Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-006-S2024/it114-chatroom-milestone-4-2024/grade/fj28

IT114-006-S2024 - [IT114] Chatroom Milestone 4 2024

Submissions:

Submission Selection

1 Submission [active] 4/26/2024 3:48:53 PM

Instructions

COLLAPSE

Implement the Milestone 4 features from the project's proposal document: https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145Xi
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: 15 Points: 10.00

● Demonstrate Chat History Export (2.25 pts.)

COLLAPSE

● Task #1 - Points: 1

Text: Screenshots of code

COLLAPSE

Checklist *The checkboxes are for your own tracking

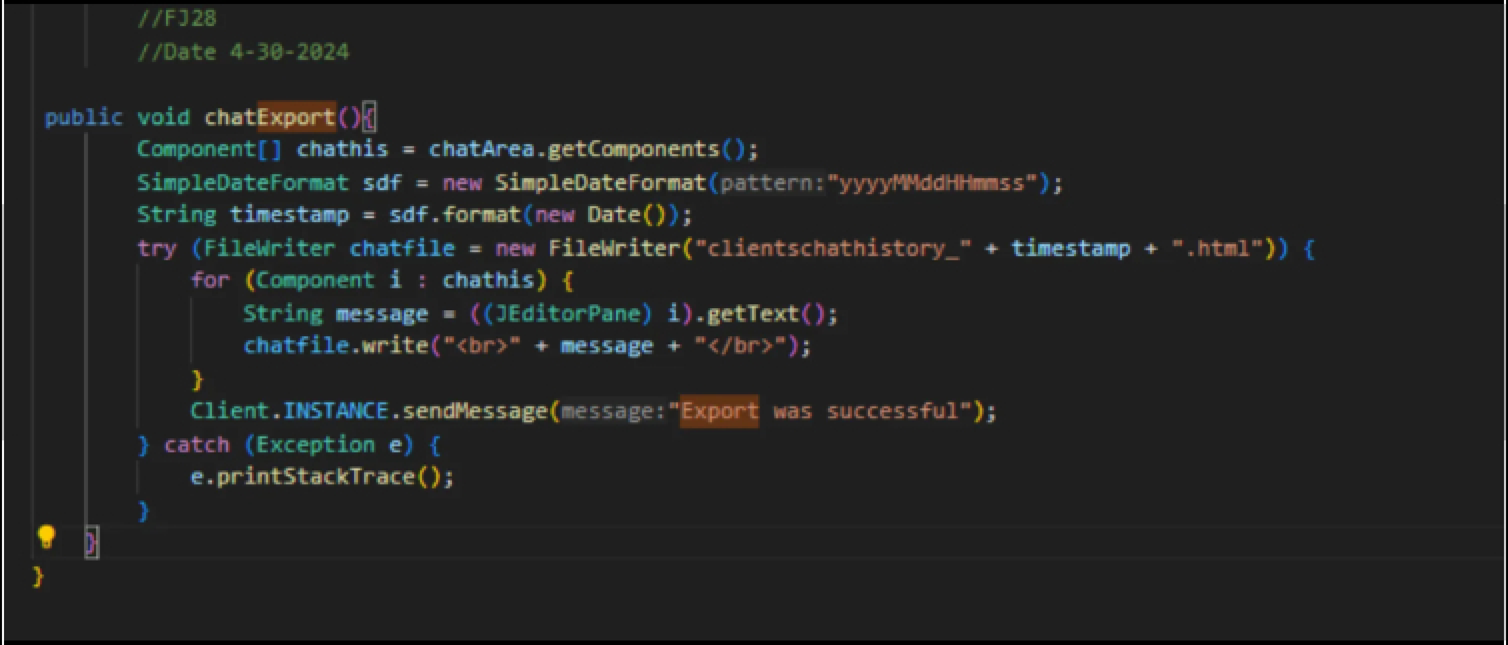
#	Points	Details

■ #1	1	Show the code that gets the messages and writes it to a file (recommended to use a StringBuilder)
■ #2	1	File name should be unique to avoid overwriting (i.e., incorporate timestamp)
■ #3	1	Screenshots should include ucid and date comment
■ #4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
        //FJ28
        //Date 4-30-2024

    public void chatExport(){
        Component[] chathis = chatArea.getComponents();
        SimpleDateFormat sdf = new SimpleDateFormat(pattern:"yyyyMMddHHmmss");
        String timestamp = sdf.format(new Date());
        try (FileWriter chatfile = new FileWriter("clientschathistory_" + timestamp + ".html")) {
            for (Component i : chathis) {
                String message = ((JEditorPane) i).getText();
                chatfile.write("<br>" + message + "</br>");
            }
            Client.INSTANCE.sendMessage(message:"Export was successful");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Export code screenshot

Checklist Items (4)

#1 Show the code that gets the messages and writes it to a file (recommended to use a StringBuilder)

#2 File name should be unique to avoid overwriting (i.e., incorporate timestamp)

#3 Screenshots should include ucid and date comment

#4 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the file

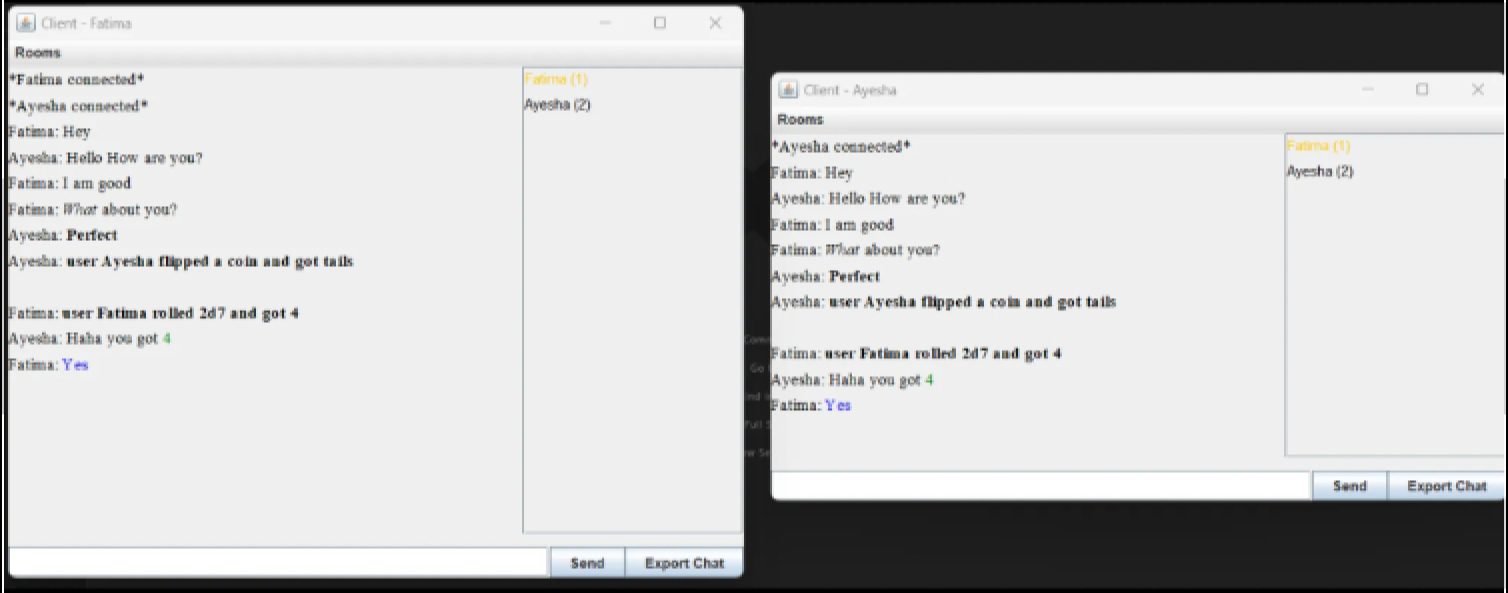
Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show content with variation of messages (i.e., flip, roll, formatting, etc)
☐ #2	1	It should be clear who sent each message
☐ #3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Client - Fatima:

```
*Fatima connected*
*Ayesha connected*
Fatima: Hey
Ayesha: Hello How are you?
Fatima: I am good
Fatima: What about you?
Ayesha: Perfect
Ayesha: user Ayesha flipped a coin and got tails

Fatima: user Fatima rolled 2d7 and got 4
Ayesha: Haha you got 4
Fatima: Yes
```

Fatima (1)
Ayesha (2)

Client - Ayesha:

```
*Ayesha connected*
Fatima: Hey
Ayesha: Hello How are you?
Fatima: I am good
Fatima: What about you?
Ayesha: Perfect
Ayesha: user Ayesha flipped a coin and got tails

Fatima: user Fatima rolled 2d7 and got 4
Ayesha: Haha you got 4
Fatima: Yes
```

Fatima (1)
Ayesha (2)

Screenshot of the file

Checklist Items (3)

#1 Show content with variation of messages (i.e., flip, roll, formatting, etc)

#2 It should be clear who sent each message

#3 Each screenshot should be clearly captioned

^COLLAPSE ^

Task #3 - Points: 1

Text: Explain solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Mention where the messages are stored and how you fetched them
☐ #2	1	Mention how the file is generated and populated

Response:

The Java application exports chat messages from a chat window to an HTML file. When a user clicks the "Export Chat" button, the program saves all the displayed messages to an HTML file. Each export is given a distinctive timestamp in its filename to prevent overwriting previous exports. The HTML files are saved in the program's folder.

● Demonstrate Mute List Persistence (2.25 pts.)

^COLLAPSE ^

● ^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshots of the code

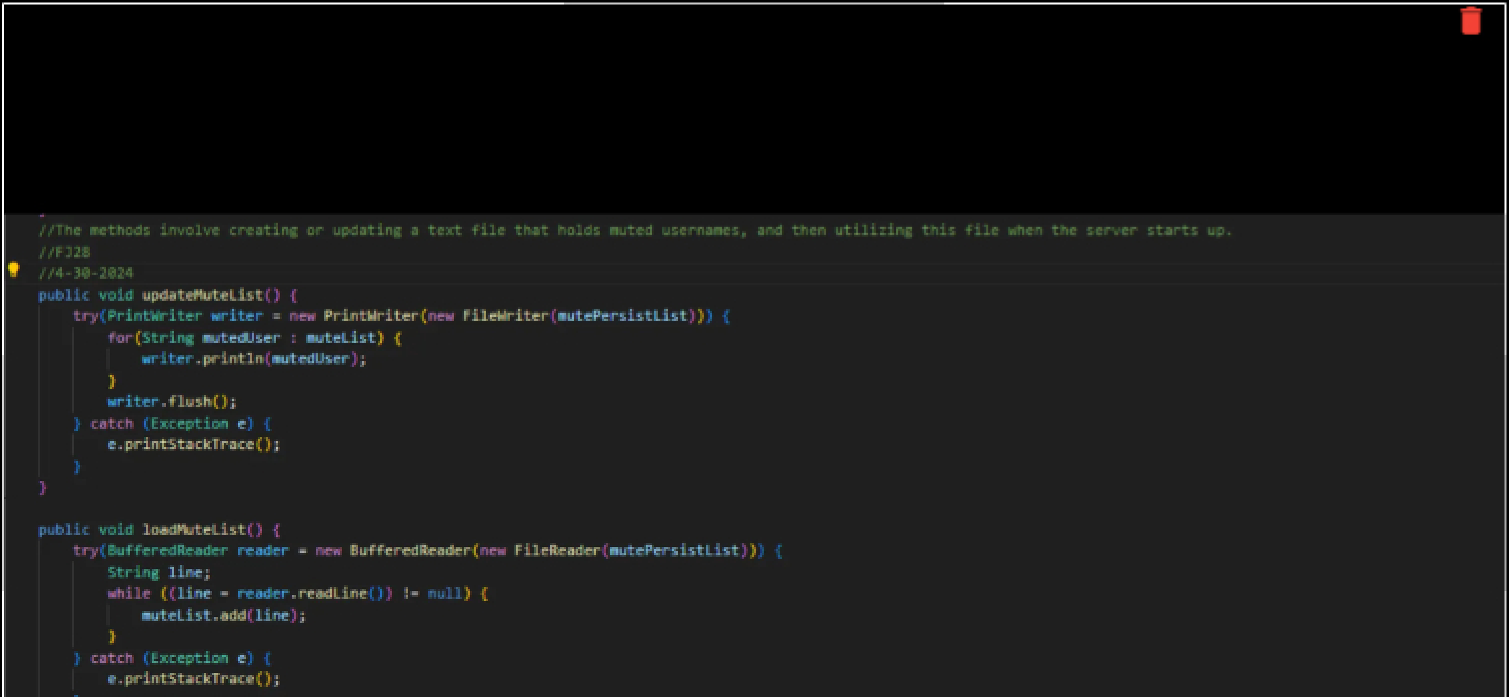
Checklist *The checkboxes are for your own tracking

#	Points	Details
■ #1	1	Show the code that saves the mute list to a file with the name of the user it belongs to
■ #2	1	Show the code that loads the mute list when a ServerThread is connected
■ #3	1	Screenshots should include ucid and date comment
■ #4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```java
//The methods involve creating or updating a text file that holds muted usernames, and then utilizing this file when the server starts up.
//FJ28
//4-30-2024
public void updateMuteList() {
    try(PrintWriter writer = new PrintWriter(new FileWriter(mutePersistList))) {
        for(String mutedUser : muteList) {
            writer.println(mutedUser);
        }
        writer.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void loadMuteList() {
    try(BufferedReader reader = new BufferedReader(new FileReader(mutePersistList))) {
        String line;
        while ((line = reader.readLine()) != null) {
            muteList.add(line);
        }
    } catch (Exception e) {
        e.printStackTrace();
```

```
    }
}
```

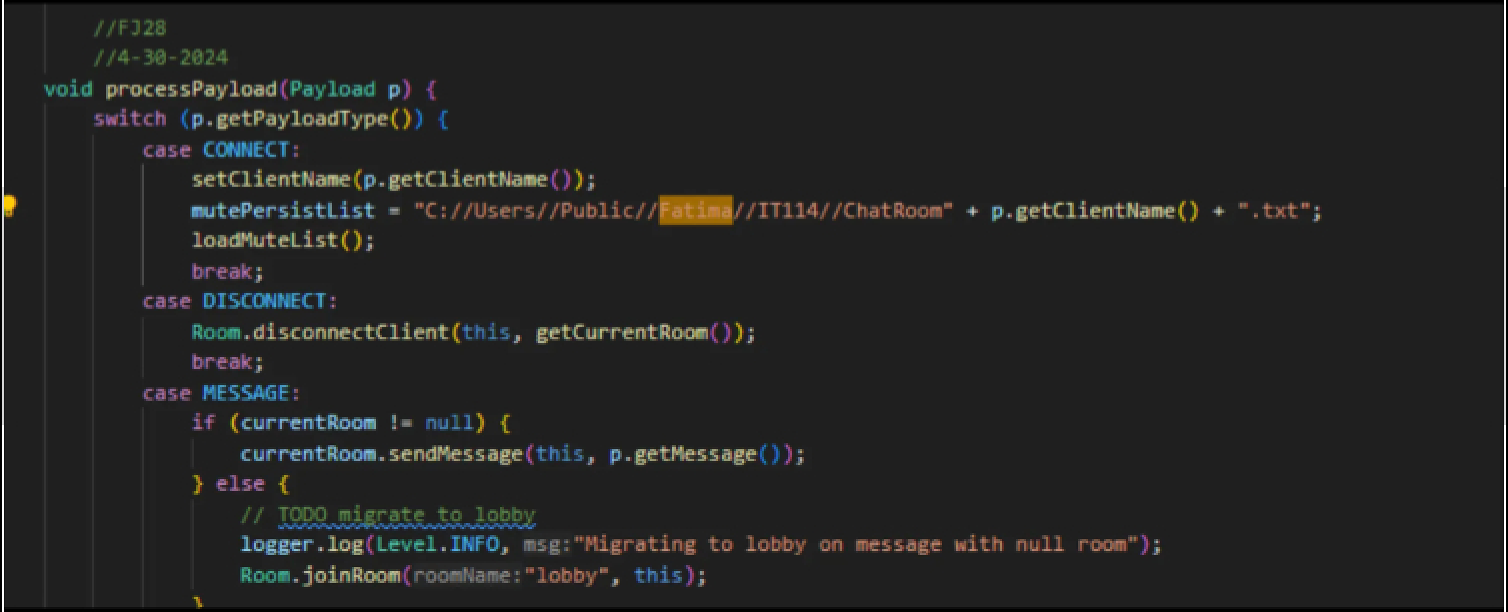
Show the code that saves the mute list to a file with the name of the user it belongs to

Checklist Items (3)

#1 Show the code that saves the mute list to a file with the name of the user it belongs to

#3 Screenshots should include ucid and date comment

#4 Each screenshot should be clearly captioned



```
    //FJ28
    //4-30-2024
void processPayload(Payload p) {
    switch (p.getPayloadType()) {
        case CONNECT:
            setClientName(p.getClientName());
            mutePersistList = "C://Users//Public//Fatima//IT114//ChatRoom" + p.getClientName() + ".txt";
            loadMuteList();
            break;
        case DISCONNECT:
            Room.disconnectClient(this, getCurrentRoom());
            break;
        case MESSAGE:
            if (currentRoom != null) {
                currentRoom.sendMessage(this, p.getMessage());
            } else {
                // TODO migrate to lobby
                logger.log(Level.INFO, msg:"Migrating to lobby on message with null room");
                Room.joinRoom(roomName:"lobby", this);
            }
```

Show the code that loads the mute list when a ServerThread is connected

Checklist Items (3)

#2 Show the code that loads the mute list when a ServerThread is connected

#3 Screenshots should include ucid and date comment

#4 Each screenshot should be clearly captioned

^COLLAPSE ^

Task #2 - Points: 1

Text: Screenshots of the demo

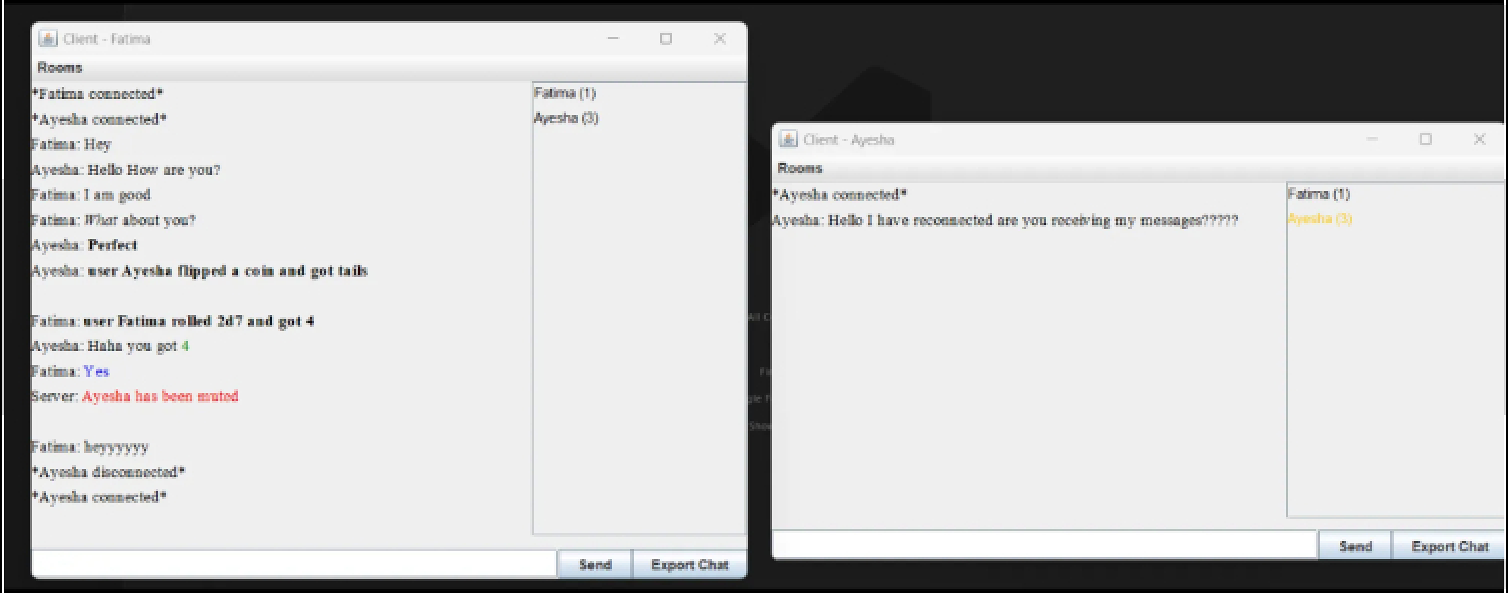
Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)
☐ #2	1	This should also be reflected in the UI per related feature in this milestone
☐ #3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
*Fatima connected*
*Ayesha connected*
Fatima: Hey
Ayesha: Hello How are you?
Fatima: I am good
Fatima: What about you?
Ayesha: Perfect
Ayesha: user Ayesha flipped a coin and got tails

Fatima: user Fatima rolled 2d7 and got 4
Ayesha: Haha you got 4
Fatima: Yes
Server: Ayesha has been muted

Fatima: heyyyyyy
*Ayesha disconnected*
*Ayesha connected*
```

```
*Ayesha connected*
Ayesha: Hello I have reconnected are you receiving my messages?????
```

Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)

Checklist Items (3)

#1 Show a user muting another user, disconnecting, reconnecting, and still having that user muted (same should be possible if the server restarts)

#2 This should also be reflected in the UI per related feature in this milestone

#3 Each screenshot should be clearly captioned

^COLLAPSE ^

Task #3 - Points: 1

Text: Explain solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention how you got the mute list to save and load
#2	1	Discuss the steps to sync the data to the client/ui

Response:

To save the list of muted users, we utilize a method called updateMuteList(). This method writes the name of each muted user to a file using a PrintWriter. When we need to load the list of muted users, we use the loadMuteList() method. This function reads each line from the file using a BufferedReader and adds the muted users to our muteList.

When a client connects to the server (CONNECT case), the server sets the client's name and then creates a file path for that client's mute list based on their name. Subsequently, it loads the mute list specifically for that client using the loadMuteList() method.

Demonstrate Mute/Unmute notification (2.25 pts.)

^COLLAPSE ^

^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshots of the code

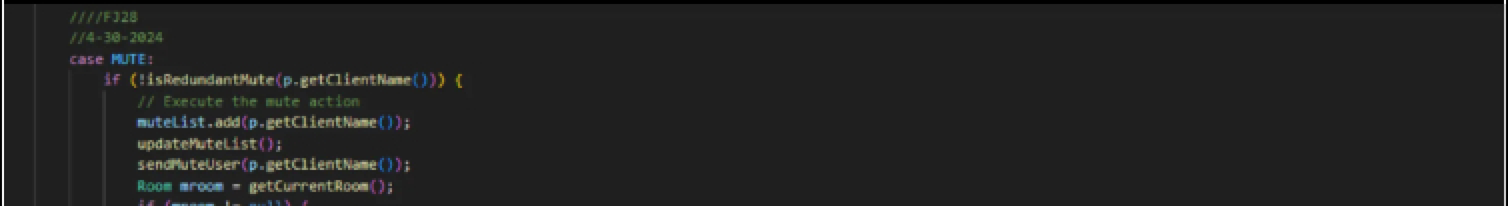
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Show how the message is sent to the target user only if their mute/unmute state changes (i.e., doing mute twice for the same user shouldn't send two mute messages)
#2	1	Screenshots should include ucid and date comment
#3	1	Each screenshot should be clearly captioned

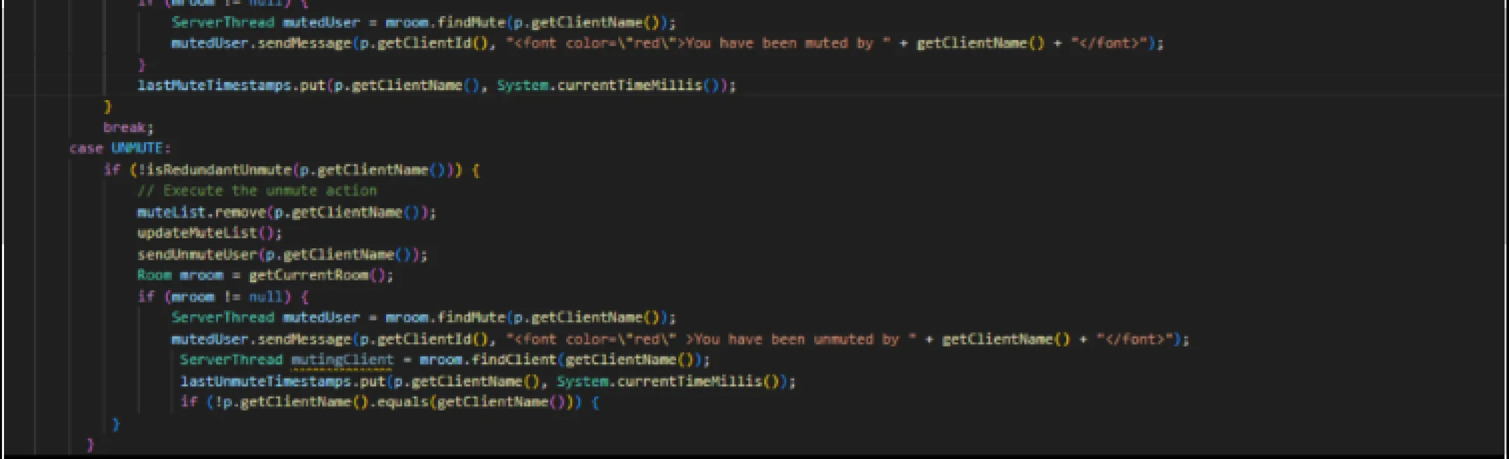
Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
////FJ28
//4-30-2024
case MUTE:
    if (!isRedundantMute(p.getClientName())) {
        // Execute the mute action
        muteList.add(p.getClientName());
        updateMuteList();
        sendMuteUser(p.getClientName());
        Room mroom = getCurrentRoom();
```



```
                if (mroom != null) {
                    ServerThread mutedUser = mroom.findMute(p.getClientName());
                    mutedUser.sendMessage(p.getClientId(), "<font color=\"red\">You have been muted by " + getClientName() + "</font>");
                }
                lastMuteTimestamps.put(p.getClientName(), System.currentTimeMillis());
            }
            break;
        case UNMUTE:
            if (!isRedundantUnmute(p.getClientName())) {
                // Execute the unmute action
                muteList.remove(p.getClientName());
                updateMuteList();
                sendUnmuteUser(p.getClientName());
                Room mroom = getCurrentRoom();
                if (mroom != null) {
                    ServerThread mutedUser = mroom.findMute(p.getClientName());
                    mutedUser.sendMessage(p.getClientId(), "<font color=\"red\" >You have been unmuted by " + getClientName() + "</font>");
                     ServerThread mutingClient = mroom.findClient(getClientName());
                     lastUnmuteTimestamps.put(p.getClientName(), System.currentTimeMillis());
                     if (!p.getClientName().equals(getClientName())) {
             }
         }
```

Message is sent to the target user only if their mute/unmute state changes (i.e., doing mute twice for the same user shouldn't send two mute messages)

Checklist Items (0)

●

^COLLAPSE^

Task #2 - Points: 1

Text: Screenshots of the demo

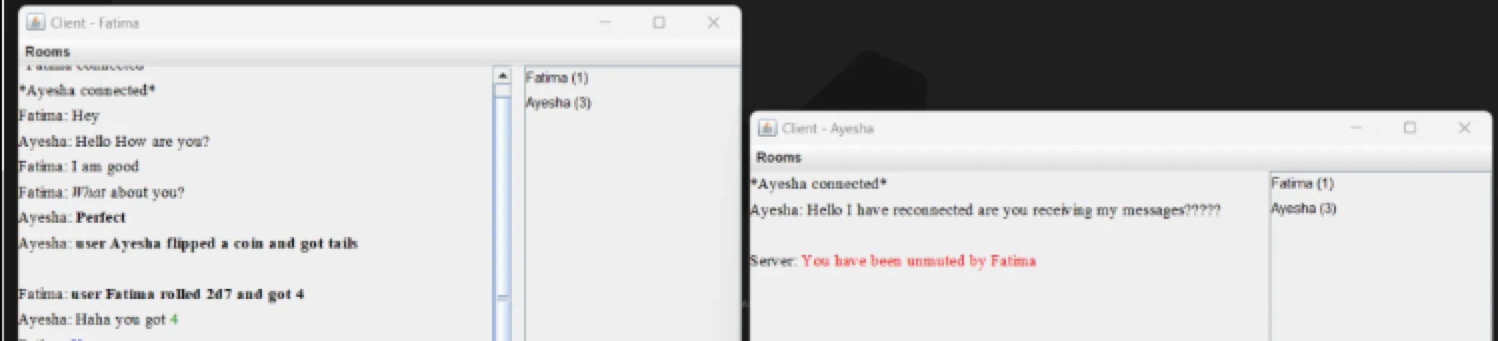
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Show examples of doing /mute twice in succession for the same user only yields one message
#2	1	Show examples of doing /unmute twice in succession for the same user only yields one message
#3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Client - Fatima

Rooms

Ayesha connected
Fatima: Hey
Ayesha: Hello How are you?
Fatima: I am good
Fatima: *What* about you?
Ayesha: **Perfect**
Ayesha: **user Ayesha flipped a coin and got tails**

Fatima: **user Fatima rolled 2d7 and got 4**
Ayesha: Haha you got 4

Fatima (1)
Ayesha (3)

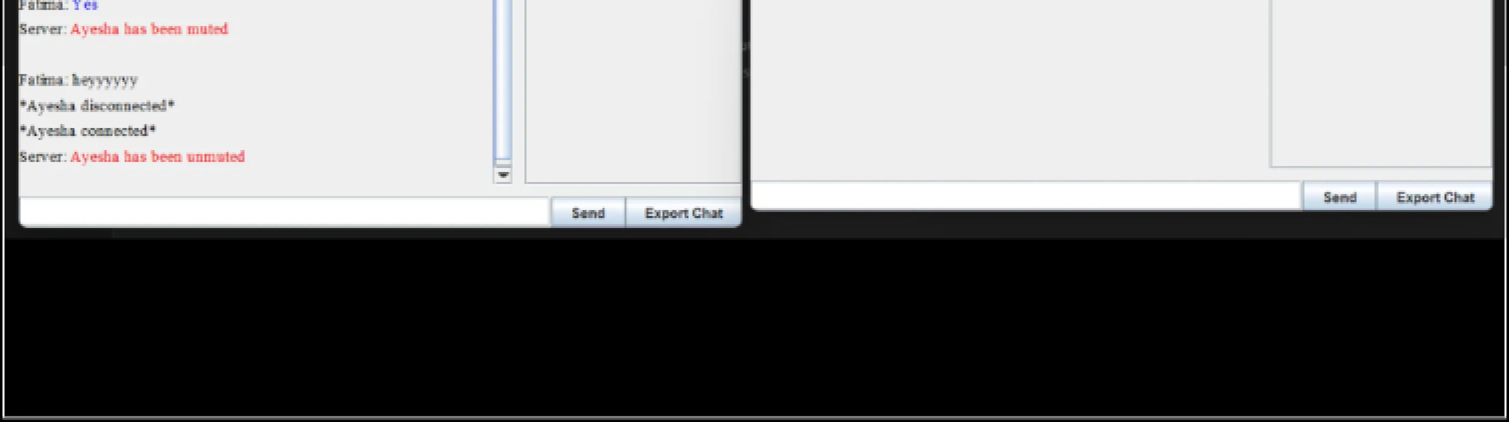
Client - Ayesha

Rooms

Ayesha connected
Ayesha: Hello I have reconnected are you receiving my messages?????

Server: You have been unmuted by Fatima

Fatima (1)
Ayesha (3)


Fatima: Yes
Server: Ayesha has been muted

Fatima: heyyyyyy
Ayesha disconnected
Ayesha connected
Server: Ayesha has been unmuted

The mute and unmute actions were executed twice each, resulting in only one message being displayed.

Checklist Items (3)

#1 Show examples of doing /mute twice in succession for the same user only yields one message

#2 Show examples of doing /unmute twice in succession for the same user only yields one message

#3 Each screenshot should be clearly captioned

●

^COLLAPSE ^

Task #3 - Points: 1

Text: Explain solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
■ #1	1	Mention how you limit the messages in each scenario
■ #2	1	Discuss how you find the correct user to send the message to

Response:

To restrict the number of messages, we rely on timestamps to determine when a mute or unmute action was last executed for a user. If less than 5 seconds have elapsed since the last action, we refrain from sending another message to prevent duplicates.

To locate the correct user, we utilize the username of the target user (retrieved with p.getClientName()). This username is used to identify their connection thread (ServerThread) within the room. This ensures that we send the mute/unmute message only to the intended user.

●

^COLLAPSE ^

Demonstrate user list visual changes (2.25 pts.)

●

^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshots of the code

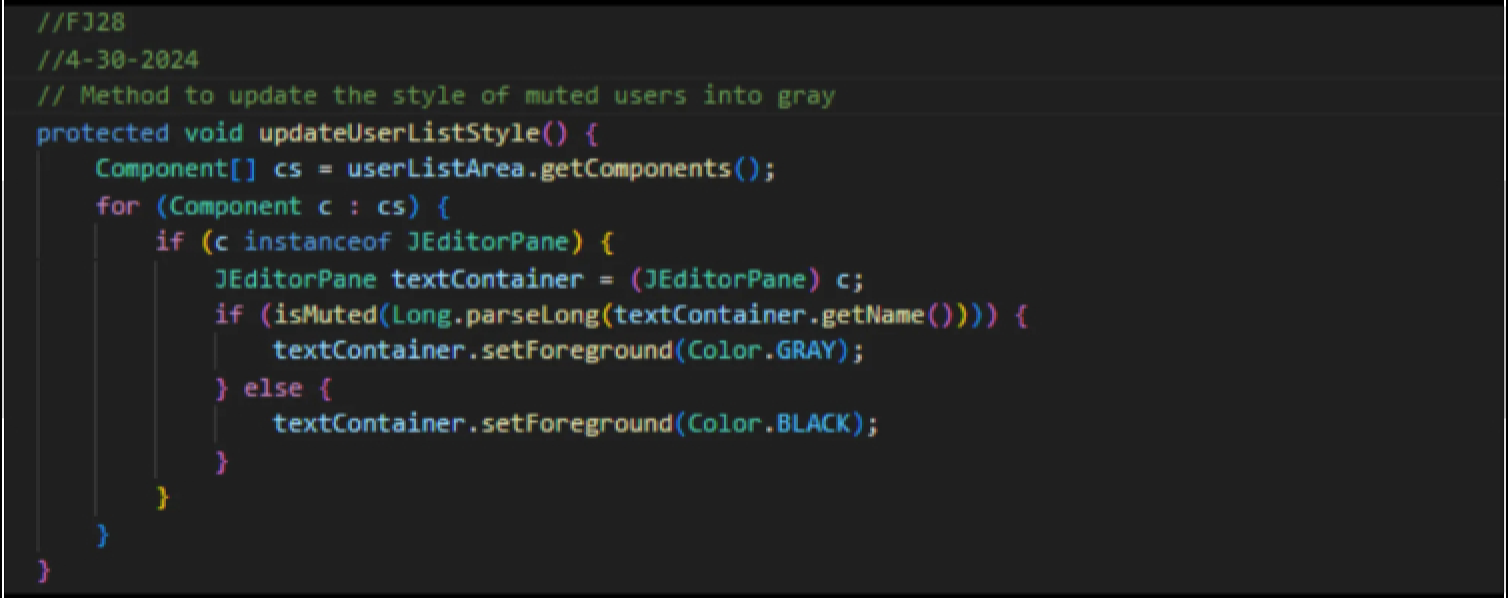
Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show the code related to "graying out" muted users and returning them to normal when unmuted
☐ #2	1	Show the code related to highlighting the user who last sent a message (and unhighlighting the remainder of the list)
☐ #3	1	Screenshots should include ucid and date comment
☐ #4	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```java
//FJ28
//4-30-2024
// Method to update the style of muted users into gray
protected void updateUserListStyle() {
    Component[] cs = userListArea.getComponents();
    for (Component c : cs) {
        if (c instanceof JEditorPane) {
            JEditorPane textContainer = (JEditorPane) c;
            if (isMuted(Long.parseLong(textContainer.getName()))) {
                textContainer.setForeground(Color.GRAY);
            } else {
                textContainer.setForeground(Color.BLACK);
            }
        }
    }
}
```

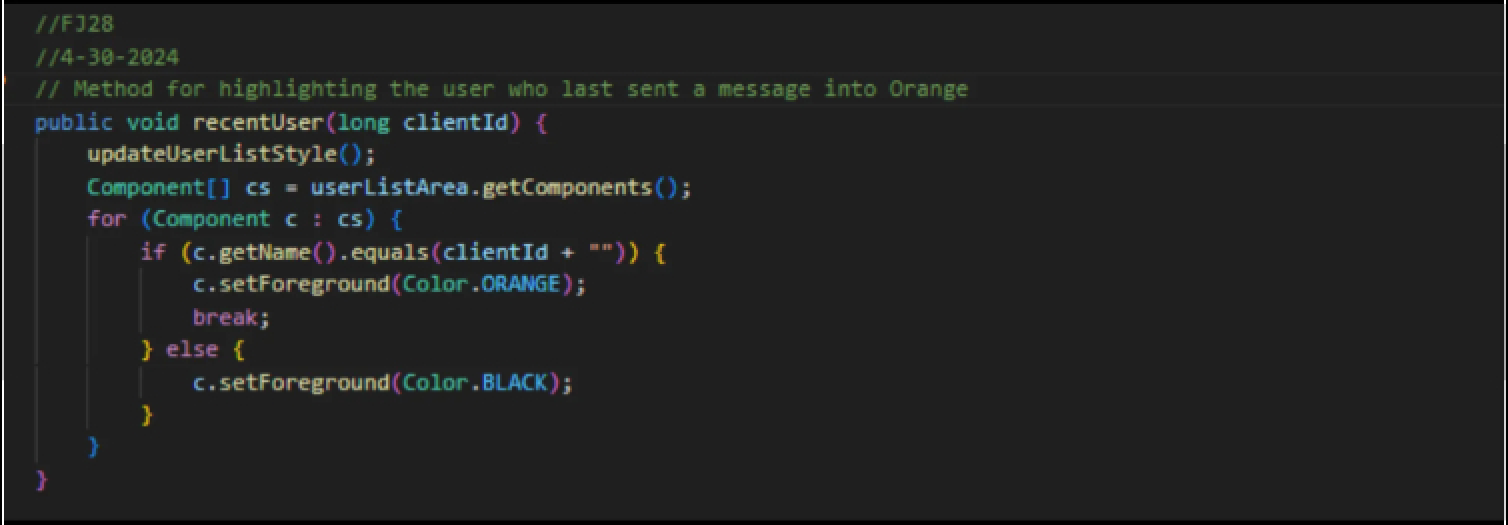
The code related to "graying out" muted users and returning them to normal when unmuted

Checklist Items (3)

#1 Show the code related to "graying out" muted users and returning them to normal when unmuted

#3 Screenshots should include ucid and date comment

#4 Each screenshot should be clearly captioned



```
//FJ28
//4-30-2024
// Method for highlighting the user who last sent a message into Orange
public void recentUser(long clientId) {
    updateUserListStyle();
    Component[] cs = userListArea.getComponents();
    for (Component c : cs) {
        if (c.getName().equals(clientId + "")) {
            c.setForeground(Color.ORANGE);
            break;
        } else {
            c.setForeground(Color.BLACK);
        }
    }
}
```

the code related to highlighting the user who last sent a message (and unhighlighting the remainder of the list)

Checklist Items (3)

#2 Show the code related to highlighting the user who last sent a message (and unhighlighting the remainder of the list)

#3 Screenshots should include ucid and date comment

#4 Each screenshot should be clearly captioned

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the demo

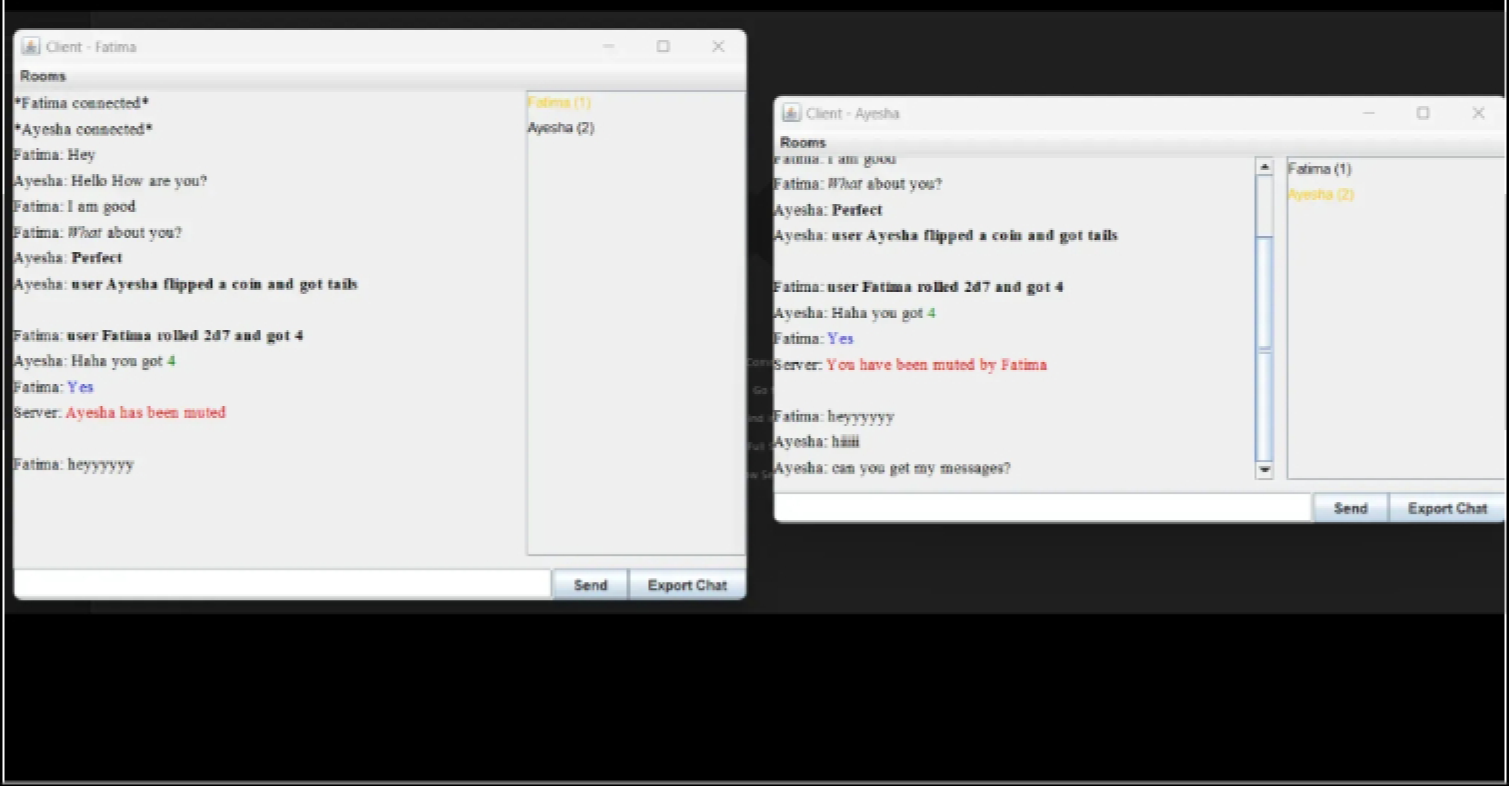
Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show before and after screenshots of the list updating upon mute and unmute
☐ #2	1	Capture variations of "last person to send a message gets highlighted"
☐ #3	1	Each screenshot should be clearly captioned

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
Client - Fatima
Rooms
*Fatima connected*
*Ayesha connected*
Fatima: Hey
Ayesha: Hello How are you?
Fatima: I am good
Fatima: What about you?
Ayesha: Perfect
Ayesha: user Ayesha flipped a coin and got tails

Fatima: user Fatima rolled 2d7 and got 4
Ayesha: Haha you got 4
Fatima: Yes
Server: Ayesha has been muted

Fatima: heyyyyyy
```

```
Client - Ayesha
Rooms
Fatima: I am good
Fatima: What about you?
Ayesha: Perfect
Ayesha: user Ayesha flipped a coin and got tails

Fatima: user Fatima rolled 2d7 and got 4
Ayesha: Haha you got 4
Fatima: Yes
Server: You have been muted by Fatima

Fatima: heyyyyyy
Ayesha: hiiii
Ayesha: can you get my messages?
```

Ayesha is muted and Fatima does not receives her message and she is grayed out in Fatima panel. Also, Ayesha sent the last message.

Checklist Items (3)

#1 Show before and after screenshots of the list updating upon mute and unmute

#2 Capture variations of "last person to send a message gets highlighted"

#3 Each screenshot should be clearly captioned

COLLAPSE

Task #3 - Points: 1

Text: Explain solution

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Mention how you got the mute/unmute effect implemented
#2	1	Mentioned how you got the highlight effect implemented (including unhighlighting the other users)

Response:

A method was developed named updateUserListStyle() to assess each user in the list. If isMuted() indicates that a user is muted, their text color is set to gray (Color.GRAY); otherwise, their text remains black (Color.BLACK).

For the highlighting effect, including unhighlighting, we created a method named recentUser(long clientId) to emphasize the user who most recently sent a message. In this method, we first updated the style using updateUserListStyle() to ensure that everyone's colors are accurate. Next, we evaluated each user in the list to determine if their ID matches the ID of the user who sent the last message (clientId). If a user's ID matches, their text determine if their ID matches the ID of the user who sent the last message (clientId). If a user's ID matches, their text color was changed to red (Color.Orange) for highlighting; otherwise, their text color remained black to indicate they are not highlighted.

● Misc (1 pt.)

^COLLAPSE ^

● ^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/fj29/IT114/pull/4

● ^COLLAPSE ^

Task #2 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

During this project milestone, one key learning was the importance of timestamp management to prevent message duplication. Additionally, implementing methods like updateUserListStyle() and recentUser() taught me about user interface customization and highlighting specific users efficiently. Overall, this assignment provided valuable insights

● ^COLLAPSE ^

Task #3 - Points: 1

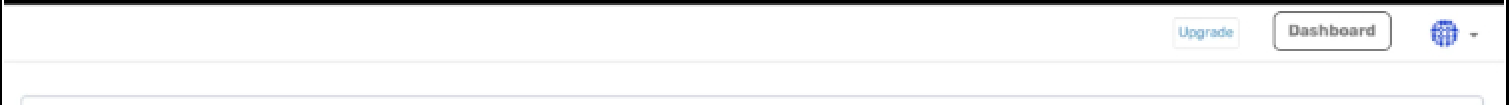
Text: WakaTime Screenshot

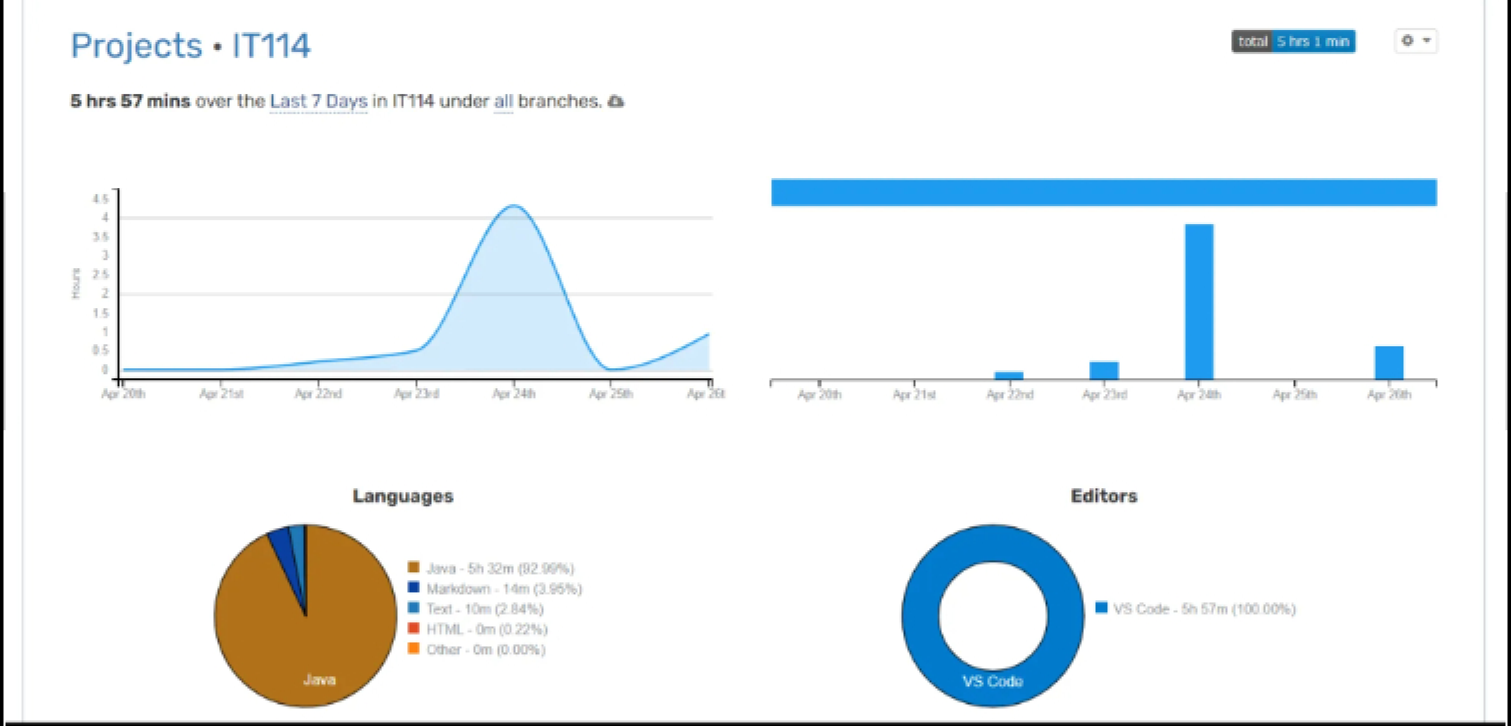
Details:
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved

Task Screenshots:

Gallery Style: Large View

Small Medium Large




Projects • IT114

total 5 hrs 1 min

5 hrs 57 mins over the Last 7 Days in IT114 under all branches.

Languages

- Java - 5h 32m (92.99%)
- Markdown - 14m (3.95%)
- Text - 10m (2.84%)
- HTML - 0m (0.22%)
- Other - 0m (0.00%)

Editors

- VS Code - 5h 57m (100.00%)

WakaTime Screenshot

End of Assignment