the ServerThread instance and the target Id. A hashset is used to track the usernames of muted clients and each ServerThread has a set basically. The addMuteClient is a boolean method in which it will add a client to the mutedCients set and if passes it will log the success. Same concept goes for the removeMutedClient which are used later on in the Room.java. isMuted() will check if the client is muted based on the mutedClients set and return it.

Sub-Task

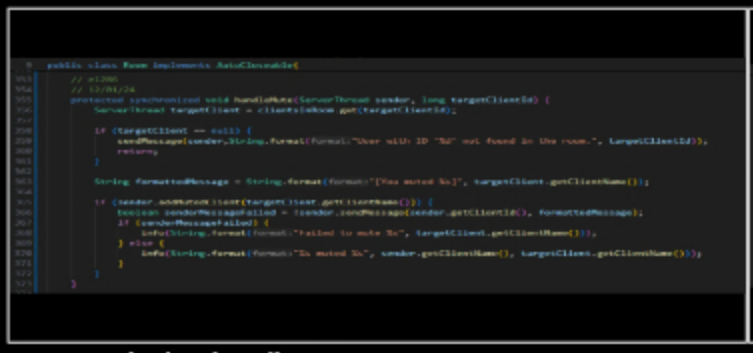
100%

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

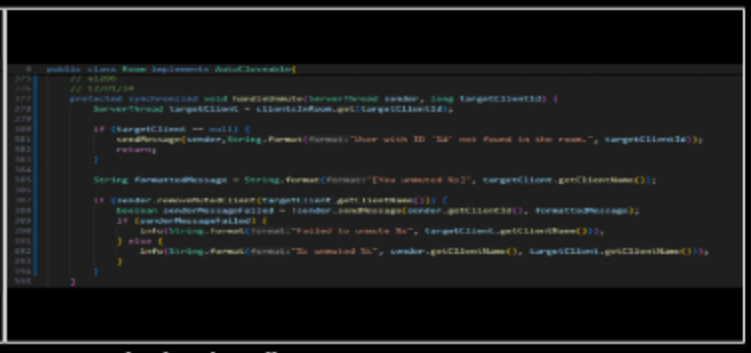
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Room code that handles Mute



Room code that handles Unmute

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

handelMute and handleUnmute have the same logic just different sendMessage but inside those merthods targetClientId is obtained from the ServerThread of the intended target and if not found it will let the sender know that user is not found. It will then use the addMutedClient() from the ServerThread to add the target client's name to go towards a muted list in the ServerThread. If successful, it will sned a message of being the sender muting the target and if not it will also get a message saying that it failed. For the handleUnmute() it will utilize the removeMutedClient which will remove the target client from muted list.

Sub-Task

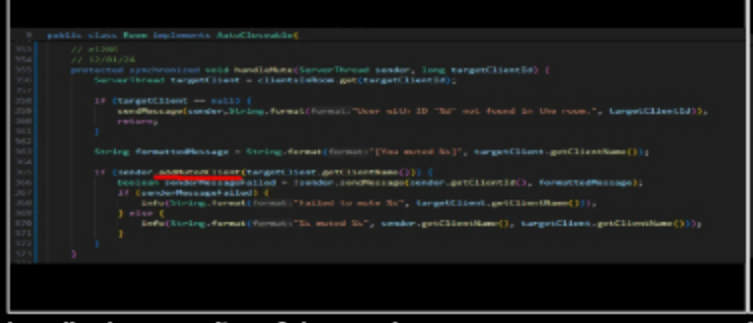
100%

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

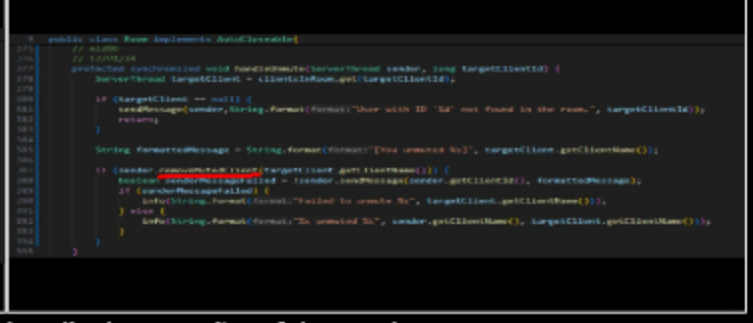
Task Screenshots

Gallery Style: 2 Columns

4 2 1



handle the mute list of the sender



handle the mute list of the sender

mute in sendMessage

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

In the sendMessage(), it checks if the recipient is muted by the sender by using the isMuted() from the ServerThread and if the client is muted, it will say that the message is skipped. The add and remove methods are the ones that have the implementation of the mute list in ServerThread.

Sub-Task

100%

Group: New Features

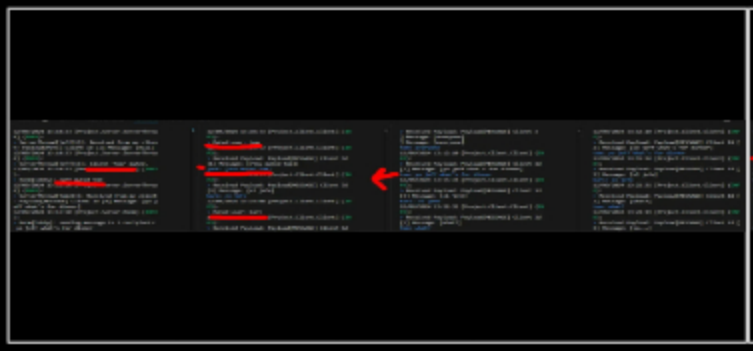
Task #2: Mute and Unmute

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



terminal supplemental evidence per the requirements

terminal supplemental evidence (message was skipped due to being muted)

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/ElLopez21/el286-IT114-005/pull/13

URL
https://github.com/ElLopez21/el286-IT114-005/p

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:

Probably the main issue would be getting the mute and unmute to be done. Although I could simply implement in the case-switch statement where it can get the commmand value and process further logic. I also noticed that the iteration

is literally similar to the iteration done in the private messagigng. The code for the ServerThread was more complicated where I didn't really know that I had to create a mute list until I reread the document and used other resources where a hash set can be implemented and work with the clients that are beign muted. I was having issues where I would have the add and remove done but it end the end it wasn't registering the client. Not only that but the message of the client that muted the intented recipient and other clients in the room also get the message which isn't suppose to happen. In order to solve that I used a boolean where it would determine if the message for either private message, mute, or unmute would be for the intented recipient and only to that client.

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

WakaTime

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment