CS310 Assignment 1

Semester 1, 2025

—

Submission date: Wednesday 11th May, 2025

Group Name: Lofi Beats

SHAMAL PRASAD S11219545

RAVINESH NARAYAN S11197324

ISAIAH NARAYAN S11175115

Table Of Content

Network Architecture And Protocol **2**
Architecture 2
Protocol 2
Communication Logic Diagram **3**
System robustness **6**
Error Handling 6
Reliable Connection 12
Learning Outcome **12**
Appendix **13**
Test Report 13
User Manual 23
Before Running Program 23
Server 23
Client 23
Admin User 24
Tutor User 26
Student User 31
Declaration of Individual Contribution 36

Network Architecture And Protocol

Architecture

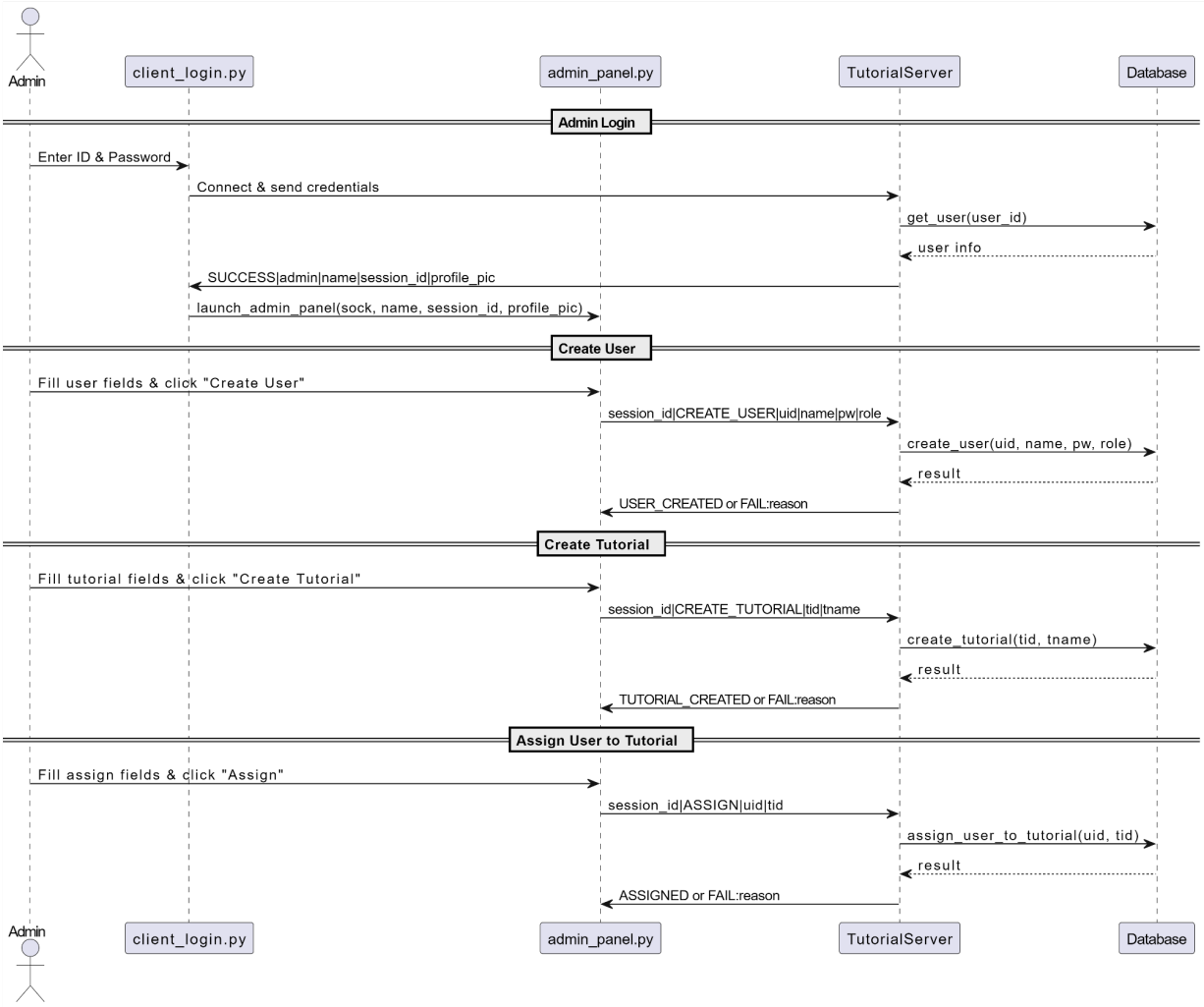
Our program was designed on the client server architecture. This particular architecture enables workload to be distributed between 2 hosts that are connected together through a network. The 2 hosts being client and server, the server functions as a provider that listens to all client requests, processes it and responds to client requests. While on the other end the client requests service from the server. This is a useful system when creating a centralized system where each client has their request processes with integrity. Hence, due to these reasons our team has decided to implement the tutorial management system with the client/server architecture in mind. When a client makes a change in the information on the server, all concerned clients that are connected to the server will have their end devices updated. This ensures that data is processed and delivered to everyone connected to the server, unlike peer-to-peer that does not guarantee information to be propagated to all concerned parties connected in a timely manner. Everytime a person sends a message in the chat, everyone's end device is updated to display the latest message, ensuring total consistency in each user's experience.

Protocol

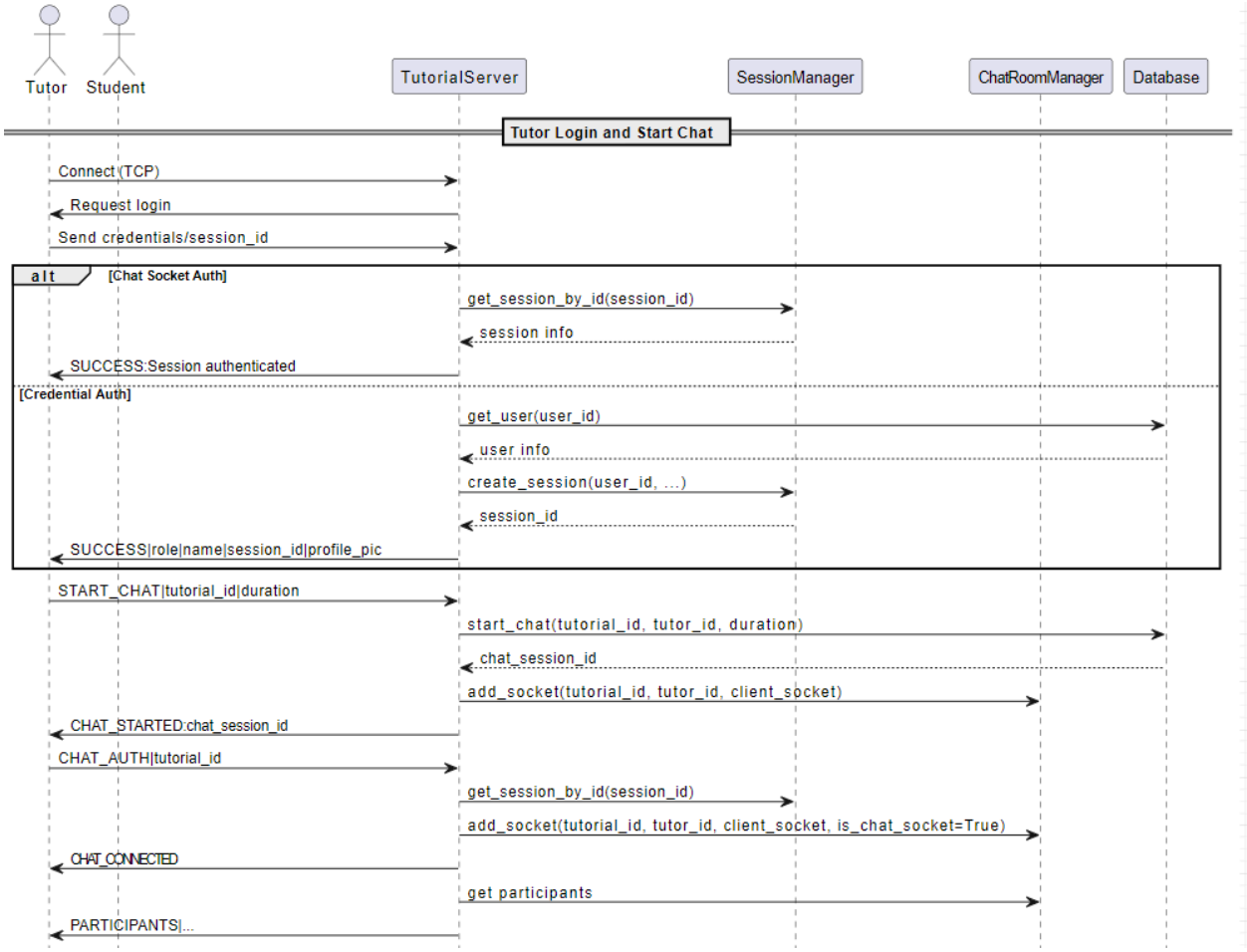
To support the client/server architecture, the tutorial management system employs the Transmission Control Protocol (TCP) to ensure guaranteed data delivery to and from the server and the client and vice versa. Our program uses a 3 way TCP handshake to ensure that the client can indeed connect to the server, and the server sends the acknowledgement back to the client and opens a raw socket connection, which is then acknowledged by the client. The use of TCP sockets allows the client application to connect to the server application and ensure data packet delivery, which is paramount to our Tutorial Management System which relies on guaranteed data packet delivery to ensure consistent message delivery when connected to a tutorial chatroom. This will ensure that every connected client (student and Tutor) in the tutorial chatroom will receive every message. UDP was not suitable as it does not guarantee data packet delivery and does not support the objectives of the Tutorial Management System.

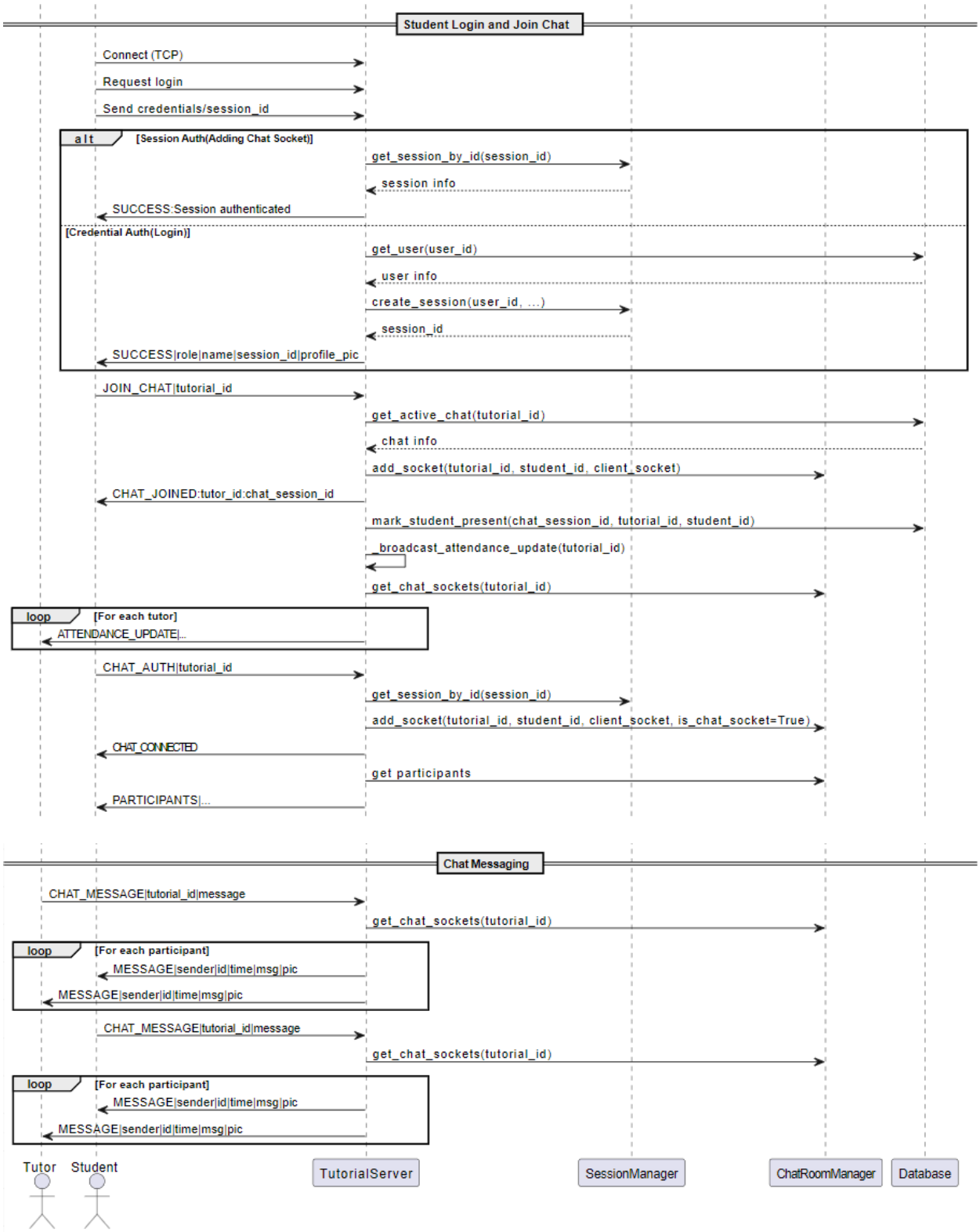
Communication Logic Diagram

Admin User



Tutor & Student User




Student Login and Join Chat
Connect (TCP)
Request login
Send credentials/session_id
alt
[Session Auth(Adding Chat Socket)]
get_session_by_id(session_id)
session info
SUCCESS:Session authenticated
[Credential Auth(Login)]
get_user(user_id)
user info
create_session(user_id, ...)
session_id
SUCCESS|role|name|session_id|profile_pic
JOIN_CHAT|tutorial_id
get_active_chat(tutorial_id)
chat info
add_socket(tutorial_id, student_id, client_socket)
CHAT_JOINED:tutor_id:chat_session_id
mark_student_present(chat_session_id, tutorial_id, student_id)
_broadcast_attendance_update(tutorial_id)
get_chat_sockets(tutorial_id)
loop
[For each tutor]
ATTENDANCE_UPDATE|...
CHAT_AUTH|tutorial_id
get_session_by_id(session_id)
add_socket(tutorial_id, student_id, client_socket, is_chat_socket=True)
CHAT_CONNECTED
get participants
PARTICIPANTS|...
Chat Messaging
CHAT_MESSAGE|tutorial_id|message
get_chat_sockets(tutorial_id)
loop
[For each participant]
MESSAGE|sender|id|time|msg|pic
MESSAGE|sender|id|time|msg|pic
CHAT_MESSAGE|tutorial_id|message
get_chat_sockets(tutorial_id)
loop
[For each participant]
MESSAGE|sender|id|time|msg|pic
MESSAGE|sender|id|time|msg|pic
Tutor
Student
TutorialServer
SessionManager
ChatRoomManager
Database

System robustness

Error Handling

- When any of the 3 clients attempts to login to their respective dashboards, any form of incorrect credential entry will be met with an error message saying invalid Login. If any textbox is left empty, then there will be another error message saying Login failed to signify to the user that they have not entered the correct login details. Below you will see a snippet of the function that prints the error message on the login screen, if it fails.

```
try:
            if not bcrypt.checkpw(password.encode(), user['password_hash'].encode()):
                logger.warning(f"Login failed - Incorrect password for user: {user_id}")
                send_data(client_socket, "FAIL:Incorrect password")
                close_socket(client_socket)
                return
        except Exception as e:
            logger.error(f"Password verification error: {str(e)}")
            send_data(client_socket, "FAIL:Authentication error")
            close_socket(client_socket)
            return
```

```
if response.startswith("SUCCESS"):
            print(response)
                        parts = response.split("|")
            if len(parts) < 5:
                status_label.configure(text="⚠️ Invalid server response", text_color="red")
                close_socket(s, cleanup=True)
                return

            role, name, session_id, profile_pic_url = parts[1:5]

            status_label.configure(text=f"✅ Logged in as {name} ({role})", text_color="green")

            app.destroy()

            if role == "admin":
                from admin_panel import launch_admin_panel
                launch_admin_panel(s, name, session_id,profile_pic_url)

            elif role == "tutor":
                from tutor_panel import launch_tutor_panel
                launch_tutor_panel(s, name, user_id, session_id,profile_pic_url)

            elif role == "student":
                from student_panel import launch_student_panel
                launch_student_panel(s, user_id, name, session_id,profile_pic_url)

        else:
            status_label.configure(text="❌ Login failed", text_color="red")
            close_socket(s, cleanup=True)
```

- When the admin user tries to create a new tutor or student, they must fill all the textboxes before they can create the user, if not done so, the program will give an error message saying the input fields have not been filled. The same is also true for when the admin tries to create a new tutorial and/or assign the student or tutor to a tutorial. Any empty input fields will produce an error message.

```
try:
            response = self.send_and_get_response("CREATE_USER", uid, name, pw, role)

            if response == "USER_CREATED":
                self.entry_user_id.delete(0, "end")
                self.entry_user_name.delete(0, "end")
                self.entry_user_pw.delete(0, "end")
                self.user_role_option.set("student")
                self.status_label.configure(text="User created successfully!", text_color="green")
            elif response.startswith("FAIL:"):
                error_msg = response.split("FAIL:", 1)[1]
                self.status_label.configure(text=f"Error: {error_msg}", text_color="red")
            else:
                self.status_label.configure(text=f"Error: {response}", text_color="red")

        except Exception as e:
            self.status_label.configure(text=f"Error: {str(e)}", text_color="red")
```

```
try:
            response = self.send_and_get_response("CREATE_TUTORIAL", tid, tname)

            if response == "TUTORIAL_CREATED":
                self.entry_tutorial_id.delete(0, "end")
                self.entry_tutorial_name.delete(0, "end")
                self.status_label.configure(text="Tutorial created successfully!", text_color="green")
            elif response.startswith("FAIL:"):
                error_msg = response.split("FAIL:", 1)[1]
                self.status_label.configure(text=f"Error: {error_msg}", text_color="red")
            else:
                self.status_label.configure(text=f"Error: {response}", text_color="red")

        except Exception as e:
            self.status_label.configure(text=f"Error: {str(e)}", text_color="red")
```

```
try:
            response = self.send_and_get_response("ASSIGN", uid, tid)

            if response == "ASSIGNED":
                self.entry_assign_user.delete(0, "end")
                self.entry_assign_tutorial.delete(0, "end")
                self.status_label.configure(text="User assigned successfully!", text_color="green")
            elif response.startswith("FAIL:"):
                error_msg = response.split("FAIL:", 1)[1]
                self.status_label.configure(text=f"Error: {error_msg}", text_color="red")
            else:
                self.status_label.configure(text=f"Error: {response}", text_color="red")

        except Exception as e:
            self.status_label.configure(text=f"Error: {str(e)}", text_color="red")
```

- If the admin tries to create a new user or tutorial when it already exists, then another error message will appear saying that user or tutorial already exists. Additionally, if the admin tries to assign a user to a tutorial that they have previously already been assigned to, another error message will appear saying the user has already been assigned to this tutorial.

```python
def _handle_create_user(self, client_socket, parts):
        """Handle CREATE_USER admin command."""
        if len(parts) != 6:
            send_data(client_socket, "FAIL:Invalid command format")
            return

        new_id = parts[2]
        name = parts[3]
        passwd = parts[4]
        role = parts[5]

        logger.info(f"Creating new user: {new_id} ({role})")

        if get_user(new_id):
            send_data(client_socket, "FAIL:User ID already exists")
        else:
            create_user(new_id, name, passwd, role)
            send_data(client_socket, "USER_CREATED")
```

```python
def _handle_create_tutorial(self, client_socket, parts):
        """Handle CREATE_TUTORIAL admin command."""
        if len(parts) != 4:
            send_data(client_socket, "FAIL:Invalid command format")
            return

        tut_id = parts[2]
        tut_name = parts[3]

        if get_tutorial(tut_id):
            send_data(client_socket, "FAIL:Tutorial ID already exists")
        else:
            create_tutorial(tut_id, tut_name)
            send_data(client_socket, "TUTORIAL_CREATED")
```

```python
def _handle_assign(self, client_socket, parts):
        """Handle ASSIGN admin command."""
        if len(parts) != 4:
            send_data(client_socket, "FAIL:Invalid command format")
            return

        uid = parts[2]
        tid = parts[3]

        role = get_user_role(uid)
        if not role:
            send_data(client_socket, "FAIL:User not found")
            return

        if role == "tutor" and is_tutor_already_assigned(tid):
            send_data(client_socket, "FAIL:Tutor already assigned to this tutorial")
            return

        if assign_user_to_tutorial(uid, tid):
            send_data(client_socket, "ASSIGNED")
        else:
            send_data(client_socket, "FAIL:Database error")
```

- When a tutor wants to start a tutorial chatroom, they will be prompted to enter the number of minutes they wish the chat to last for. The number of minutes cannot be below 5 mins and the textbox cannot be left empty when the tutor tries to start the room. Both these scenarios will produce their respective error messages.

```python
duration = simpledialog.askinteger("Chat Duration",f"Enter chat duration in minutes for {tutorial_name}:",
minvalue=5, maxvalue=120)
            if duration is None:
                return

            response = self.send_and_get_response(f"START_CHAT|{tutorial_id}|{duration}")
            if response.startswith("CHAT_STARTED:"):
                chat_session_id = response.split(":")[1]
                self.redirect_to_chat_room(tutorial_id, duration, chat_session_id)
            else:
                self.status_label.configure(text=response, text_color="red")
        except Exception as e:
            messagebox.showerror("Error", str(e))
```

- When a student tries to join a tutorial chat room that is yet to be activated by the tutor, they will receive an error message saying the tutorial chat is not active.

```
chat = get_active_chat(tutorial_id)
        if not chat:
            send_data(client_socket, "FAIL:No active chat for this tutorial")
            return
```

```
response = self.send_and_get_response("JOIN_CHAT", tutorial_id)
        if response.startswith("ERROR:"):
            messagebox.showerror("Error", response)
            return
```

Reliable Connection

- The program uses 2 sets of raw sockets. The first raw socket is used for connecting the client to the server while the second raw socket is used for connecting the user to the tutorial chat room. The reason our system has been implemented in such a manner is to separate the chatroom traffic from the normal socket. Having both information being sent through the same raw socket would have hindered performance drastically. This strategy is similar to that of the File Transfer Protocol.

Learning Outcome

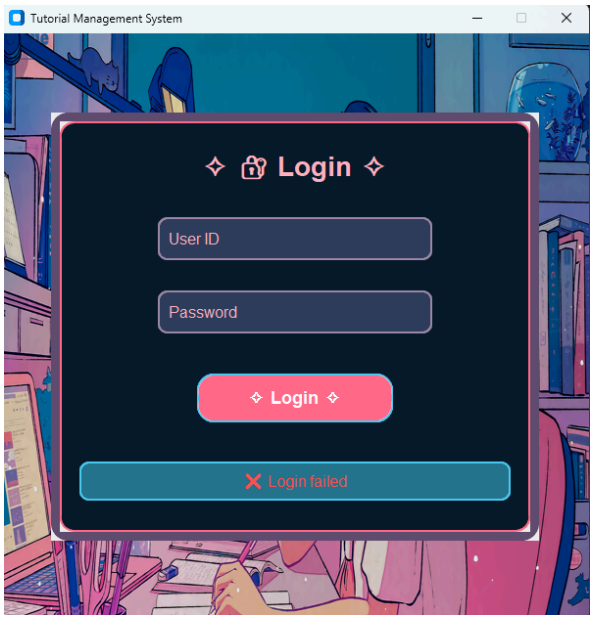
After having developed the Tutorial Management System almost twice, first with using the inbuilt socket python library and second without any help from the inbuilt socket python library, my team and I have come to discover the intricacies of what truly goes on “under the hood” of a normal TCP socket. We have designed and implemented our own socket and socket creation, operation and termination functionalities. We have also introduced other functions to increase the performance of our raw sockets. Moreover, we have not also been well acquainted with python and its environment and will use this unique programming language in future courses and the work field. The significance of the server/client and peer-to-peer architecture was made clear to us and their use cases. Additionally, the importance of TCP network protocol helped us construct our raw socket.

Appendix

Test Report

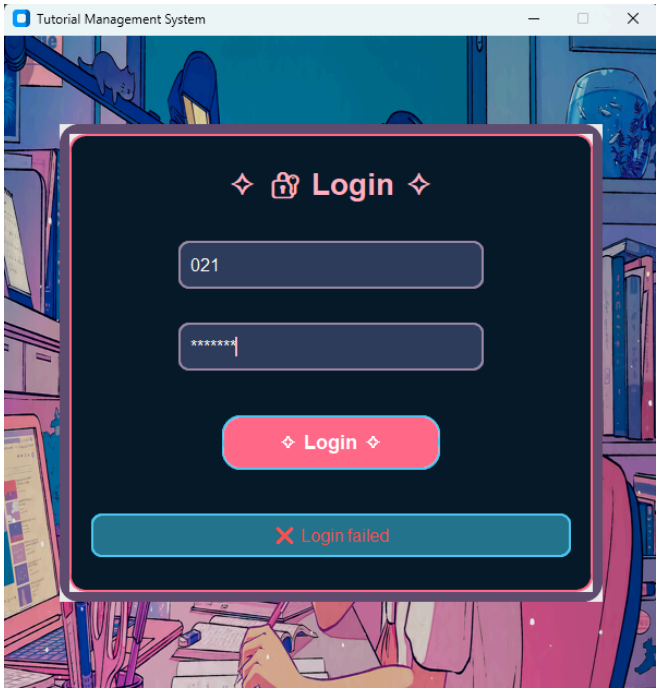
Test Case 1: Login Form Entered Inappropriately

There will be an error saying Login Failed



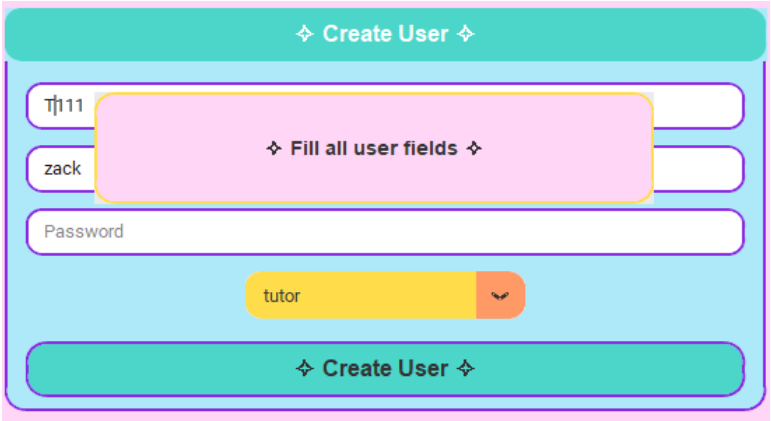
Test Case 2: Wrong Login Credentials

There will be an error saying Login Failed



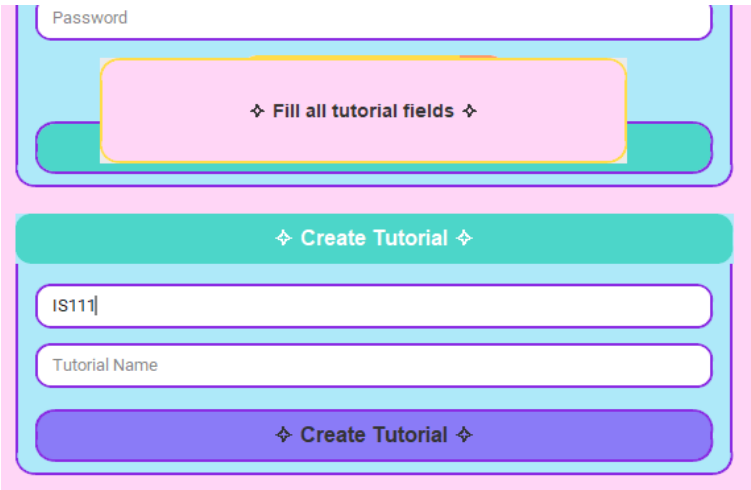
Test Case 3: Admin - User Creation Fields Entered Inappropriately

If all input fields are not filled, you will receive an error as shown below.



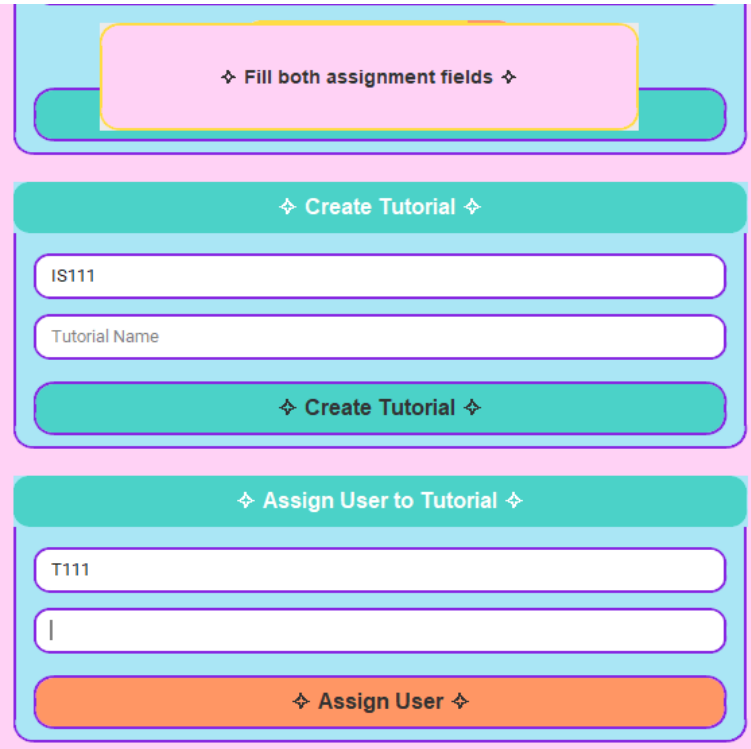
Test Case 4: Admin - Tutorial Creation Fields Entered Inappropriately

If all input fields are not filled, you will receive an error as shown below.



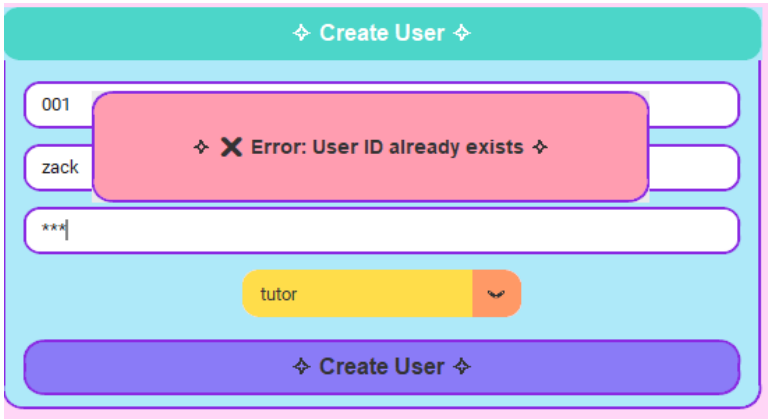
Test Case 5: Admin - User To Tutorial Creation Fields Entered Inappropriately

If all input fields are not filled, you will receive an error as shown below.



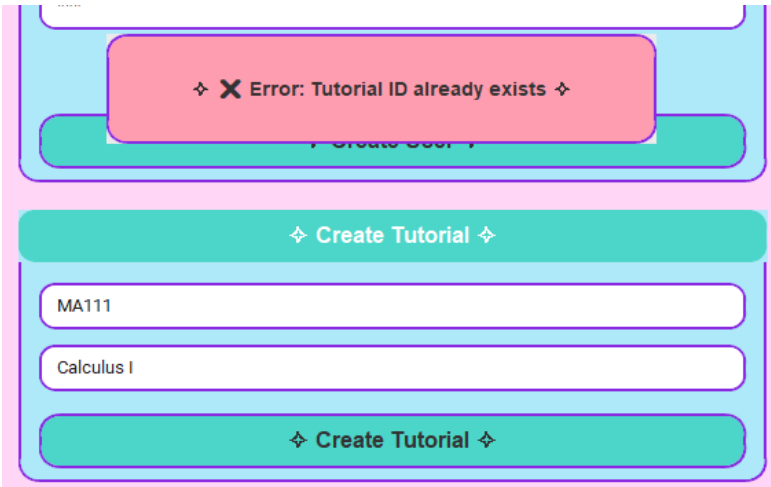
Test Case 6: Admin - User Already Exist

If a user already exists when trying to create the same user, either student or tutor, you will receive an error as shown below.



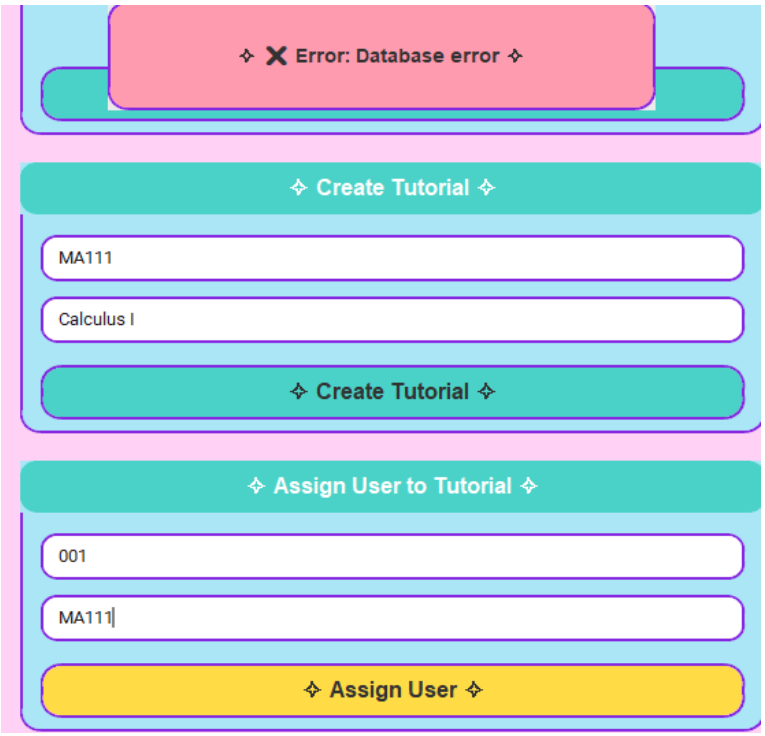
Test Case 7: Admin - Tutorial Already Exist

If a tutorial already exists when trying to create the same tutorial you will receive an error as shown below.



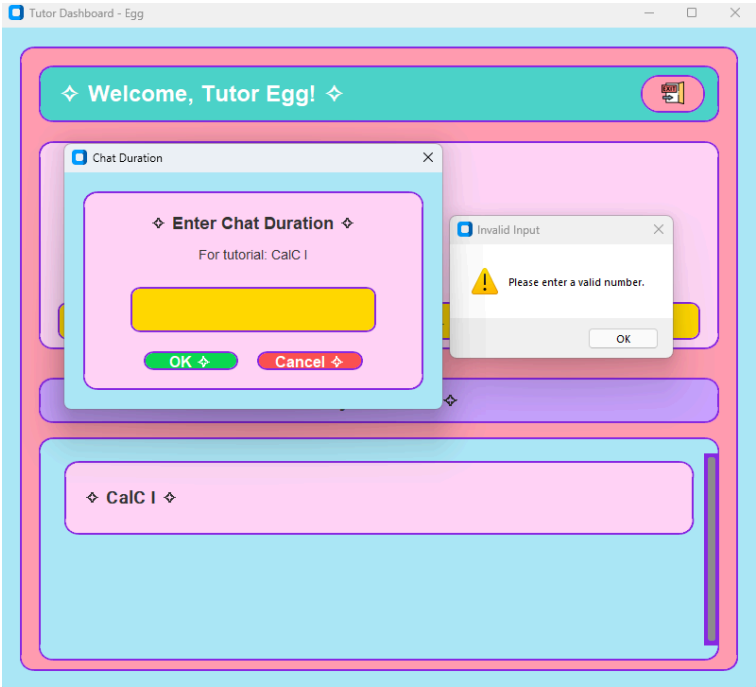
Test Case 8: Admin - User Already Assigned to Tutorial

If a user is already allocated to a tutorial when trying to allocate them, either student or tutor, you will receive an error as shown below.



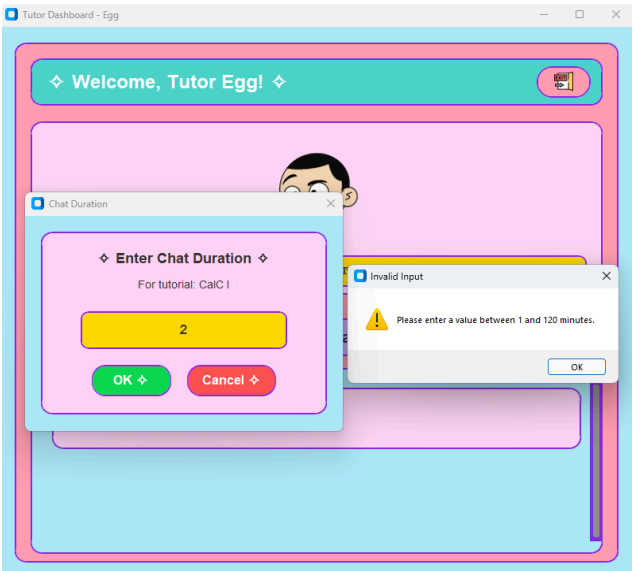
Test Case 9: Tutor Entering Tutorial Duration Fields Inappropriately

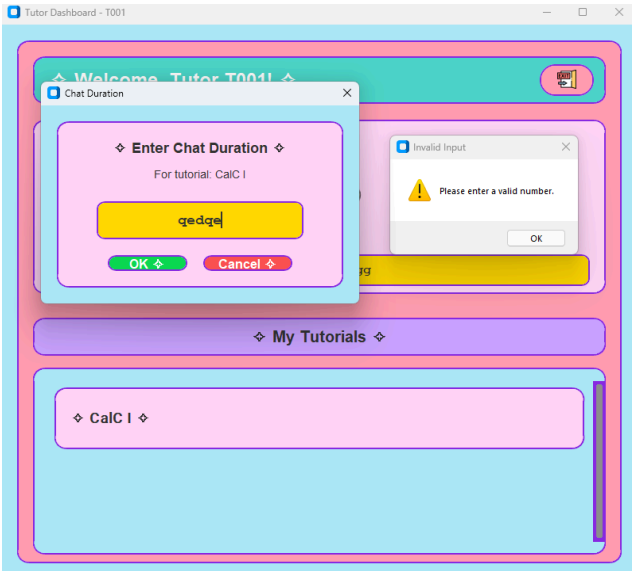
If the tutor does not enter any value or a non integer value as the duration, an error similar to the one below will appear.



Test Case 10: Tutor Entering Tutorial Duration Below Minimum Or Above Maximum Allowed Time

If the tutor tries to set the duration of the chat below 5 minutes or above 120 minutes, the error shown below will be produced.



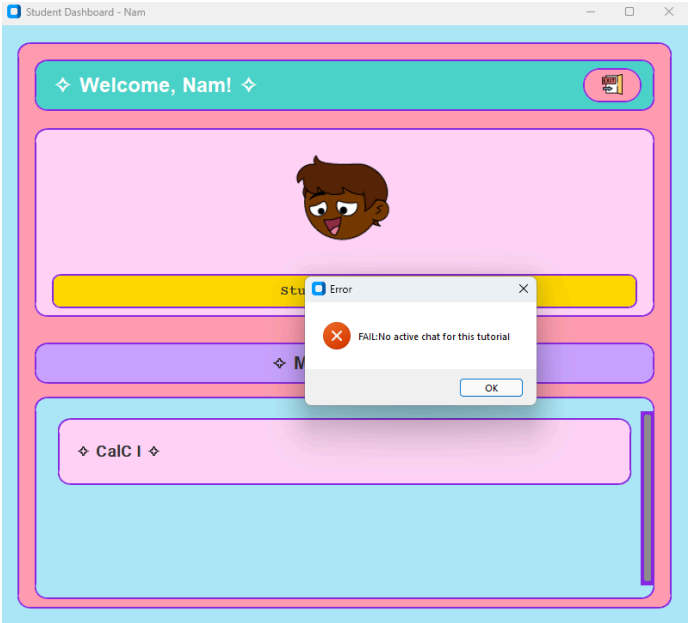


Test Case 11: Tutor Exiting Active Chat and Reentering After Disconnection

A tutor can disconnect, but the chat will still be active, he/she can join back into the chat. When trying to rejoin there will be a prompt saying that chat is still active and you like to join back.

Test Case 12: Students Trying to Enter A Inactive Chat

If a student tries to join a chat that the tutor has not created, then the error shown below will appear.



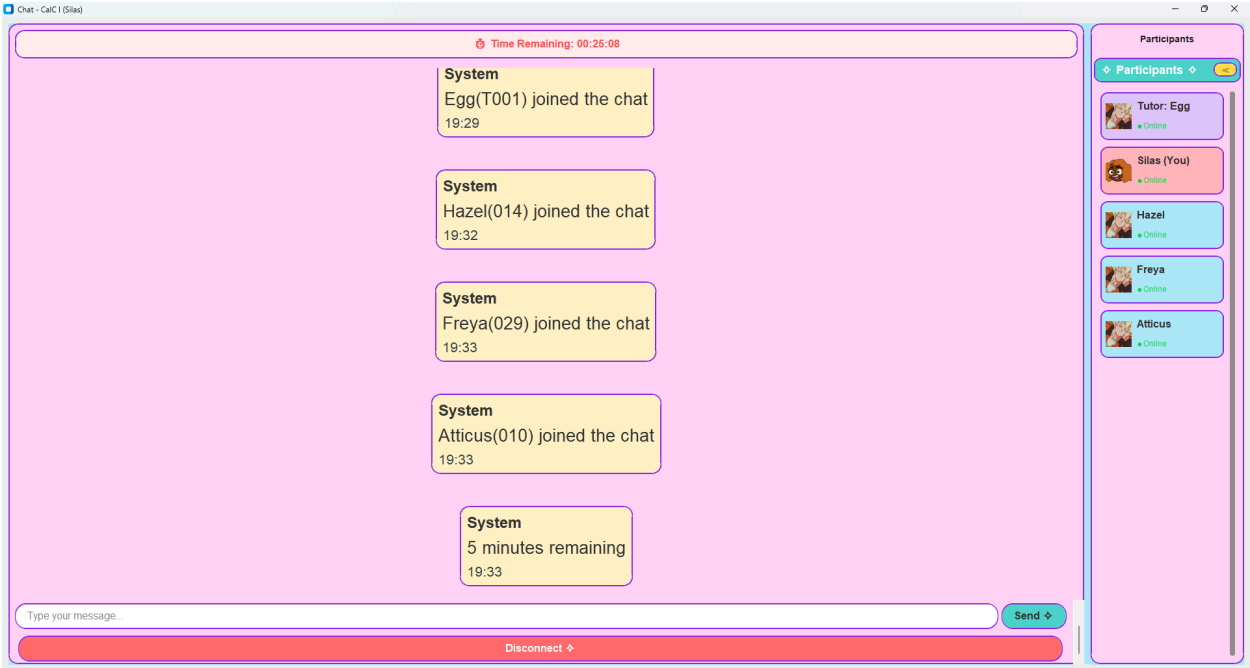
Test Case 13: Having Multiple Users in Active Tutorial

Our program allows many students to connect to the chat and message each other. You can even see the list of students or participants connected to the chat in the participant sidebar.

Test Case 14: Dynamic List of Users in Tutorial Room

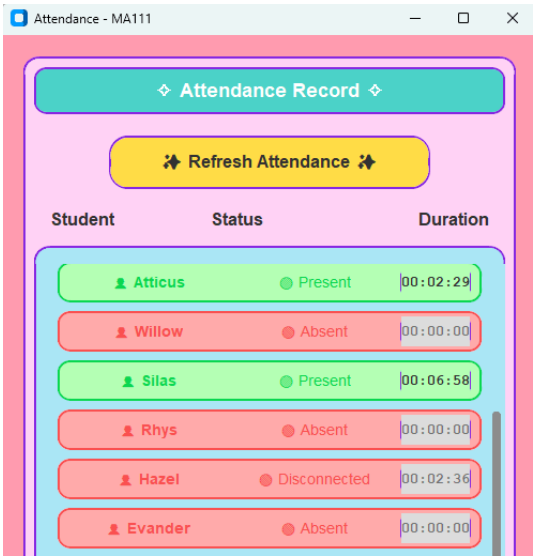
As students join and leave the room the attendance screen and the participant screen will dynamically change to show in real time the currently connected users of the chat.

Currently 5 users are in chat.



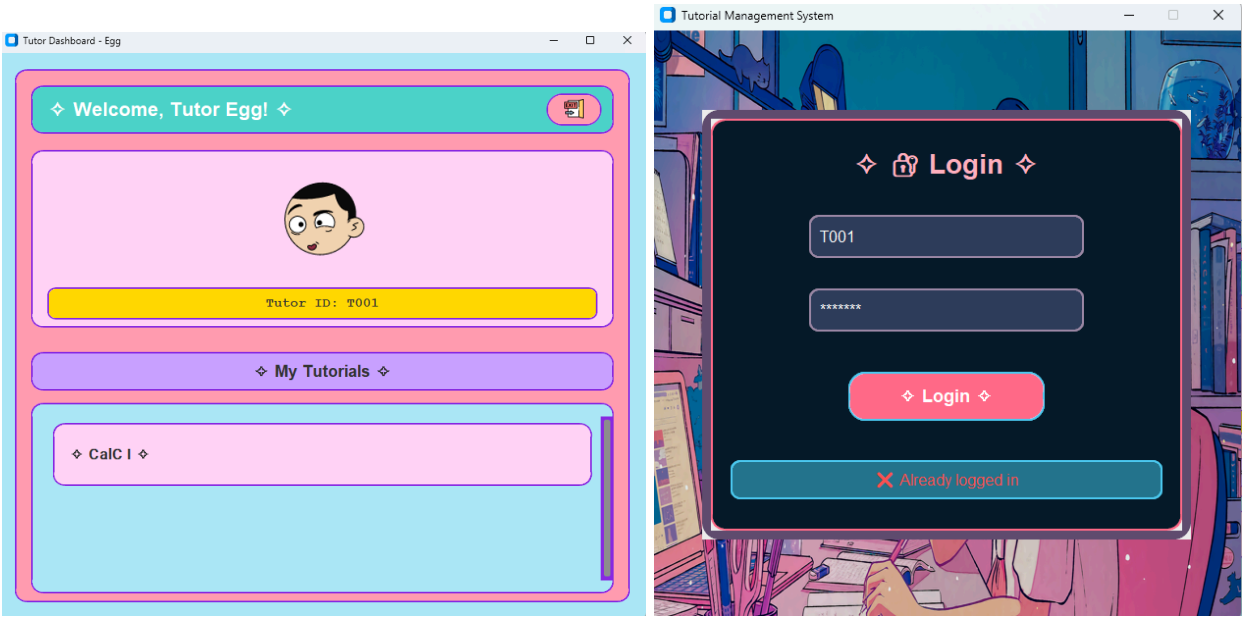
1 person leaves chat

The attendance reflects that the student has disconnected.



Test Case 15: Duplicate Login

If a user, who is already logged in, tries to login again, there will be an error message saying that user has already logged in.



User Manual

Before Running Program

- Ensure Database is active before running the server.
- Install the following packages for the through the terminal code to compile:
    - pip install customtkinter
    - pip install Pillow
    - pip install requests

Server

To run the server, first ensure that the MySQL database contains the necessary database that is running through XAMPP or any other relevant Software. After which you can run the server2.py file to start the server.

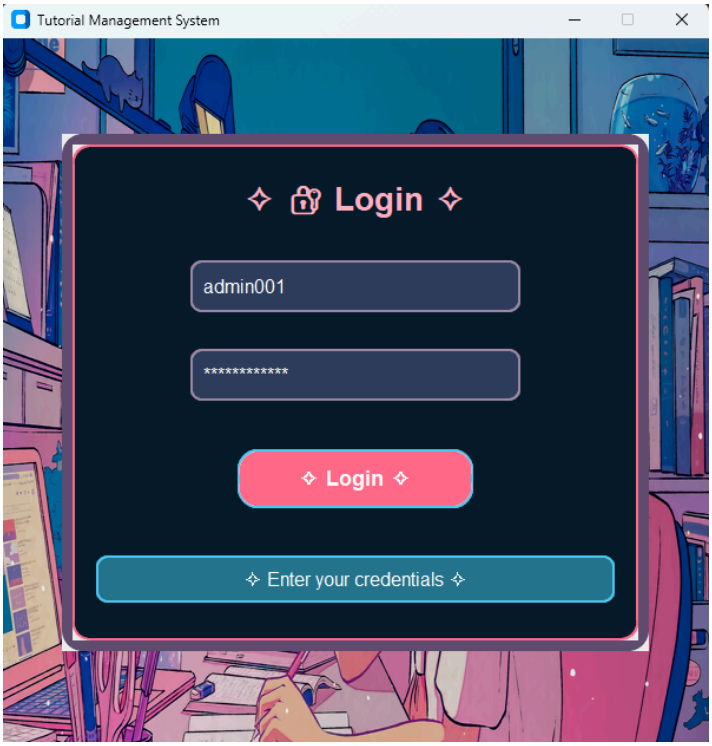
Client

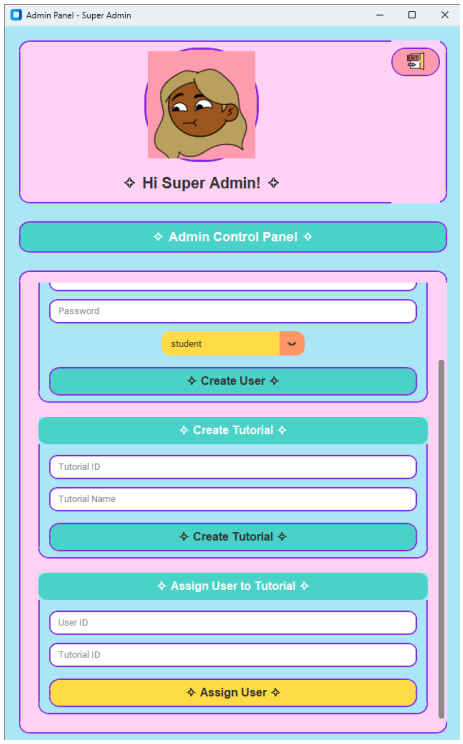
To start the client, ensure the server is up and running and only after that you can run the client_login.py file.

NOTE: To have the Best Experience, Maximize the screen Dashboard and ChatRoom Screens.

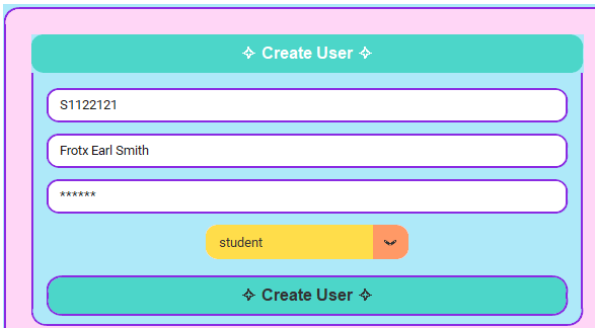
Admin User

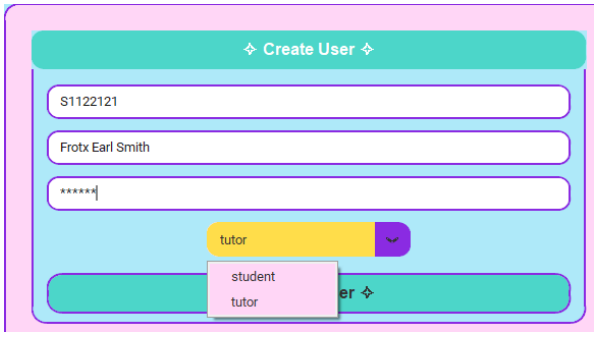
Step 1: Enter your login credentials into the input fields and press enter. You will be directed to your personal dashboard.



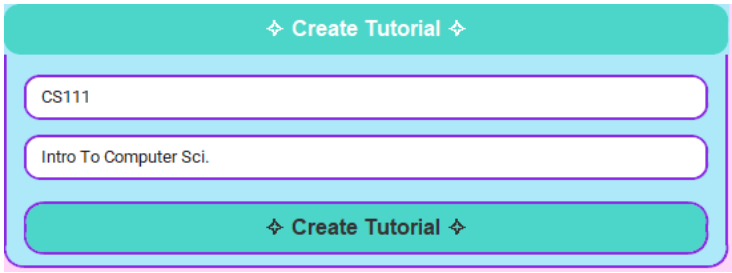


Step 2: To create a new student, enter student name, id and password and press create. Same goes for creating a tutor, just change user type in the dropdown.

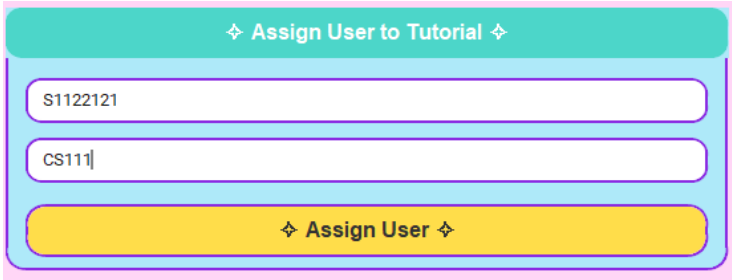




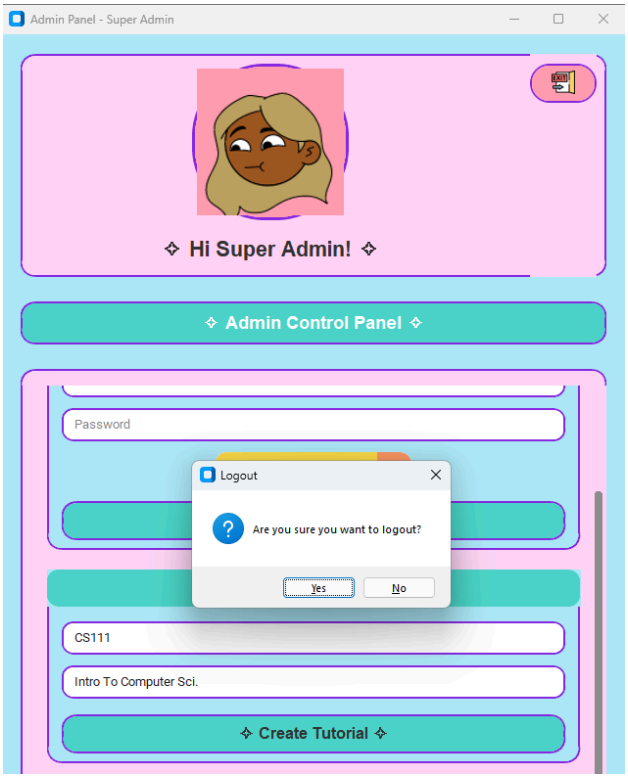
Step 3:When creating a new tutorial, enter its name and id into the textbox and press create.



Step 4: To assign any user to a tutorial, whether a tutor or a student, enter the id of the user and tutorial and press enter.

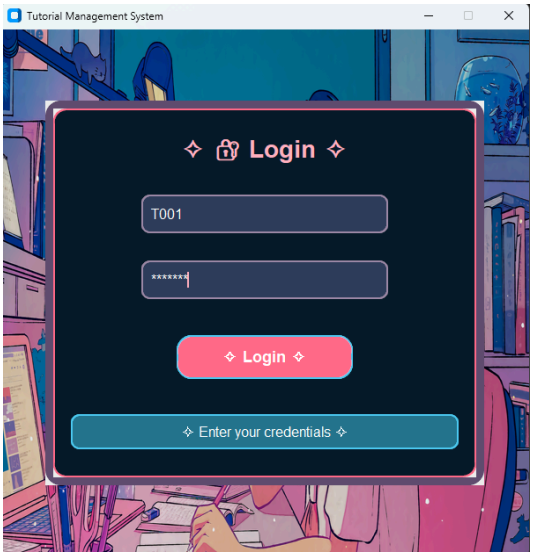


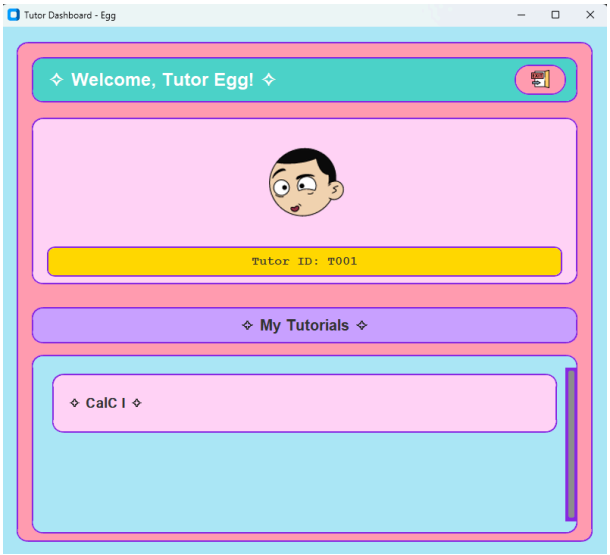
Step 5: To disconnect, you can press the logout button, located at the top right of the screen, to log off the dashboard and be redirected to the login screen, you can also close the window to logout.



Tutor User

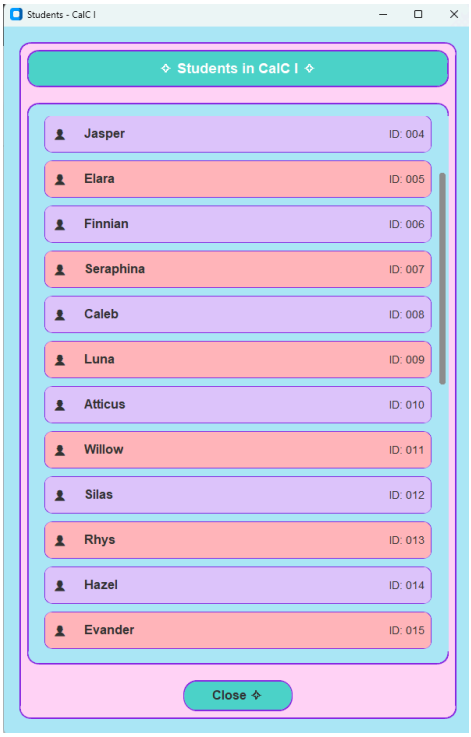
Step 1: Enter your login credentials into the input fields and press enter. You will be directed to your personal dashboard.



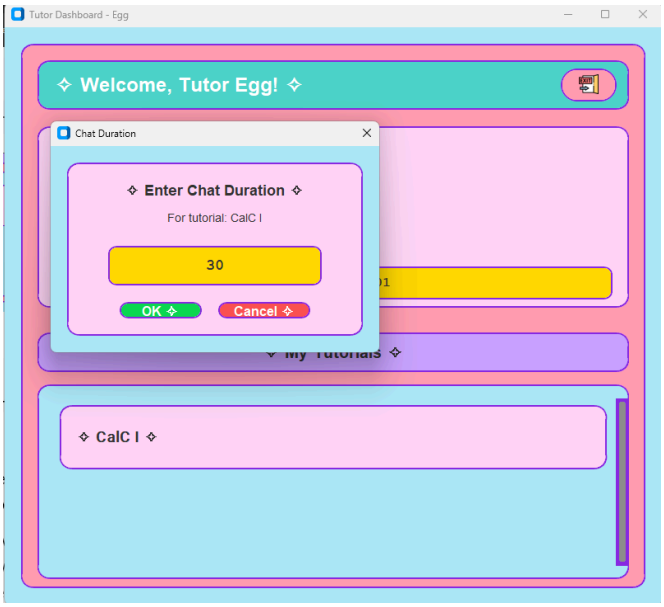


Step 2: Press on the eye button in a tutorial card to see all the allocated participants of the tutorial and press the chat button to start the chat. You will be prompted to enter the duration which must be more than 5 minutes. After entering, press OK to create the tutorial chat room for students to join.

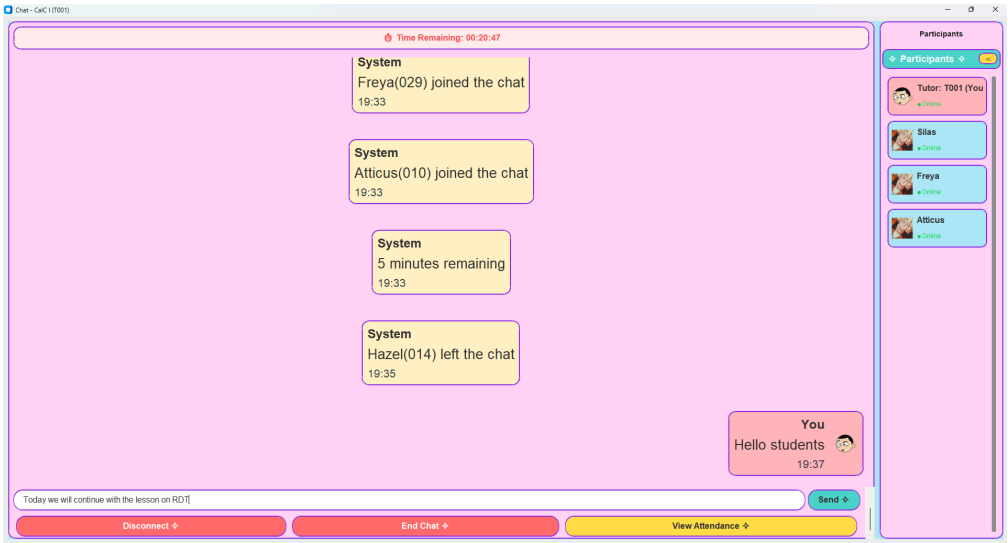
- View Students



- Start New Tutorial

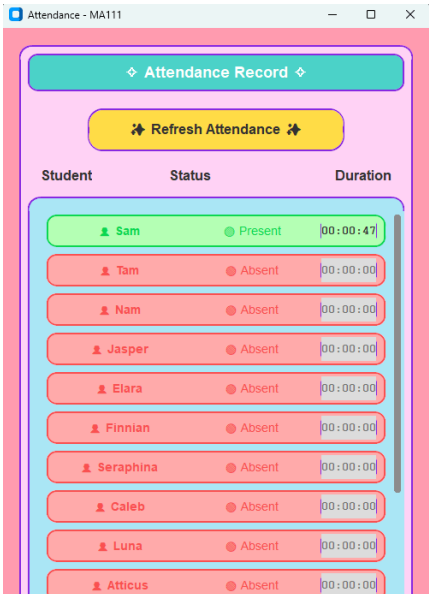


Step 3: Enter a message in the textbox below the chat area to send a message and do so by pressing send or enter on your keyboard.

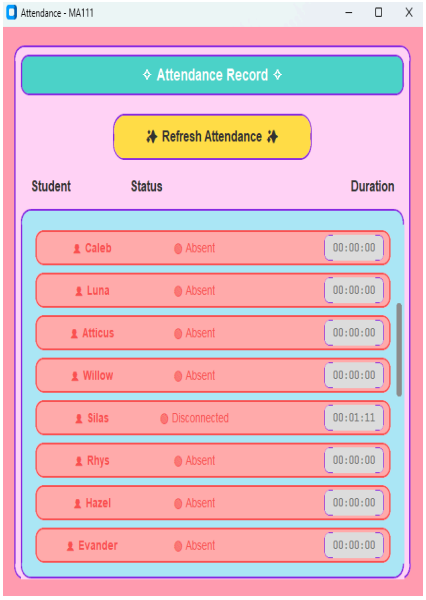


Step 4: To view the attendance of the students that are enrolled in the chat, you can press the View Attendance button to see who all are present in the chat and for how long they were in the chat so that you can decide whether they deserve the attendance or not. If students disconnect, the attendance screen will reflect that the student is disconnected and also reflect a change if the disconnected student rejoined.

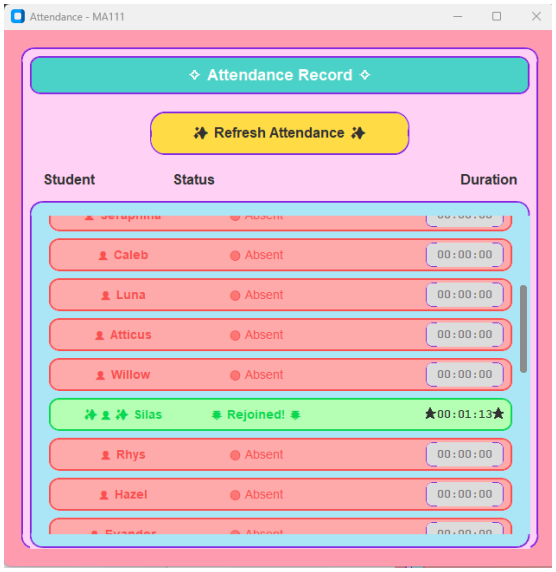
Student Present



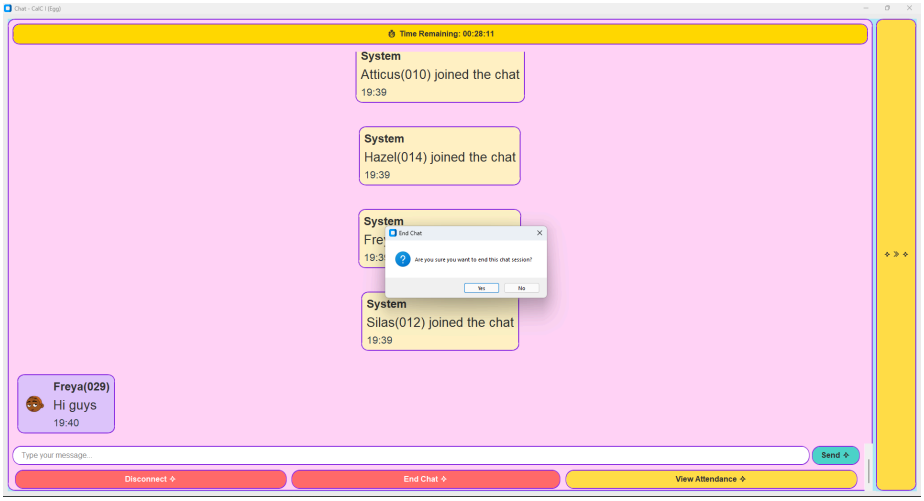
Student Disconnected

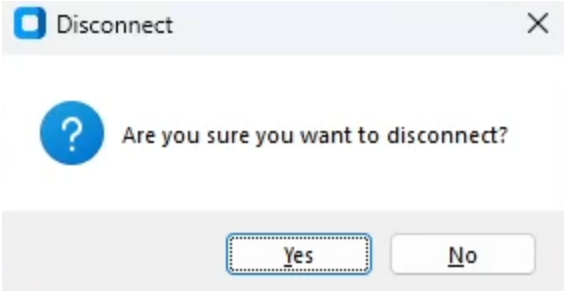


Student Rejoined

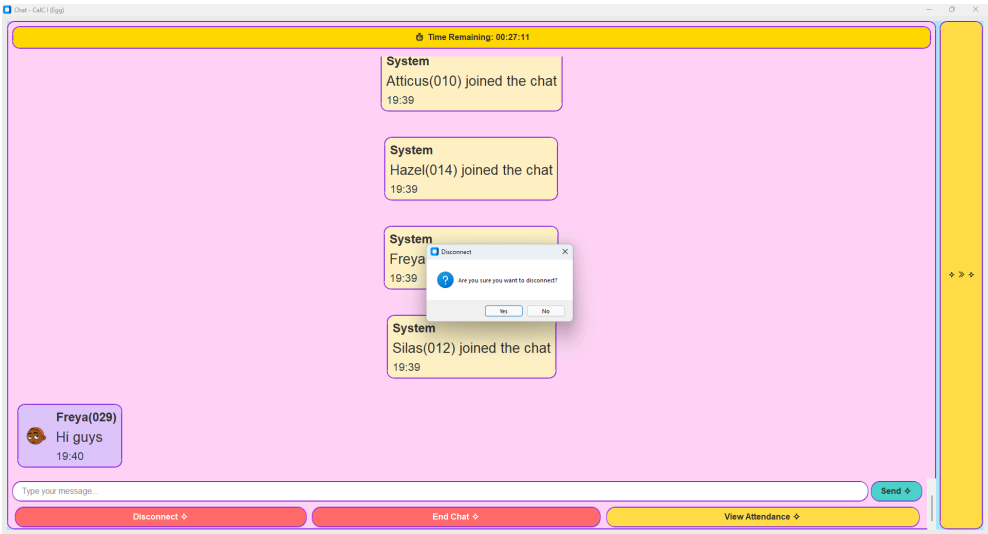


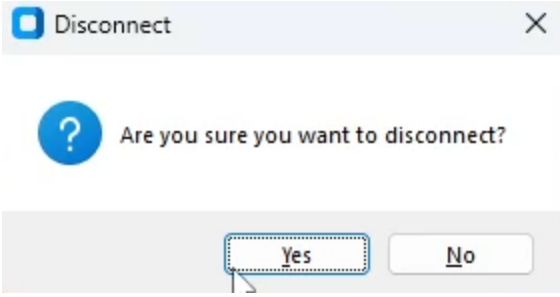
Step 5: For whatever reason, if you want to end your tutorial early, press End Chat and you will be presented with a warning message to check if you want to take the action. Choose yes and the chat will end and you will be directed back to the dashboard.



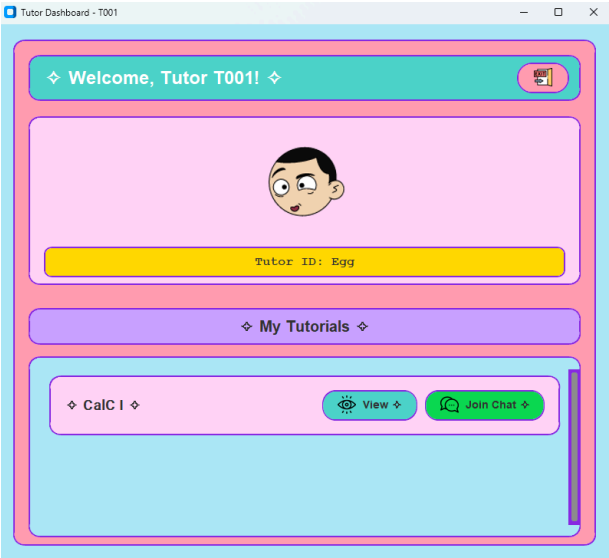


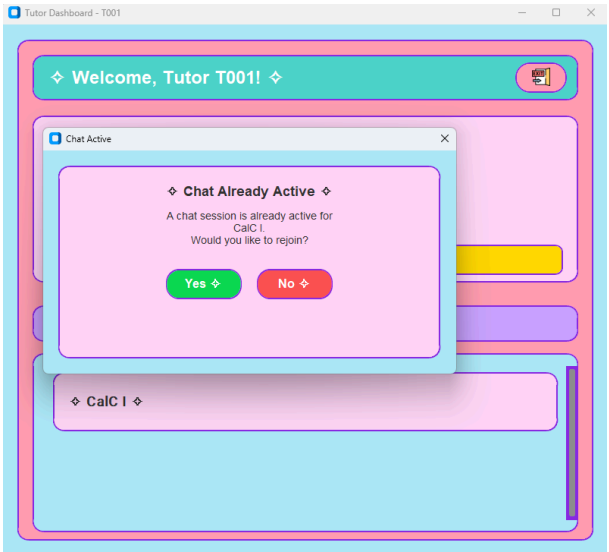
Step 6: If you ever need to leave early from the chat, just press the disconnect button and you will disconnect from the server and be redirected to the dashboard screen, but the chat will still be active if you choose to disconnect without ending the chat. You will be prompted to confirm if you want to leave.



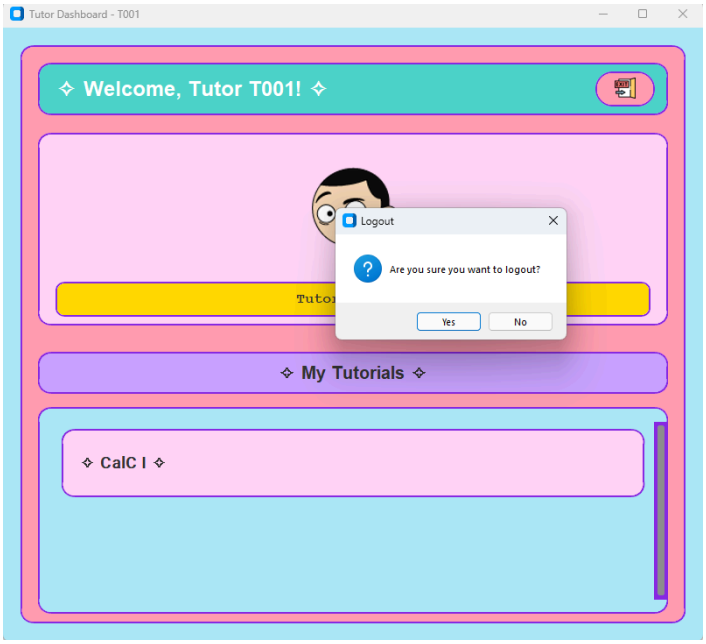


Step 7: If you disconnect from the server without having ended the chat you can re-login and rejoin the active chat and when trying to rejoin the chat, the join chat button will be colour green to show that the chat is in fact active.



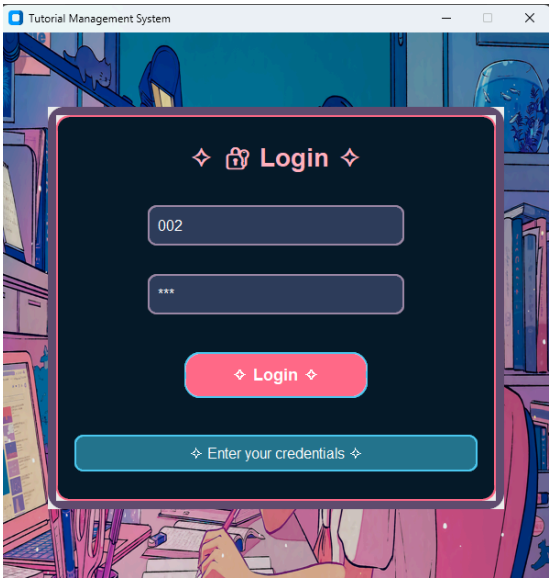


Step 8: To disconnect, you can press the logout button, located at the top right of the screen, to log off the dashboard and be redirected to the login screen, you can also close the window to logout.



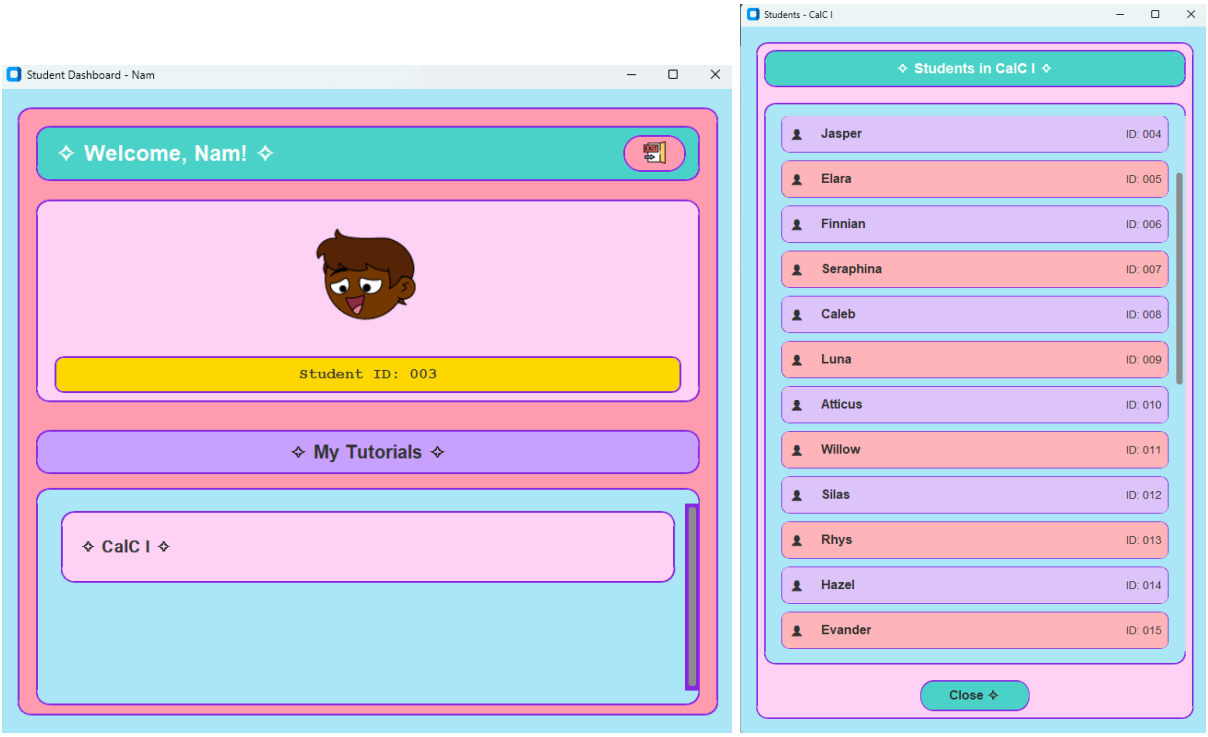
Student User

Step 1: Enter your login credentials into the input fields and press enter. You will be directed to your personal dashboard.



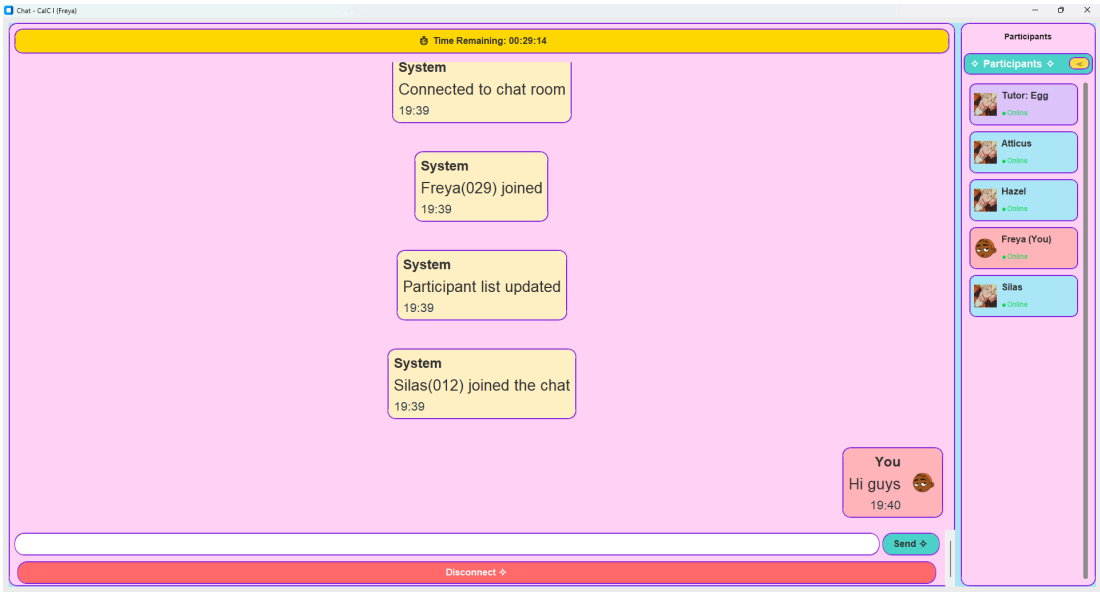
Step 2: Press on the eye button in a tutorial card to see all the allocated participants of the tutorial and press the chat button to enter the chat. Note: if the chat is yet to be activated by the tutor, you will receive an error message saying that the chat is not yet active and will have to wait for the tutor to start.

- View Students

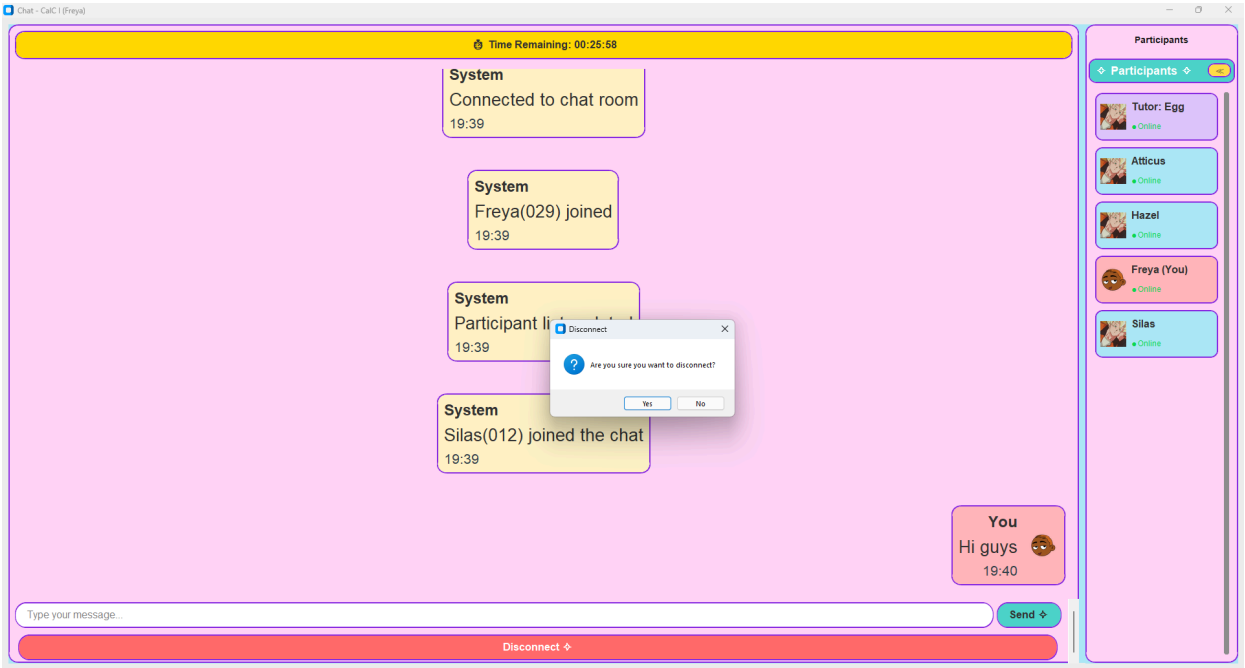


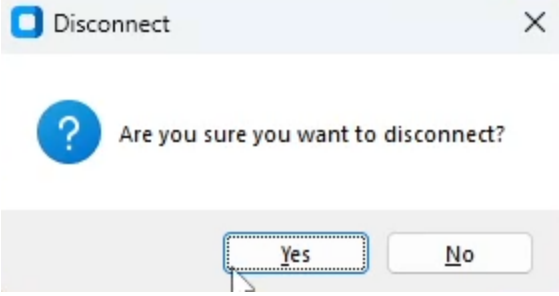
- If Chat is active, the button will be green, else the button will be red.

Step 3: Enter a message in the textbox below the chat area to send a message and do so by pressing send or enter on your keyboard.



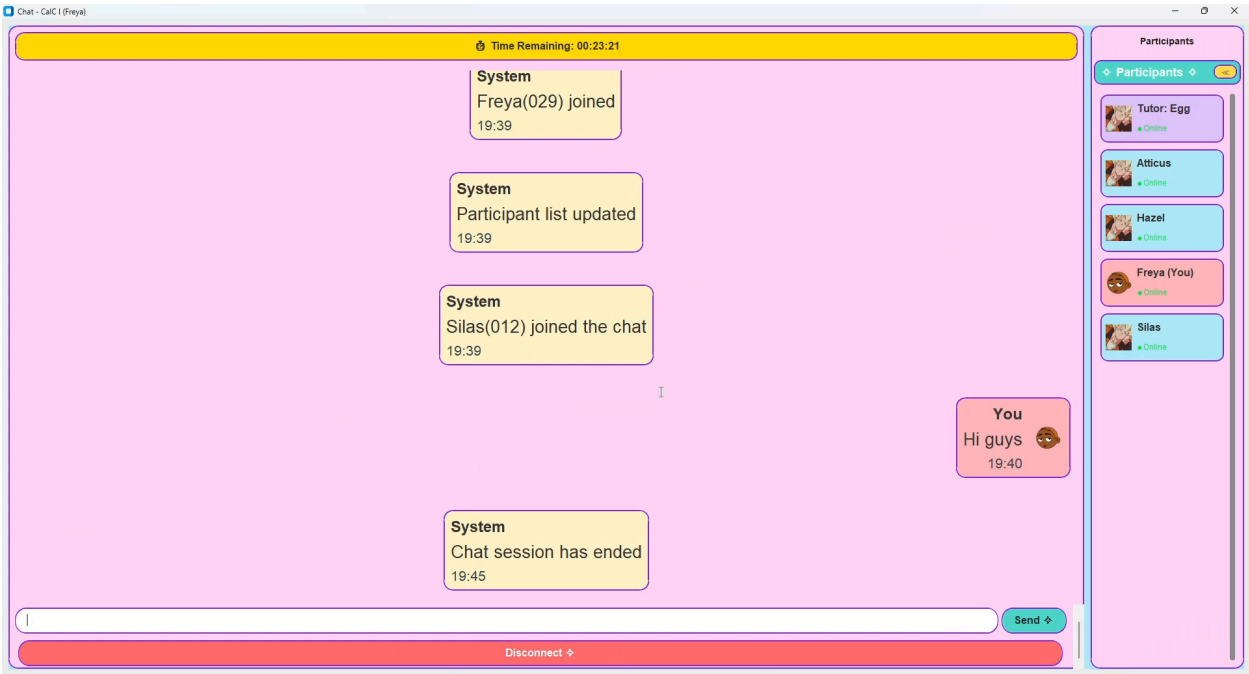
Step 4: If you ever need to leave early from the chat, just press the disconnect button and you will disconnect from the server and be redirected to the dashboard screen, but the chat will still be active. You will be prompted to confirm if you want to leave.



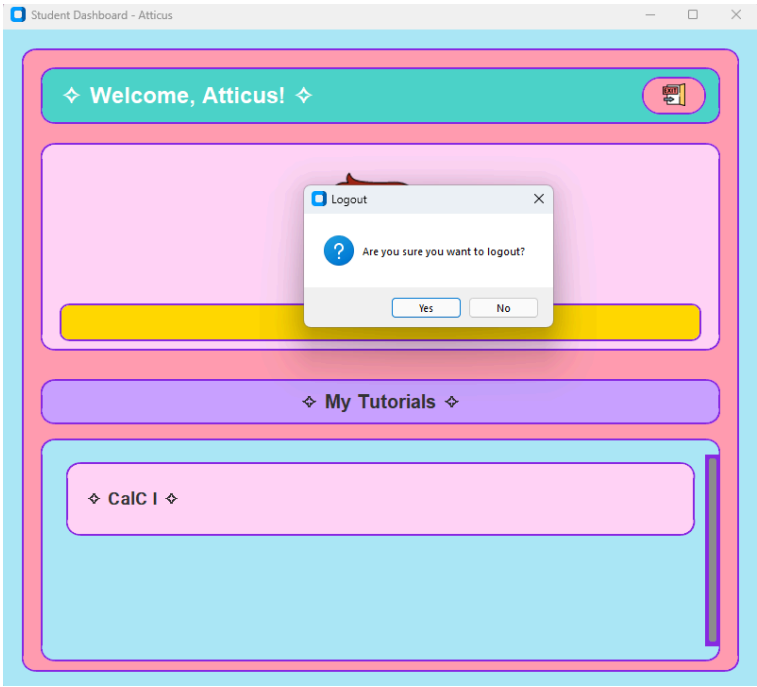


Step 5: If the chat automatically ends or the tutor ends the chat, then you will be directed back to the dashboard where you can press the logout button, located at the top right of the screen, to disconnect from the server or you can close the window.

The system will inform the students connected to chat that the session has ended and they will be redirected back to the dashboard.



Students can now choose to logout if they wish to.



Declaration of Individual Contribution

We declare that the work submitted in this report is our own. Each member equally contributed to design decisions, and integration. Below is the breakdown of each group member's contribution

Student Name	**Contribution**	**Justification**	**Signature**
Shamal Prasad	100%	Chat Room, Database, User Functions, Raw Socket Testing	SPrasad
Ravinesh Narayan	100%	Raw Socket, Dashboard, Chat Room,	RNarayan
Isaiah Narayan	100%	Documentation,Testing,Server Functions	INarayan