

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-milestone-4-chatroom-2024-m24/grade/rev>

Course: IT114-003-F2024

Assignment: [IT114] Milestone 4 Chatroom 2024 M24

Student: Ricardo V. (rev)

Submissions:

Submission Selection

1 Submission [submitted] 12/9/2024 8:13:50 PM

Instructions

^ COLLAPSE ^

- Implement the Milestone 4 features from the project's proposal document:
<https://docs.google.com/document/d/1ONmvEvel97GTFPGfVwwQC96xSsobbSbk56145XizOG4/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

Group



Group: Features

Tasks: 4

Points: 9

^ COLLAPSE ^

Task

100%

Group: Features


Task #1: Client can export chat history of their current session (client-side)

Weight: ~0%

Points: ~0.01

^ 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.

Columns: 4

Sub-Task

100%

Group: Features
Task #1:
Client can
export
chat
history of

Sub-Task

100%

Group: Features
Task #1:
Client can
export
chat
history of

Sub-Task

100%

Group: Features
Task #1:
Client can
export
chat
history of



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



Show the code related to building the export data (where the messages are gathered from, the StringBuilder, and the file generat

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

≡ Task

Response



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



Show the UI interaction that will trigger an export

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

≡ Task

Response

Prompt

Explain where you put it any why

Response:

Response

Prompt

Explain in concise steps how this logically works

Response:

- **Gather Messages:** Retrieve messages from the existing location where they are stored.
 - **Consolidate Messages:** Use a `StringBuilder` to consolidate all messages into a single string.
 - **Generate File:** Create a file and write the consolidated string to it, ensuring the file is saved with a unique name.
- **UI Interaction:** The export button is placed in the chat interface toolbar for easy access.
 - **Reason:** This location is intuitive for users, allowing them to quickly export their chat history without navigating away from the chat window.

End of Task 1

Task



Group: Features

Task #2: Client's Mute List will persist across sessions (server-side)

Weight: ~0%

Points: ~0.01

^ COLLAPSE ^

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.



Columns: 4

Sub-Task



Group: Features
Task #2:
Client's
Mute List
will persist
across

Sub-Task



Group: Features
Task #2:
Client's
Mute List
will persist
across

Sub-Task



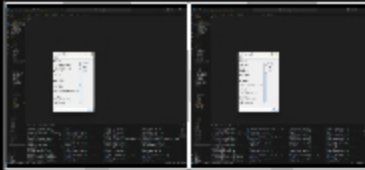
Group: Features
Task #2:
Client's
Mute List
will persist
across

Task

Screenshots

Gallery Style: 2 Columns

4 2 1



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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

Task

Screenshots

Gallery Style: 2 Columns

4 2 1



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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

Task

Response

Prompt

Explain in concise steps how this logically works

Response:

- **Load Mute List:** When a client connects, check if a mute list file exists for them.
- **Handle Missing File:** If no file exists, create an empty mute list.
- **Apply Mute List:** Load the mute list and apply it to the client's session.

Task

Screenshots

Gallery Style: 2 Columns

4 2 1



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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

Task

Response

Prompt

Explain in concise steps how this logically works

Response:

- **Detect Changes:** Monitor the mute list for any changes during the session.
- **Save Changes:** Whenever the mute list changes, save the updated list to the server.
- **File Management:** Ensure the file is named appropriately to associate it with the correct client.

Task

100%

Group: Features

Task #3: Clients will receive a message when they get muted/unmuted by another user

Weight: ~0%

Points: ~0.00

^ COLLAPSE ^

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

Columns: 4

Sub-Task

100%

Group:
Features
Task #3:
Clients will
receive a
message
when they

Sub-Task

100%

Group:
Features
Task #3:
Clients will
receive a
message
when they



Task

Screenshots

Gallery Style: 2 Columns

4 2 1



Show the code that generates examples of the well formatted message only when the mutestate changes (see notes in the details above)

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown



Task



Task

Screenshots

Gallery Style: 2 Columns

4 2 1

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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

being shown

Response

Prompt

Explain in concise steps how this logically works

Response:

- Detect State Change: Monitor for mute/unmute commands.
- Generate Message: When a state change is detected, generate a well-formatted message.
- Send Message: Send the message to the affected client, ensuring only one message is sent per state change.

End of Task 3

Task



Group: Features
Task #4: The user list on the Client-side should update per the status of each user
Weight: ~0%
Points: ~0.01

^ COLLAPSE ^

i Details:

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



Columns: 4

<p>Sub-Task</p> <p>100%</p> <p>Group: Features Task #4: The user list on the Client-side should</p>	<p>Sub-Task</p> <p>100%</p> <p>Group: Features Task #4: The user list on the Client-side should</p>	<p>Sub-Task</p> <p>100%</p> <p>Group: Features Task #4: The user list on the Client-side should</p>	<p>Sub-Task</p> <p>100%</p> <p>Group: Features Task #4: The user list on the Client-side should</p>
---	---	---	---

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Show the UI for Muted users appear grayed out (or similar indication of your choosing) include a few examples showing it updates

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

Task Screenshots

Gallery Style: 2 Columns

4 2 1



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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

Task Response Prompt

Explain in concise steps how this logically works

Response:

- **Receive Status Updates:** The client receives status updates from the server.
- **Update UI:** The UI updates to reflect the status changes, such as graying out muted users.
- **Highlight Last Sender:** The UI highlights the last person to send a message, ensuring real-time updates.

Task Screenshots

Gallery Style: 2 Columns

4 2 1



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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

Task Screenshots

Gallery Style: 2 Columns

4 2 1



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

Caption(s) (required) ✓

Caption Hint:

Describe/highlight what's being shown

Task Response Prompt

Explain in concise steps how this logically works

Response:

- **Client Receives Message:** The client listens for incoming messages from the server. When a new message is received, it triggers an event handler.
- **Update User Status:** The event handler identifies the sender of the message and updates the status

of the user in the client's data model.

- UI Update: The client's UI component responsible for displaying the user list is notified of the change. It highlights the last person to send a message by updating the visual representation (e.g., changing the background color or adding an icon next to the user's name).

End of Task 4

End of Group: Features
Task Status: 4/4

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/#



URL #1

<https://github.com/RicardoVas9991/Rev-IT-114-0036>

URL

<https://github.com/RicardoVas9991/Rev-IT-114-0>

End of Task 1

Task



Group: Misc

Task #2: Talk about any issues or learnings during this assignment

Weight: ~33%

Points: ~0.33

^ COLLAPSE ^

Task Response Prompt

Response:

It was tough putting all the things together, but I pulled it through and now everything is completed. I also did the extra credit too.

End of Task 2

Task



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 Screenshot



WakaTime Screenshot



Extra credit

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment