

# Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-451-M2024/it114-milestone-4-chatroom-2024-m24/grade/st278>

IT114-451-M2024 - [IT114] Milestone 4 Chatroom 2024 M24

## Submissions:

Submission Selection

1 Submission [active] 7/30/2024 11:08:55 PM

## Instructions

^ COLLAPSE ^

- Implement the Milestone 4 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 Milestone4 branch
- Create a pull request from Milestone4 to main and keep it open until you get the output PDF from this assignment.
- Gather the evidence of feature completion based on the below tasks.
- Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
- Run the necessary git add, commit, and push steps to move it to GitHub
- Complete the pull request that was opened earlier
- Upload the same output PDF to Canvas

Branch name: Milestone4

Tasks: 7 Points: 10.00



Features (9 pts.)

^ COLLAPSE ^



Task #1 - Points: 3

Text: Client can export chat history of their current session (client-side)

^ COLLAPSE ^

### Details:

For this requirement it's not valid to have another list keep track of messages. The goal is to utilize the location where messages are already present.

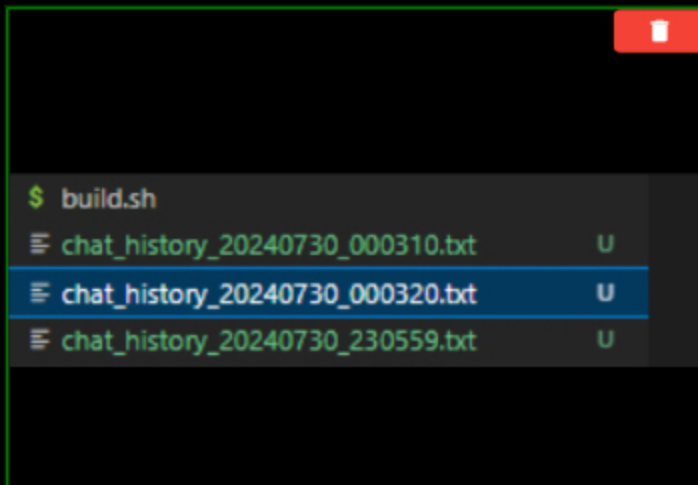
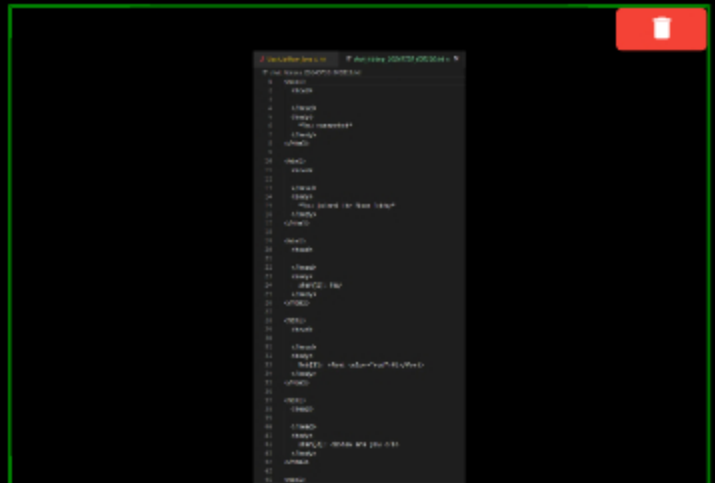
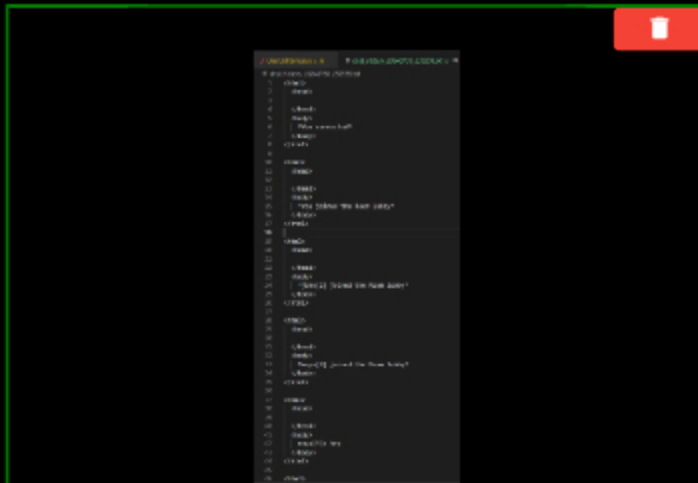
This must be a client-side implementation.

A StringBuilder must be used for consolidation.

Screenshots of editors must have the frame title visible with your ucid and the client name.

Code screenshots must have ucid/data comments.

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



**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing a few examples of exported chat history

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



```
//st278 and 07/27/24
private void exportChatHistory() {
    StringBuilder chatHistory = new StringBuilder();

    for (Component component : chatArea.getComponents()) {
        if (component instanceof JEditorPane) {
            JEditorPane textContainer = (JEditorPane) component;
            chatHistory.append(textContainer.getText()).append("\n");
        }
    }

    LocalDate now = LocalDate.now();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern: "yyyyMMdd_HHmmss");
    String fileName = "chat_history_" + now.format(formatter) + ".txt";

    try (FileWriter writer = new FileWriter(fileName)) {
        writer.write(chatHistory.toString());
        JOptionPane.showMessageDialog(this, "Chat history exported to " + fileName, title: "Export Successful", JOptionPane.INFORMATION_MESSAGE);
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(this, "Error exporting chat history: " + ex.getMessage(), title: "Export Failed", JOptionPane.ERROR_MESSAGE);
    }
}
```

### Caption (required) ✓

Describe/highlight what's being shown

Showing the code related to building the export data

### Explanation (required) ✓

Explain in concise steps how this logically works

#### PREVIEW RESPONSE

When a user clicks the "Export" button, the `exportChatHistory()` method is called, creating a `StringBuilder` to store the chat history. The method iterates through all components in the `chatArea`, extracting the text from each `JEditorPane` (representing messages) and appending it to the `StringBuilder`. A unique filename is generated using the current date and time, and a `FileWriter` is created with this filename. The `StringBuilder`'s contents are written to the file, and if successful, a success message is displayed; otherwise, an error message is shown. The file is closed automatically due to the use of `try-with-resources`. This process efficiently collects all messages, combines them into a single string, and saves them to a uniquely named file, providing a simple way for users to export their chat history.

### #3) Show the UI interaction that will trigger an export



```
107
108 //st278 and 07/27/24
109 input.setLayout(new BorderLayout(input, BorderLayout.X_AXIS));
110 input.setBorder(new EmptyBorder(top:5, left:5, bottom:5, right:5));
111
112 JtextField textValue = new JtextField();
113 input.add(textValue);
114
115 JButton button = new JButton(text:"Send");
116 exportButton = new JButton(text:"Export Chat");
117
```

### Caption (required) ✓

Describe/highlight what's being shown

Showing the UI interaction that will trigger an export

### Explanation (required) ✓

Explain where you put it any why

 PREVIEW RESPONSE

Placing the export button here makes it visible, easily accessible, and logically grouped with other chat controls, improving the usability of the chat interface.

### Task #2 - Points: 3

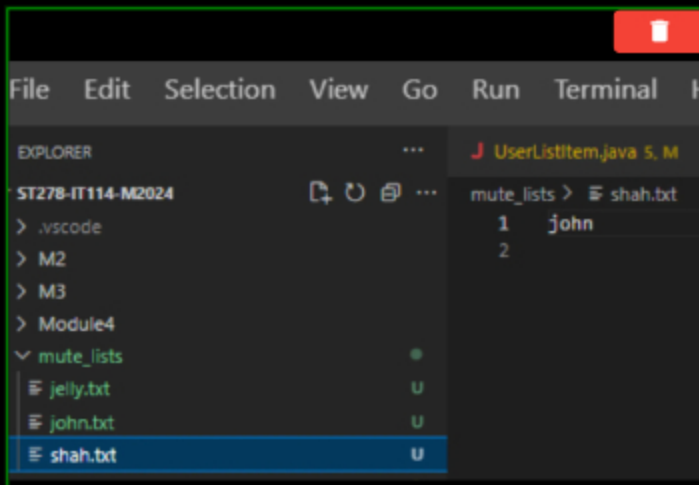
Text: Client's Mute List will persist across sessions (server-side)

### Details:

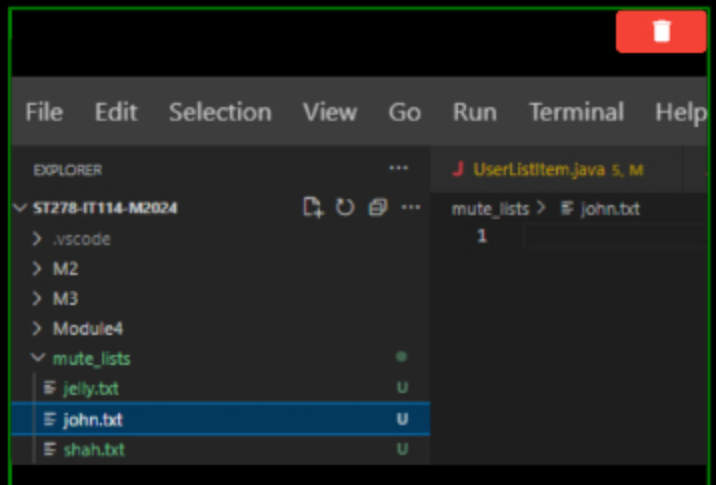
This must be a server-side implementation.

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

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



```
File Edit Selection View Go Run Terminal Help
EXPLORER
ST278-IT114-M2024
> .vscode
> M2
> M3
> Module4
v mute_lists
  jelly.txt
  john.txt
  shah.txt
```




```
File Edit Selection View Go Run Terminal Help
EXPLORER
ST278-IT114-M2024
> .vscode
> M2
> M3
> Module4
v mute_lists
  jelly.txt
  john.txt
  shah.txt
```

### Caption (required) ✓

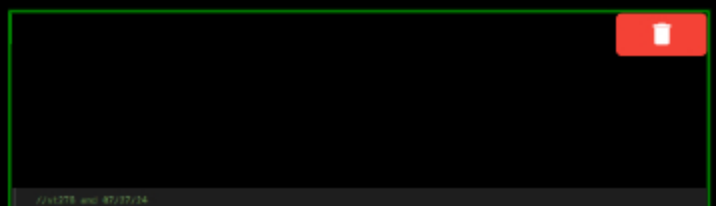
Describe/highlight what's being shown

Showing multiple examples of mutualist files and their content

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



```
//st278 and 07/27/24
```



```
//st278 and 07/27/24
```

```

public void setClientName(String name) {
    if (name == null) {
        throw new NullPointerException(s:"Client name can't be null");
    }
    this.clientName = name;
    loadMuteList(); // Loads the mute list after setting the client name
    onInitialized();
}

```

```

private void loadMuteList() {
    String fileName = MUTE_LIST_DIRECTORY + File.separator + clientName + ".txt";
    try {
        List<String> lines = Files.readAllLines(Paths.get(fileName));
        mutedUsers.clear(); // Clear existing mute list before loading
        mutedUsers.addAll(lines);
        LoggerUtil.INSTANCE.info("Loaded mute list for " + clientName);
    } catch (IOException e) {
        LoggerUtil.INSTANCE.info("No existing mute list found for " + clientName + ". Starting with an empty list.");
    }
}

```

### Caption (required) ✓

*Describe/highlight what's being shown*

Showing the code related to loading the mutelist for a connecting client

### Explanation (required) ✓

*Explain in concise steps how this logically works*

#### PREVIEW RESPONSE

When a client connects and provides a username, `setClientName()` is called, followed by `loadMuteList()`. A filename is created based on the username, and the code attempts to read all lines from the file. If successful, the existing mute list is cleared, and usernames from the file are added to it, with this success logged. If the file doesn't exist or can't be read, an empty mute list is maintained, and this is logged as information. The client now has their mute list loaded, either populated or empty, and the client initialization process continues.

## #3) Show the code related to saving the mutelist whenever the list changes for a client



```

//st278 and 07/28/24
private void saveMuteList() {
    String fileName = MUTE_LIST_DIRECTORY + File.separator + clientName + ".txt";
    try {
        Files.write(Paths.get(fileName), mutedUsers);
        LoggerUtil.INSTANCE.info("Saved mute list for " + clientName);
    } catch (IOException e) {
        LoggerUtil.INSTANCE.severe("Error saving mute list for " + clientName, e);
    }
}

```

```

//st278 and 07/28/24
public boolean addMutedUser(String username) {
    boolean added = mutedUsers.add(username);
    if (added) {
        saveMuteList();
        LoggerUtil.INSTANCE.info(clientName + " muted user: " + username);
        notifyUserOfMuteStatus(username, isMuted:true);
    }
    return added;
}

public boolean removeMutedUser(String username) {
    boolean removed = mutedUsers.remove(username);
    if (removed) {
        saveMuteList();
        LoggerUtil.INSTANCE.info(clientName + " unmuted user: " + username);
        notifyUserOfMuteStatus(username, isMuted:false);
    }
    return removed;
}

```

### Caption (required) ✓

*Describe/highlight what's being shown*

Showing the code related to saving the mutelist whenever the list changes for a client

### Explanation (required) ✓

*Explain in concise steps how this logically works*

#### PREVIEW RESPONSE

When a client mutes or unmutes a user, `addMutedUser()` or `removeMutedUser()` is called. If the mute list changes, the change is applied to the in-memory set, and `saveMuteList()` is triggered. This method constructs a filename from the client's name, writes the entire mute list to the file, and logs success or failure. The updated mute list is

from the client's name, writes the entire mute list to the file, and logs success or failure. The updated mute list is now persisted, efficiently saving the entire mute list whenever it changes to ensure persistent storage always reflects the current state of the client's mute preferences.

### Task #3 - Points: 1

Text: Clients will receive a message when they get muted/unmuted by another user

#### Details:

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

I.e., /mute Bob followed by a /mute Bob should only send one message because Bob can only be muted once until they're unmuted. Similarly for /unmute Bob

#1) Show the code that generates the well formatted message only when the mute state changes (see notes in the details above)



```
//st278 and 47/28/24
protected void handleMessage(ServerThread sender, long targetId) {
    ServerThread target = clientsKnown.get(targetId);
    if (target != null) {
        if (sender.addMutedUser(target.getClientName()) {
            sender.sendMessage("You have muted " + target.getClientName());
        } else {
            sender.sendMessage(target.getClientName() + " is already muted");
        }
    } else {
        sender.sendMessage("User not found in this room.");
    }
}

protected void handleMessage(ServerThread sender, long targetId) {
    ServerThread target = clientsKnown.get(targetId);
    if (target != null) {
        if (sender.removeMutedUser(target.getClientName()) {
            sender.sendMessage("You have unmuted " + target.getClientName());
        } else {
            sender.sendMessage(target.getClientName() + " was not muted");
        }
    } else {
        sender.sendMessage("User not found in this room.");
    }
}

public void sendPrivateSystemMessage(ServerThread sender, String targetUsername, String message) {
    for (ServerThread client : clientsKnown.values()) {
        if (client.getClientName().equals(targetUsername)) {
            client.sendMessage(ServerThread.DETAIL_03DM_03, message);
        }
    }
}
```

```
//st278 and 47/28/24
public boolean addMutedUser(String username) {
    boolean added = mutedUsers.add(username);
    if (added) {
        saveMuteList();
        LoggerUtil.INSTANCE.info(clientName + " muted user: " + username);
        notifyUserOfMuteStatus(username, isMuted:true);
    }
    return added;
}

public boolean removeMutedUser(String username) {
    boolean removed = mutedUsers.remove(username);
    if (removed) {
        saveMuteList();
        LoggerUtil.INSTANCE.info(clientName + " unmuted user: " + username);
        notifyUserOfMuteStatus(username, isMuted:false);
    }
    return removed;
}

private void notifyUserOfMuteStatus(String targetUsername, boolean isMuted) {
    String message = String.format("As %s you, clientName, isMuted ? %s : %s",
        currentRoom.sendPrivateSystemMessage(this, targetUsername, message);
}
```

#### Caption (required) ✓

Describe/highlight what's being shown

Showing the code that generates the well formatted message only when the mute state changes

#### Explanation (required) ✓

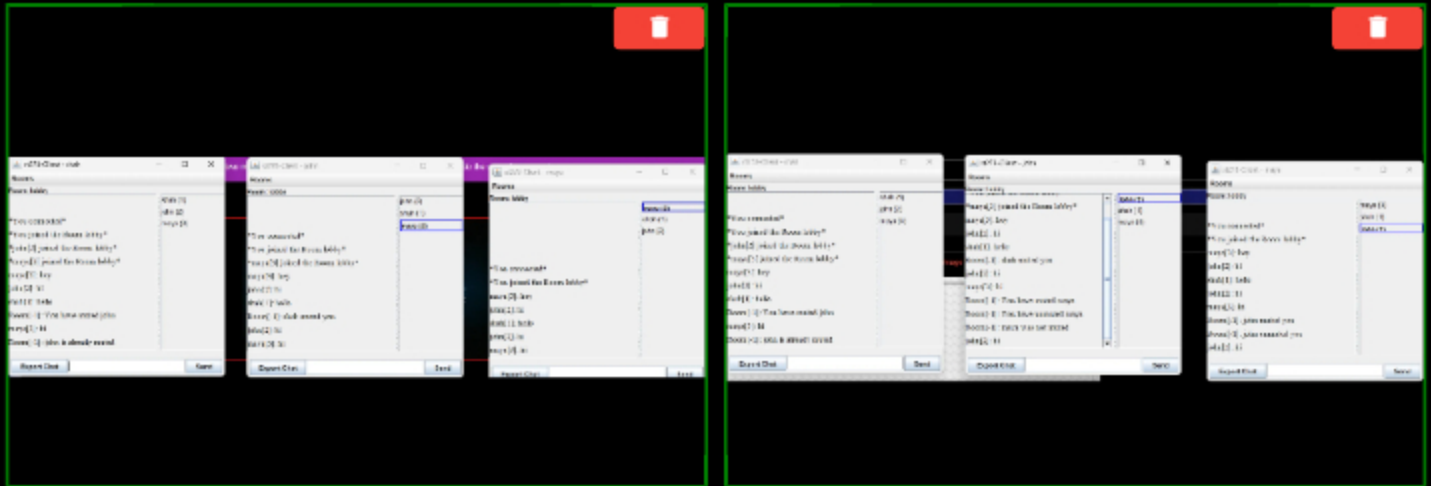
Explain in concise steps how this logically works

#### PREVIEW RESPONSE

When a user gives a /mute or /unmute command, the Room's handleMute or handleUnmute method is called. The target user is located in the room, and if found, an attempt is made to add or remove them from the mute list, checking if the list actually changed. If it did, the sender is informed of the successful action, the target is notified of their new mute status, and the updated mute list is saved. If the list didn't change, the sender is informed that the status was already set. If the target is not found, the sender is informed that the user isn't in the room. This process ensures mute status changes and notifications occur only when the mute list is actually modified, preventing redundant messages and actions.



#2) Show a few examples of this occurring and demonstrate that two mutes of the same user in a row generate only one message, do the same for unmute)



Caption (required) ✓

Describe/highlight what's being shown

Showing a few examples of this occurring

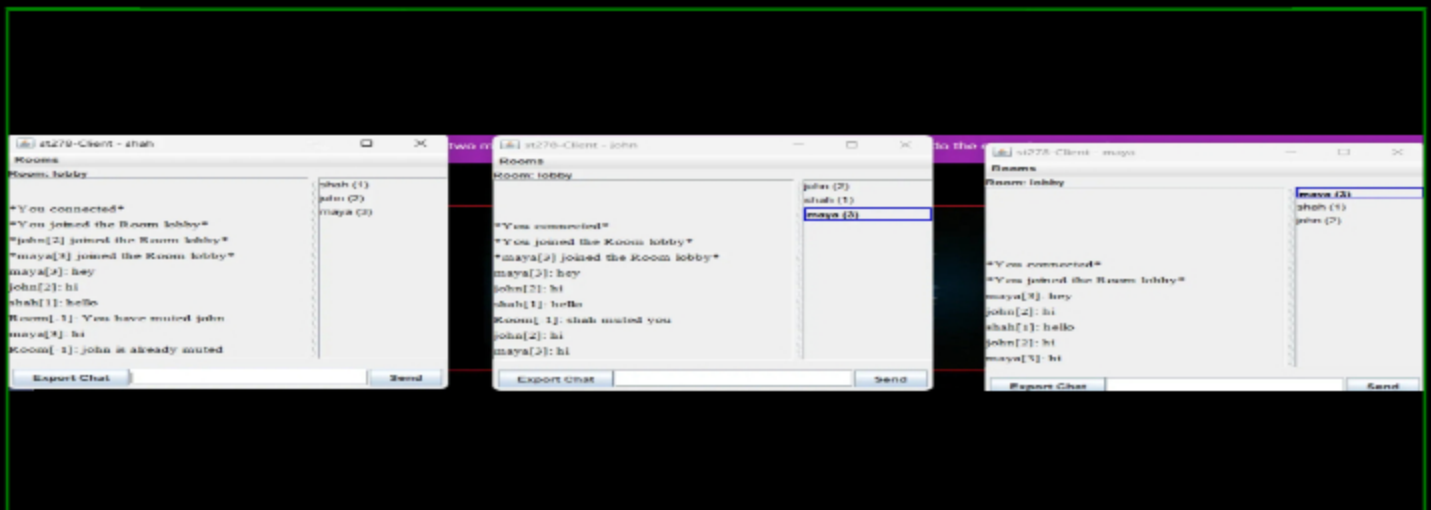
#### Task #4 - Points: 3

Text: The user list on the Client-side should update per the status of each user

#### Details:

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

#1) Show the UI for Muted users appear grayed out (or similar indication of your choosing) include a few examples showing it updates correctly when changing from mute/unmute and back



Caption (required) ✓

Describe/highlight what's being shown

Showing the UI for Muted users appears grayed out

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



```
//st278 and 07/30/24
case MUTE:
case UNMUTE:
    boolean isMuted = payload.getPayloadType() == PayloadType.MUTE;
    long targetId = payload.getClientId();
    ((ClientUI) events).onMuteStatusChange(targetId, isMuted);
    break;
```

```
//st278 and 07/30/24
public void onMuteStatusChange(long clientId, boolean isMuted) {
    if (currentCard.ordinal() >= CardView.CHAT.ordinal()) {
        chatPanel.updateUserMuteStatus(clientId, isMuted);
    }
}
```

```
public void updateUserMuteStatus(long clientId, boolean isMuted) {
    userListPanel.updateUserMuteStatus(clientId, isMuted);
}
```

```
//st278 and 07/30/24
public void updateUserMuteStatus(long clientId, boolean isMuted) {
    SwingUtilities.invokeLater(() -> {
        UserListItem item = userItems.get(clientId);
        if (item != null) {
            item.setMuted(isMuted);
        }
    });
}
```

```
//st278 and 07/30/24
public void setLastSender(boolean lastSender) {
    this.isLastSender = lastSender;
    updateAppearance(lastSender);
}

public void updateAppearance(boolean isMuted){
    if (isMuted) {
        nameLabel.setForeground(Color.RED);
    } else {
        nameLabel.setForeground(Color.BLACK);
    }

    if (isLastSender) {
        nameLabel.setFont(nameLabel.getFont().deriveFont(Font.BOLD));
    } else {
        nameLabel.setFont(nameLabel.getFont().deriveFont(Font.PLAIN));
    }

    revalidate();
    repaint();
}
```

Caption (required) ✓

Describe/highlight what's being shown

Showing the code flow (client receiving -> UI) for Muted users appear grayed out

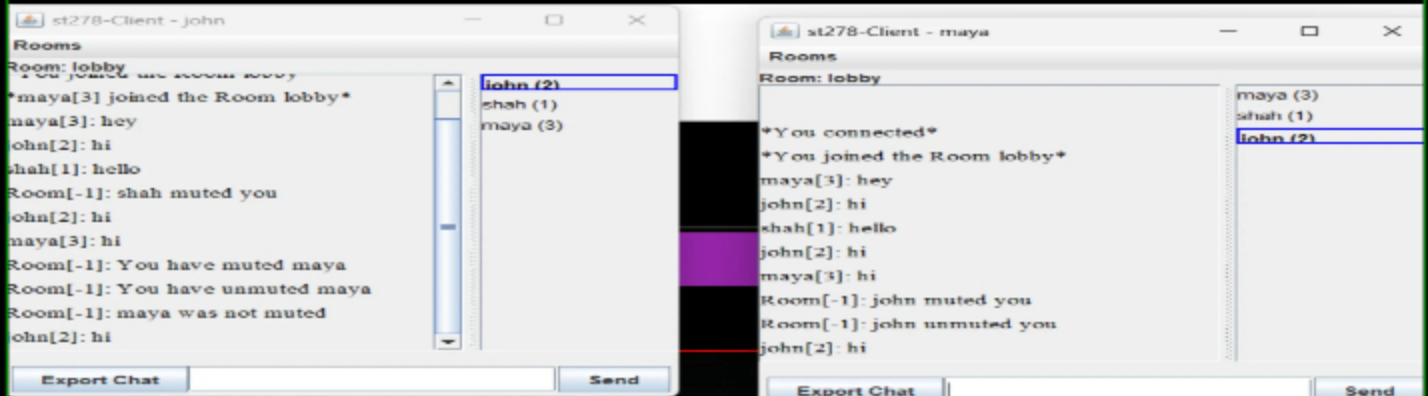
Explanation (required) ✓



## PREVIEW RESPONSE

When the client processes a mute or unmute action, it notifies the ClientUI. The ClientUI tells the ChatPanel to update, which delegates to the UserListPanel. The UserListPanel finds the correct UserListItem and updates its mute status, changing its appearance based on the new status. This process happens each time a mute or unmute action occurs, keeping the UI in sync with the current mute statuses.

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

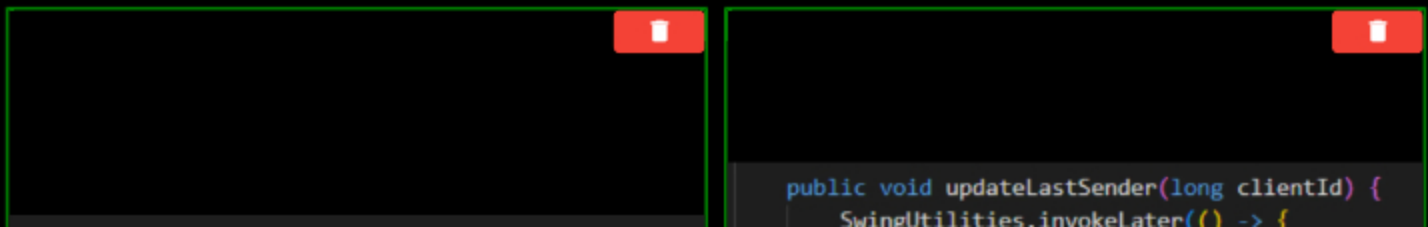
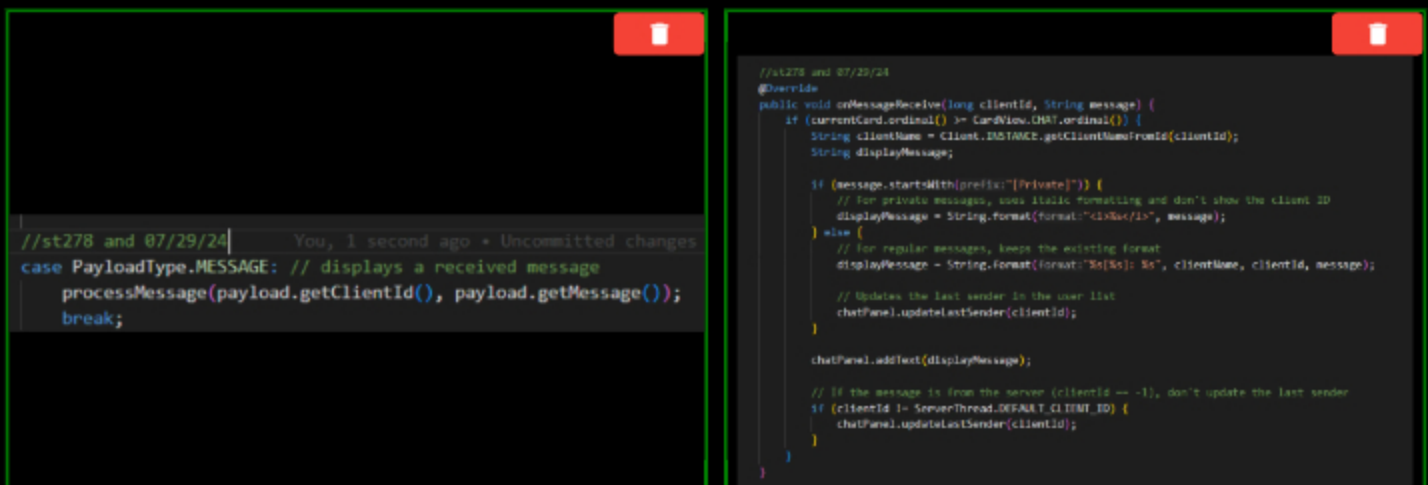


Caption (required) ✓

Describe/highlight what's being shown

Showing the UI for the Last person to send a message gets highlighted

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



```
public void updateLastSender(long clientId) {
    userListPanel.updateLastSender(clientId);
}
```

```
changeLastSenderId() {
    lastSenderId = clientId;
    updateAllUserAppearances();
}
}
```

```
//st278 and 07/29/24
public void setLastSender(boolean lastSender) {
    this.isLastSender = lastSender;
    updateAppearance(lastSender);
}

public void updateAppearance(boolean isMuted){
    if (isMuted) {
        textContainer.setForeground(Color.GRAY);
    } else {
        textContainer.setForeground(Color.BLACK);
    }

    if (isLastSender) {
        textContainer.setFont(textContainer.getFont().deriveFont(Font.BOLD));
        this.setBorder(BorderFactory.createLineBorder(Color.BLUE, thickness:2));
    } else {
        textContainer.setFont(textContainer.getFont().deriveFont(Font.PLAIN));
        this.setBorder(BorderFactory.createEmptyBorder());
    }

    this.revalidate();
    this.repaint();
}
```

#### Caption (required) ✓

*Describe/highlight what's being shown*

Showing the code flow (client receiving -> UI) for Last person to send a message gets highlighted

#### Explanation (required) ✓

*Explain in concise steps how this logically works*

##### PREVIEW RESPONSE

When a message payload is received, the client processes it and notifies the ClientUI, which tells the ChatPanel to add the message text and update the last sender. The ChatPanel adds the message to the chat area and delegates the last sender update to the UserListPanel. The UserListPanel updates the last sender, removing the highlight from the previous sender and adding it to the new one. The UserListItem changes its appearance based on whether it's the last sender. This process ensures that the UI always highlights the most recent message sender in the user list.

Misc (1 pt.)

^COLLAPSE ^

Task #1 - Points: 1

Text: Add the pull request link for the branch

#### Details:

Note: the link should end with /pull/#

URL #1

<https://github.com/st278/st278-IT114-M2024/pull/11>

URL

<https://github.com/st278/st278-IT114-M2024/pull/11>

+ ADD ANOTHER URL

^COLLAPSE ^

### Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

There was barely any issues during this milestone but being sick for the past week made it challenging to complete this assignment on time.

^COLLAPSE ^

### Task #3 - Points: 1

Text: WakaTime Screenshot

#### 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: Large View

Small

Medium

Large

Projects • st278-IT114-M2024

7 hrs 17 mins over the Last 7 Days in st278-IT114-M2024 under all branches. 

Overall time spent.

Files		Branches	
1 hr 53 mins	...ct/server/ServerThread.java	7 hrs 17 mins	Milestone
1 hr 18 mins	...client/views/ChatPanel.java		
1 hr 4 mins	Project/server/Room.java		
41 mins	...nt/views/UserListPanel.java		
38 mins	Project/client/Client.java		
33 mins	...ent/views/UserListItem.java		
22 mins	Project/client/ClientUI.java		
9 mins	mute_lists/jenny.txt		
8 mins	Project/server/Server.java		
5 mins	mute_lists/shah.txt		
5 mins	...history_20240730_000310.txt		
4 mins	...history_20240730_000320.txt		
4 mins	...024-m24_IT114-451-M2024.pdf		
3 mins	mute_lists/john.txt		

Individual files time spent

End of Assignment