Alamein International University

Chatting Room Application (CRA)

21100861	Noor Basm Mokemar
21100829	Abdelrahman Ahmed Haleem
21100847	Salah Ezzat salah
21100871	Mohamed Ahmed Saad

TRELLO BOARD FOR PROJECT PROGRESS TRACKING

To monitor and manage the progress of our project, we have employed Trello as our project management platform. The Trello board for project progress tracking is accessible. This board provides real-time updates on tasks, facilitates collaboration, and enhances overall project visibility.

PROJECT DESCRIPTION

The project is a **Chatting Room Application** designed as a client-server system where authenticated users can communicate with each other. The application allows users to see the status of other users (either "Available" or "Offline"). Additionally, users have the option to save the chat logs locally on their device.

Client Side:

- A Graphical User Interface (GUI) based application with the following features:
  - **Sign-up**: A screen for user registration with fields for username, password, and confirmation password, along with "Sign-up" and "Cancel" buttons.

    - **Login**: A screen for user authentication with fields for username and password, and "Login" and "Cancel" buttons.
    - **Chatting Room**:
        - A ComboBox or radio buttons to toggle between "Available" and "Busy" statuses.
        - A TextArea displaying incoming messages from other users, including the sender's username.
        - A TextField for composing and sending messages.
        - A "Send" button to transmit messages to the server.
        - A "Save Chat Log" button to save the session's chat history to a local text file.

Server Side:

- A GUI-based application responsible for:
    - Managing connections between clients.
    - Handling user sign-up and login processes with database interactions.
    - Exchanging messages between clients.
    - Displaying lists of online and offline users.

USE REQUIREMENTS

The Chatting Room Application is designed to provide basic yet essential features to facilitate user communication securely and efficiently. The key user requirements include:

- **User Authentication**: Users must sign up and log in before accessing the chat room.
- **Real-time Messaging**: Users can send and receive messages in real-time.
- **Status Indicator**: Users can change their status to "Available" or "Busy" to inform others.
- **Chat History Saving**: Users can save their chat logs locally for future reference.

SYSTEM REQUIREMENTS

1. Client Operations:

1.1. The system must allow users to **Sign-Up** by entering a username, password, and confirming the password.
1.2. The system must allow users to **Log In** by entering a username and password.
1.3. The system must provide a **Chatting Room** where:
    1.3.1. Users can change their status to "Available" or "Busy".
    1.3.2. Users can send and receive messages in real-time.
    1.3.3. The chat room must display the messages with the sender's username.
    1.3.4. Users can save the chat log locally by clicking the "Save Chat Log" button.

2. Server Operations:

2.1. The system must handle **User Authentication**:
    2.1.1. The system must allow the admin to add new users to the database.
    2.1.2. The system must allow the admin to edit existing user information such as username and password.
    2.1.3. The system must allow the admin to delete users from the database.
    2.1.4. The system must display a list of registered users, showing their usernames and statuses.

2.2. The system must handle **Message Exchange** between clients:
    2.2.1. The system must support multi-client connections and enable real-time communication.
    2.2.2. The server must correctly route messages to the intended recipients.
2.3. The system must manage **User Status**:
    2.3.1. The system must display a list of online users, showing their status (Available or Busy).
    2.3.2. The system must display a list of offline users.

3. Data Management:

3.1. The system must allow the admin to manage the **User Data**:
    3.1.1. The system must allow the admin to add, edit, and delete user accounts.

3.1.2. The system must show a table of all users with options for CRUD (Create, Read, Update, Delete) operations.

3.2. The system must allow users to manage **Chat Logs**:

3.2.1. The system must allow users to save their chat history to a text file.

3.2.2. The system must allow the admin to view all saved chat logs stored on the server.

4. Additional Features:

4.1. The system must allow the admin to add **CRUD Operations** for specific tasks:

4.1.1. The system must allow the admin to add, edit, and delete information about connected clients.

4.1.2. The system must show a table listing all connected clients, showing their connection status and options for CRUD operations.

4.2. The system must allow the admin to view the list of **Busy Clients** on the server side.

4.3. The system must include a **Controller** for managing all server-side operations and ensuring smooth communication between clients and the server.

NON-FUNCTIONAL REQUIREMENTS:

1. Security:

1.1. The system must protect users' personal data, including usernames and passwords.

1.2. The system must ensure secure communication between clients and the server, preventing unauthorized access.

1.3. User credentials and sensitive information should be encrypted before storage or transmission.

2. Usability:

2.1. The system's user interface should be intuitive and easy to navigate, accommodating users of varying technical skill levels.

2.2. The system should allow users to easily send messages, change their status, and save chat logs without requiring complex procedures.

2.3. Error messages should be clear and provide guidance to users on how to resolve issues.

3. Accessibility:

3.1. The system should be accessible from various devices, including desktops and laptops running on different operating systems (e.g., Windows, macOS, Linux).
3.2. The system should support screen readers and other accessibility tools to ensure that users with disabilities can interact with it effectively.

4. Availability:

4.1. The system must be highly available, ensuring uptime of at least 99.9% throughout the year to support continuous user communication.
4.2. The system should implement failover mechanisms to maintain availability in case of server issues.

5. Maintainability:

5.1. The system should be designed with modular architecture, allowing easy updates, bug fixes, and enhancements without disrupting the entire system.
5.2. The codebase should be well-documented to facilitate maintenance and onboarding of new developers.

6. Scalability:

6.1. The system must be able to scale horizontally to handle an increasing number of users and messages as the application grows in popularity.
6.2. The server should be capable of managing multiple client connections simultaneously without a drop in performance.

7. Performance:

7.1. The system should handle real-time messaging efficiently, ensuring minimal latency between sending and receiving messages.
7.2. The system should quickly load user interfaces and transition smoothly between different screens (e.g., login to chat room).

7.3. The system should ensure fast processing of login credentials and immediate feedback to the user.
7.4. The system should efficiently save chat logs without causing delays in the user experience.

SUBSYSTEMS

Subsystem Name	Subsystem Function	Subsystem Interface
User Authentication Management	It is a subsystem for managing user access to the application. It includes functionalities like: A) User Registration: Adding new users to the system by entering their details like username, password, and email. B) User Login: Authenticating users using their credentials and allowing access to the chat rooms. C) Password Recovery: Allowing users to reset their password using their email.	public function register (Request $request) public function login (Request $request) public function logout (Request $request) public function forgotPassword(Request $request) public function resetPassword(Request $request)

Chat Room Management	It is a subsystem for managing chat rooms within the application. It includes functionalities like: A) Creating and Deleting Chat Rooms: Allowing admins or users to create new chat rooms or delete existing ones. B) Managing Participants: Adding or removing participants from chat rooms. C) Sending and Receiving Messages: Handling the exchange of messages between users in real-time.	public function createRoom(Request $request) public function deleteRoom(string $roomId) public function addUserToRoom(Request $request, string $roomId) public function removeUserFromRoom(Request $request, string $roomId) public function sendMessage(Request $request, string $roomId) public function getMessages(string $roomId)
Message Management	Is a subsystem for managing individual messages within the chat rooms. It includes functionalities like: A) Editing Messages: Allowing users to edit their previously sent messages. B) Deleting Messages: Allowing users to delete their messages from the chat room. C) Displaying Messages: Showing a history of messages within a chat room.	public function editMessage(Request $request, string $messageId) public function deleteMessage(string $messageId) public function getMessageHistory(string $roomId)
User Status Management	It is a subsystem for managing user statuses within the application. It includes functionalities like: A) Updating Status: Allowing users to change their status to available, busy, or offline. B) Displaying Status: Showing the current status of users to others within the chat room.	public function updateStatus(Request $request) public function getStatus(string $userId)
Notification Management	It is a subsystem for handling notifications within the application. It includes functionalities like: A) Sending Notifications: Alerting users when they receive a new message or when a user joins/leaves a chat room.	public function sendNotification(Request $request, string $userId) public function manageNotificationSettings(Request $request)

	B) Managing Notification Settings: Allowing users to customize their notification preferences.	
File Sharing Management	It is a subsystem for managing file sharing within chat rooms. It includes functionalities like: A) Uploading Files: Allowing users to upload files to the chat room. B) Downloading Files: Enabling users to download files shared within the chat room. C) Displaying File Previews: Showing a preview of shared files in the chat.	public function uploadFile(Request $request, string $roomId) public function downloadFile(string $fileId) public function previewFile(string $fileId)
Admin Management	It is a subsystem for managing administrative functions within the application. It includes functionalities like: A) User Management: Allowing admins to add, remove, or modify user accounts. B) Chat Room Configuration: Enabling admins to configure chat room settings like privacy, maximum participants, etc.	public function manageUsers(Request $request) public function configureChatRoom(Request $request, string $roomId)
Analytics and Reporting	It is a subsystem for generating analytics and reports for the application. It includes functionalities like: A) Generating Usage Reports: Providing admins with data on chat room usage, user activity, and message statistics. B) Exporting Data: Allowing admins to export analytics data to CSV or PDF formats.	public function generateReport(Request $request) public function exportReport(Request $request, string $for

Traceability Matrix:

Features	User Authentication	Chat Room Man	Message Management	User Status Mana	Notification Management	File Sharing Management	Admin Mana	Analytics and Reporting

		age men t		geme nt			geme nt	
User Regist ration	✓						✓	
User Login/L ogout	✓						✓	
Passw ord Recove ry	✓						✓	
Create Chat Room		✓					✓	
Delete Chat Room		✓					✓	
Add/Re move Particip ants		✓					✓	
Send/R eceive Messa ges		✓			✓			
Edit Messa ges			✓					
Delete Messa ges			✓					
View Messa ge History		✓	✓					✓
Update User Status				✓				
Display User Status		✓		✓				
Send Notifica tions		✓			✓			

Manage Notification Settings					✓			
Upload Files						✓		
Download Files						✓		
Preview Files		✓				✓		
Manage Users							✓	
Configure Chat Room		✓					✓	
Generate Usage Reports								✓
Export Analytics Data								✓

TEST REQUIREMENT

1. Manage Outcomes:

1.1 **Validate that a new outcome can be added to the outcomes table:** Test that an admin can successfully input and save a new outcome (Abet code, Description, and Category) into the database.

1.2 **Ensure that the added outcome appears in the list:** Verify that the newly added outcome is correctly displayed in the outcomes list with all attributes.

1.3 **Confirm that admin can update the selected outcome with user-friendly input:** Test that the admin can easily modify an outcome's attributes (Abet code, Description, Category) and save the changes.

1.4 **Verify that the updated outcome data is reflected correctly:** Ensure that any changes made to the outcome are accurately displayed in the outcomes list.

1.5 **Check that admin can delete the selected outcome:** Confirm that the admin can remove an outcome from the system.

1.6 **Validate that a delete option is available:** Ensure that the delete option is visible and accessible for all outcomes in the list.

1.7 **Ensure that the deleted outcome is removed from the list:** Verify that once deleted, the outcome no longer appears in the outcomes list.

2. Manage Programs:

2.1 **Validate that a new program can be added to the program table:**
Test that an admin can input and save a new program (Program name, Category) into the database.

2.2 **Validate that duplicate programs are not allowed:**
Ensure the system prevents adding a program with a name that already exists in the database.

2.3 **Ensure that the added program is visible in the program list:**
Verify that the newly added program appears correctly in the program list.

2.4 **Confirm that admin can update the selected program data with user-friendly input:**
Test that the admin can easily modify a program's attributes (Program name, Category) and save the changes.

2.5 **Verify that the updated program data is displayed correctly:**
Ensure that any changes made to the program are accurately displayed in the program list.

2.6 **Check that admin can delete the selected program:**
Confirm that the admin can remove a program from the system.

2.7 **Validate that a delete option is available:**
Ensure that the delete option is visible and accessible for all programs in the list.

2.8 **Ensure that the deleted program is no longer in the program list:**
Verify that once deleted, the program no longer appears in the program list.

3. Manage Courses:

3.1 **Validate that a new course can be added to the courses table:**
Test that an admin can input and save a new course (Course name, Course code, Program) into the database.

3.2 **Ensure the added course data is accurately reflected:**
Verify that the newly added course appears correctly in the courses list with all attributes.

3.3 **Confirm that admin can update the course data with user-friendly input:**
Test that the admin can easily modify a course's attributes (Course name, Course code, Program) and save the changes.

3.4 **Validate that the updated course data is correctly displayed:**
Ensure that any changes made to the course are accurately displayed in the courses list.
3.5 **Check that administrators can delete the selected course:**
Confirm that the admin can remove a course from the system.
3.6 **Validate that a delete option is available:**
Ensure that the delete option is visible and accessible for all courses in the list.
3.7 **Ensure the deleted course data is removed from the course list:**
Verify that once deleted, the course no longer appears in the courses list.

4. Manage Categories:

4.1 **Validate that a new category can be added to the category table:**
Test that an admin can input and save a new category (Category name) into the database.

4.2 **Ensure that the added category is visible in the category list:**
Verify that the newly added category appears correctly in the categories list.
4.3 **Confirm that admin can update the selected category with user-friendly input:**
Test that the admin can easily modify a category's name and save the changes.
4.4 **Verify that the updated category data is displayed correctly:**
Ensure that any changes made to the category are accurately displayed in the categories list.
4.5 **Check that administrators can delete the selected category:**
Confirm that the admin can remove a category from the system.
4.6 **Validate that a delete option is available:**
Ensure that the delete option is visible and accessible for all categories in the list.
4.7 **Ensure that the deleted category is removed from the category list:**
Verify that once deleted, the category no longer appears in the categories list.

5. Manage Courses Offering:

5.1 **Verify that admin can add course offering details (semester, year) with user-friendly input:**
Test that an admin can successfully input and save a new course offering (Course, Year, Semester) into the database.
5.2 **Ensure the added course offering is displayed correctly:**
Verify that the newly added course offering appears correctly in the course offerings list with all attributes.
5.3 **Validate that year >2020 is not allowed:**
Ensure that the system prevents the entry of a course offering a year later than 2020.
5.4 **Validate that only valid semesters (Fall, Spring, Summer) are allowed:**

Ensure that the system restricts semester options to Fall, Spring, or Summer only.

5.5 **Confirm that admin can update the selected course offering (semester, year) with user-friendly input:**
Test that the admin can easily modify a course offering's details and save the changes.

5.6 **Verify that the updated course offering details are correctly reflected:**
Ensure that any changes made to the course offering are accurately displayed in the course offerings list.

5.7 **Check that admin can delete the selected course offering (semester, year):**
Confirm that the admin can remove a course offering from the system.

5.8 **Validate that a delete option is available:**
Ensure that the delete option is visible and accessible for all course offerings in the list.

5.9 **Ensure that the deleted course offering is removed from the list:**
Verify that once deleted, the course offering no longer appears in the course offerings list.

6. **Manage Courses Assessment:**

6.1 **Ensure that the added course assessment data appears correctly:**
Verify that the newly added course assessment data (Outcome, Degree by %, Degree by average) is accurately displayed in the assessments list.

6.2 **Validate that degree by % > 100 is not allowed:**
Ensure that the system restricts the degree percentage to a maximum of 100%.

6.3 **Validate that administrators can add course assessment data with user-friendly input:**
Test that the admin can input and save new course assessment data into the database.

6.4 **Confirm that admin can update the selected course assessment with user-friendly input:**
Test that the admin can easily modify a course assessment's details and save the changes.

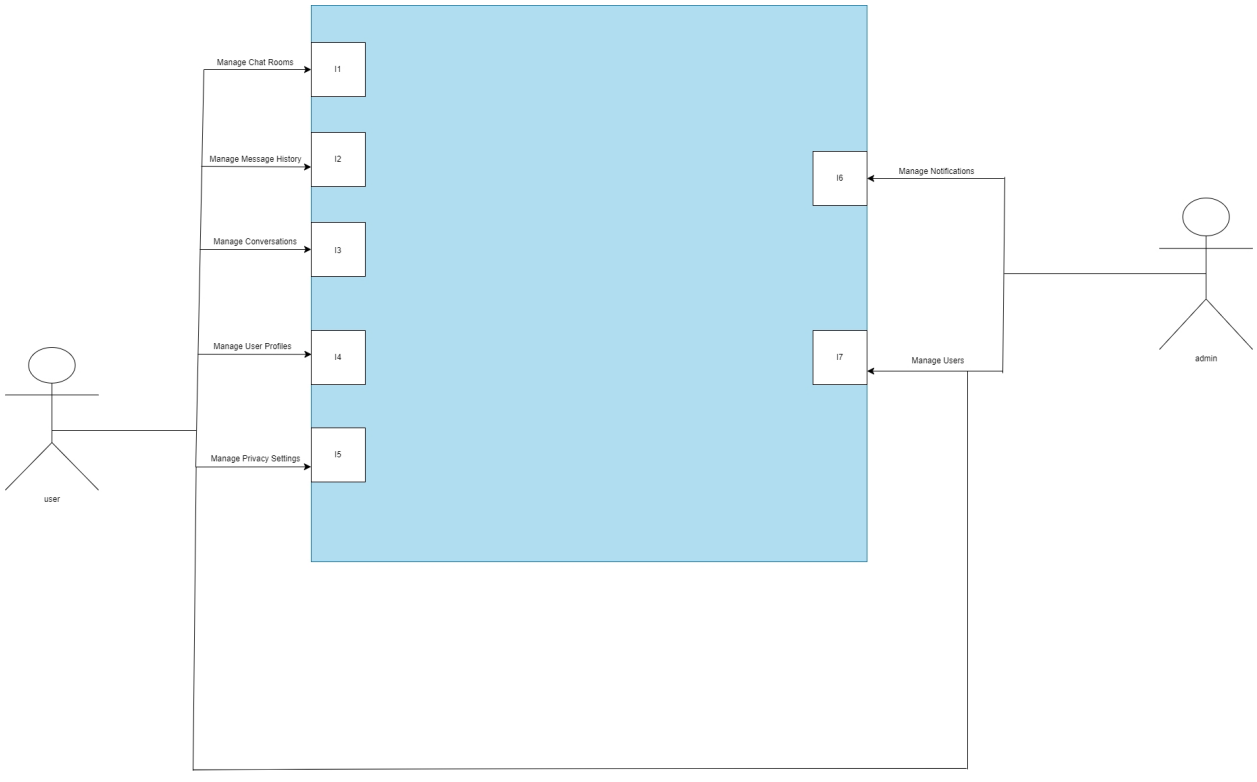
6.5 **Verify that the updated course assessment data is displayed accurately:**
Ensure that any changes made to the course assessment are accurately reflected in the assessments list.

6.6 **Check that admin can delete the selected course assessment:**
Confirm that the admin can remove a course assessment from the system.

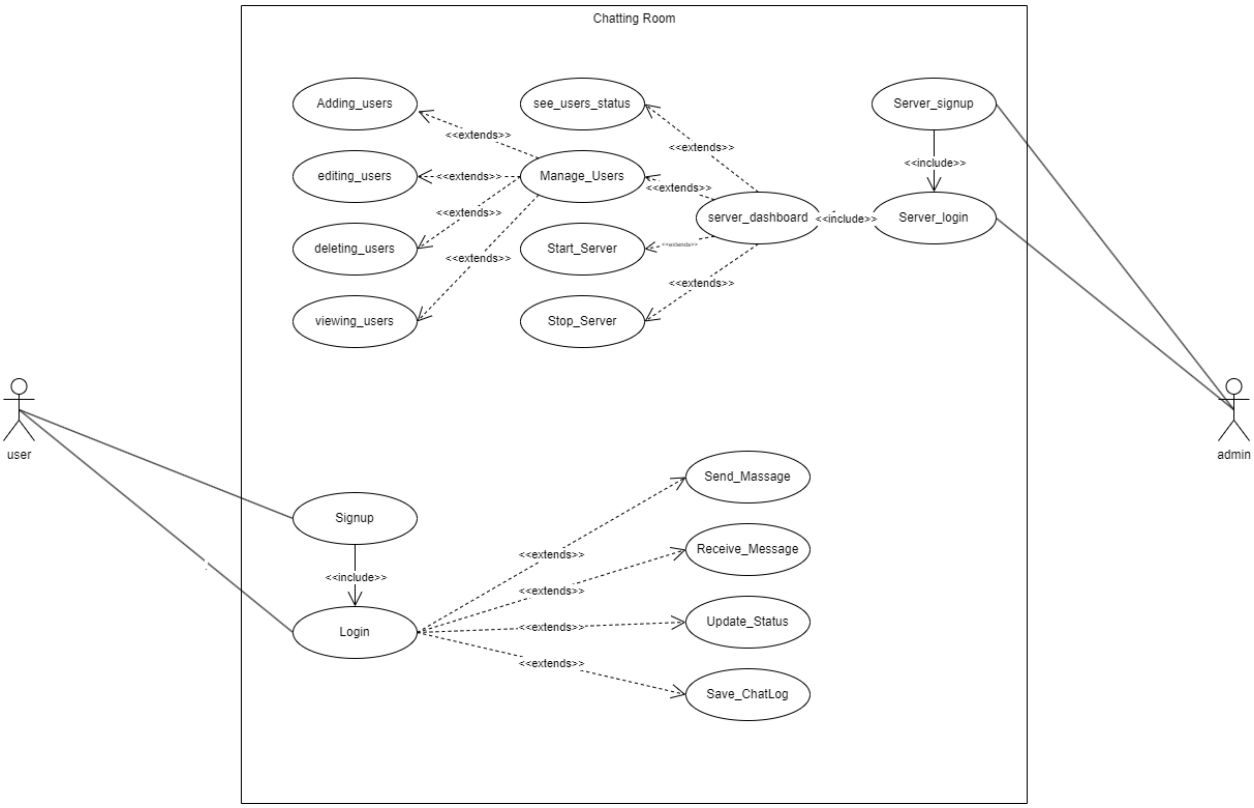
6.7 **Validate that a delete option is available:**
Ensure that the delete option is visible and accessible for all course assessments in the list.

6.8 **Ensure that the deleted course assessment data is removed from the list:**
Verify that once deleted, the course assessment data no longer appears in the assessments list.
6.9 **Confirm that admin can filter data based on year, semester, and course with user-friendly filter options:**
Test that the admin can effectively filter course assessments based on specific criteria (year, semester, course) and view the results.
6.10 **Ensure the filtered data is displayed in a table correctly:**
Verify that the filtered data appears accurately in the table format.
6.11 **Verify that admin can export the filtered data selected by the user in an Excel sheet using user-friendly export options:**
Test that the admin can export filtered data to an Excel file.
6.12 **Ensure that the exported data is correctly saved in an Excel file:**
Confirm that the exported data is accurately saved and can be opened and viewed correctly in an Excel file.

Level-0 Context Diagram



Use case Diagram:



Use case description and infraction scenario:

Actor intentions	System responsibility
Signup The user wants to create a new account in the chat application to gain access to its features. The user provides required information such as username, password, and any additional details required by the system. The user expects confirmation of successful signup or feedback if any issues arise.	Present the user with a signup form requesting necessary information (e.g., username, password, email). Validate the entered data for correctness and uniqueness (e.g., ensuring the username is not already taken and the password meets security criteria). Check for system constraints, like mandatory fields or specific formats (e.g., email format). If validation passes, create the user account and store the details in the database. Provide a success message and either log the user in automatically or direct them to the login page. If any validation fails, prompt the user to correct the issues without losing already entered data.
Login The user or admin wants to authenticate their identity to access the chat application. They provide their username and password to the system. They expect either access to their account or feedback if their credentials are incorrect.	Display a login form where the user/admin can input their username and password. Validate the credentials by checking them against stored data in the database. If the credentials match, grant access to the appropriate area (user dashboard or admin server dashboard).

	If credentials do not match, display an error message, and prompt the user to retry. Track failed login attempts to prevent brute-force attacks and lock accounts, if necessary, after a certain number of failed attempts.
Send Message The user wants to send a message to one or more users in the chat room. The user composes the message and submits it for delivery.	Provide an interface for the user to type and send messages. Once the message is sent, validate it (e.g., ensuring it is not empty or too long). Transmit the message to the server for processing. Ensure that the server broadcasts the message to all connected clients (or specific recipients if in a private chat). Provide feedback to the user, such as confirming the message was sent or notifying them if there was an error.
Receive Message The user wants to receive and read messages from other participants in the chat room. The user expects to be notified when a new message arrives.	Listen for incoming messages from the server. Notify the user of new messages (e.g., with a sound alert, visual indicator, or notification). Display the message in the chat interface in the correct order with relevant details (sender, timestamp). Ensure the message is correctly formatted and legible.

Update Status The user wants to update their availability status (e.g., "Online", "Busy", "Away") to inform other users of their current availability. The user expects this status update to be visible to all other users.	Provide options for the user to change their status within the chat application. Send the updated status to the server once the user makes a selection. Broadcast the updated status to all connected clients so they can see the user's current availability. Update the status display within the user interface to reflect the change.
Save Chat Log The user wants to save a record of their chat conversation to review later. The user selects the option to save the chat log and expects it to be stored on their device.	Compile the chat history that the user wants to save. Prompt the user to select a location on their device to save the chat log. Write the chat history to a file in the chosen location and in a readable format (e.g., .txt, .html). Provide confirmation that the chat log was successfully saved or an error message if the process fails (e.g., due to lack of storage space or permissions issues).
Manage Users The admin wants to manage user accounts, including adding new users, editing existing user information, deleting users, or viewing user details. The admin expects the system to execute these tasks efficiently and accurately.	Allow the admin to select an action (add, edit, delete, or view users) within the server dashboard. For adding users: Collect and validate the new user’s information, then store it in the database. For editing users: Retrieve the current user data, allow the admin to make changes, validate the changes, and update the database.

	For deleting users: Confirm the deletion, then remove the user's data from the database. For viewing users: Display the list of users with their details as stored in the system. Ensure all actions are logged and report any errors back to the admin.
Server Signup The admin wants to create a new admin account to gain access to the server management features. The admin expects the account to be created only if the provided information is valid and unique.	Prompt the admin to provide necessary details (username, password, etc.) for account creation. Validate the entered information, ensuring it meets security and uniqueness criteria. Create the admin account if validation is successful, storing the details in the database. Confirm account creation and either redirect the admin to the login page or log them in automatically.
Server Login The admin wants to access the server dashboard to manage the chat application. The admin provides their credentials and expects access to the server management features.	Prompt the admin for their login credentials (username and password). Validate the credentials against the stored admin data. Grant access to the server dashboard if the credentials are correct. If credentials are incorrect, deny access and prompt the admin to try again. Implement security measures like locking the account after multiple failed attempts to prevent unauthorized access.

Use case description

Use Case: Signup

Actors:

- User

Type:

- Primary and essential

Description:

- The user navigates to the signup form to create a new account in the chat application.
- The system presents fields to enter a username and password.
- The user enters the required information.
- The system validates the provided information, ensuring the username is unique and the password meets security criteria.
- If validation is successful, the system creates the account and stores the user data in the database.
- The system displays a confirmation message indicating successful signup.
- The system may also redirect the user to the login page or log them in automatically.

Use Case: Login

Actors:

- User

Type:

- Primary and essential

Description:

- The user navigates to the login page to access their account.
- The system presents fields to enter a username and password.

- The user provides their credentials.
- The system validates the credentials against stored data in the database.
- If the credentials match, the system grants access to the chat room and displays the chat interface.
- If the credentials are incorrect, the system displays an error message and prompts the user to try again.

Use Case: Send Message

Actors:

- User

Type:

- Primary and essential

Description:

- The user composes a message in the chat interface.
- The system provides a field to enter the message and a send button.
- The user sends the message.
- The system validates the message (e.g., ensuring it is not empty).
- The system transmits the message to the server.
- The server broadcasts the message to all connected users.
- The system displays the message in the chat interface, including the sender's username and timestamp.

Use Case: Receive Message

Actors:

- User

Type:

- Primary and essential

Description:

- The system listens for incoming messages from the server.
- When a new message arrives, the system notifies the user (e.g., with a sound alert or visual indicator).
- The system displays the message in the chat interface, including the sender's username and timestamp.

Use Case: Update Status

Actors:

- User

Type:

- Primary and essential

Description:

- The user wants to update their availability status (e.g., "Online", "Busy").
- The system provides options for the user to change their status.
- The user selects a status option.
- The system updates the user's status in the database and broadcasts the updated status to all connected users.
- The system updates the user's status display in the chat interface.

Use Case: Save Chat Log

Actors:

- User

Type:

- Secondary

Description:

- The user wants to save a record of their chat conversation.
- The system provides an option to save the chat log.
- The user selects the option and chooses a location on their device.
- The system compiles the chat history and saves it to the specified location in a readable format.
- The system displays a confirmation message indicating that the chat log was successfully saved.

Use Case: Manage Users

Actors:

- Admin

Type:

- Primary and essential

Description:

- The admin navigates to the user management section in the server dashboard.
- The system displays options to add, edit, delete, and view users.
- Add Users:
    - The admin selects the option to add a new user.
    - The system presents a form to enter user details.
    - The admin submits the form.
    - The system validates the data and creates the user in the database if valid.
    - The system displays a success message.
- Edit Users:
    - The admin selects a user to edit.
    - The system presents a form with the current user details.
    - The admin updates the details and submits the changes.
    - The system validates the data and updates the user in the database if valid.
    - The system displays a success message.
- Delete Users:
    - The admin selects a user to delete.
    - The system prompts for confirmation.
    - If confirmed, the system removes the user from the database.
    - The system displays a success message.
- View Users:
    - The admin selects an option to view the list of users.
    - The system displays the users with their details.

Use Case: Start Server

Actors:

- Admin

Type:

- Primary and essential

Description:

- The admin navigates to the server dashboard to start the server.
- The system provides an option to start the server.

- The admin clicks the start button.
- The system initiates the server, allowing users to connect.
- The system updates the server status to "Running" and displays this status in the dashboard.

Use Case: Stop Server

Actors:

- Admin

Type:

- Primary and essential

Description:

- The admin navigates to the server dashboard to stop the server.
- The system provides an option to stop the server.
- The admin clicks the stop button.
- The system stops the server, disconnecting all users.
- The system updates the server status to "Stopped" and displays this status in the dashboard.

Use Case: Server Signup

Actors:

- Admin

Type:

- Primary and essential

Description:

- The admin navigates to the server signup form to create a new admin account.
- The system presents fields to enter the username and password.
- The admin enters the required information.
- The system validates the provided information for uniqueness and security criteria.
- If validation is successful, the system creates the admin account and stores the data in the database.
- The system displays a confirmation message and may redirect the admin to the login page.

Use Case: Server Login

Actors:

- Admin

Type:

- Primary and essential

Description:

- The admin navigates to the server login page to access the server dashboard.
- The system presents fields to enter the username and password.
- The admin provides their credentials.
- The system validates the credentials against stored admin data in the database.
- If the credentials match, the system grants access to the server dashboard.
- If the credentials are incorrect, the system displays an error message and prompts the admin to try again.

Use Case: See User Status

Actors:

- Admin

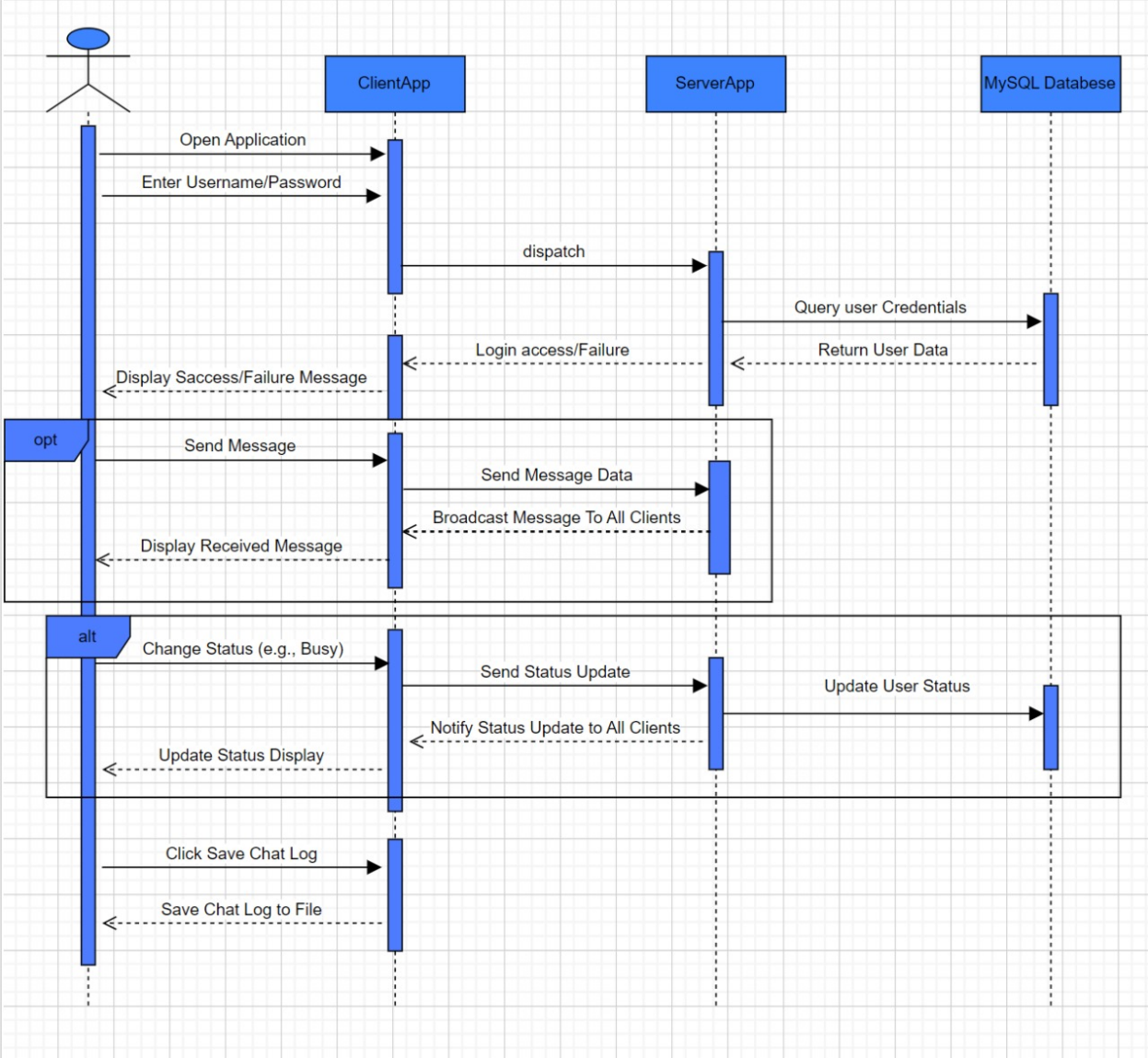
Type:

- Secondary

Description:

- The admin wants to view the current status of all users.
- The system provides a list of users with their current status (e.g., online, busy, offline) in the server dashboard.
- The admin views the list to monitor user activity.

Sequence Diagram



.

TEST REQUIREMENTS

1. User Registration Tests

- **TR-01:** Verify that a new user can successfully register with a unique username and password.
- **TR-02:** Ensure that an error message is displayed if the username already exists.

- **TR-03:** Test for validation errors when the username or password fields are left empty.
- **TR-04:** Confirm that the user is redirected to the login screen after successful registration.

2. User Login Tests

- **TR-05:** Verify that a registered user can log in using valid credentials.
- **TR-06:** Ensure that an error message is displayed for incorrect username or password.
- **TR-07:** Test the login process when fields are left empty.
- **TR-08:** Check that the user is directed to the chat interface after successful login.

3. Chat Interface Tests

- **TR-09:** Verify that users can send and receive messages in the chat room.
- **TR-10:** Ensure that the sender's username appears alongside each message.
- **TR-11:** Test that the chat history is correctly displayed for each user.
- **TR-12:** Verify that the chat window scrolls correctly as new messages are added.
- **TR-13:** Confirm that the user list displays all online users.
- **TR-14:** Test the real-time update of the user list when a user logs in, logs out, or changes their status.
- **TR-15:** Ensure that users can change their status (online, offline, busy) and that it is reflected in the user list.
- **TR-16:** Verify that the "Save Chat Log" button correctly saves the chat history to a file.
- **TR-17:** Test the "Exit" button to ensure it logs the user out and sets their status offline.

4. Server GUI Tests

- **TR-18:** Verify that the server can be started and stopped using the GUI controls.
- **TR-19:** Ensure that the server correctly displays lists of online, offline, and busy users.
- **TR-20:** Test the server's ability to handle multiple client connections simultaneously.
- **TR-21:** Verify that the server can refresh user lists at regular intervals.

- **TR-22:** Ensure that the admin can add, update, and delete users via the server GUI.
- **TR-23:** Confirm that changes made by the admin (e.g., user status updates) reflect immediately in the client interfaces.

5. Database Interaction Tests

- **TR-24:** Verify that user data is correctly stored in the users table upon registration.
- **TR-25:** Ensure that the database correctly updates user status (online, offline, busy) in real-time.
- **TR-26:** Test that all CRUD operations (Create, Read, Update, Delete) for user management function correctly in the database.
- **TR-27:** Confirm that the server_users table is properly accessed for admin login and user management.
-

6. Performance and Stress Tests

- **TR-28:** Test the application's performance with a high number of concurrent users.
- **TR-29:** Verify that the server can handle sudden spikes in user activity without crashing.
- **TR-30:** Test the database's performance under heavy read/write operations.

7. Security Tests

- **TR-31:** Ensure that passwords are securely stored and managed (e.g., hashed) in the database.
- **TR-32:** Test that the application is protected against SQL injection and other common security threats.
- **TR-33:** Verify that unauthorized users cannot access the chat room or server management functionalities.

8. User Interface Tests

- **TR-34:** Verify that all buttons, text fields, and other UI elements are functioning as expected.
- **TR-35:** Ensure that the application's styling is consistent across different screens and resolutions.
- **TR-36:** Test the responsiveness of the UI to ensure it works correctly on different screen sizes.

Test Plan

1. Unit Test Plan:

- **Unit Test 1:** User registration → Verify successful registration.
- **Unit Test 2:** User registration → Validate unique username enforcement.
- **Unit Test 3:** User registration → Test empty fields validation.
- **Unit Test 4:** User login → Verify successful login.
- **Unit Test 5:** User login → Validate error handling for incorrect credentials.
- **Unit Test 6:** User login → Test empty fields validation.
- **Unit Test 7:** Chat interface → Verify sending and receiving messages.
- **Unit Test 8:** Chat interface → Validate sender's username display with messages.
- **Unit Test 9:** Chat interface → Test chat history display.
- **Unit Test 10:** Chat interface → Verify user list display of online users.
- **Unit Test 11:** Chat interface → Test real-time update of user list.
- **Unit Test 12:** Chat interface → Validate status change functionality (online, offline, busy).
- **Unit Test 13:** Chat interface → Verify "Save Chat Log" functionality.
- **Unit Test 14:** Chat interface → Test "Exit" button functionality.
- **Unit Test 15:** Server GUI → Verify server start and stop controls.
- **Unit Test 16:** Server GUI → Validate user lists (online, offline, busy).
- **Unit Test 17:** Server GUI → Test handling of multiple client connections.
- **Unit Test 18:** Server GUI → Verify admin user management functionalities (add, update, delete users).
- **Unit Test 19:** Database → Validate user data storage upon registration.
- **Unit Test 20:** Database → Test real-time update of user status.
- **Unit Test 21:** Database → Verify CRUD operations for user management.
- **Unit Test 22:** Database → Ensure secure password handling and storage.
- **Unit Test 23:** Security → Test protection against SQL injection.
- **Unit Test 24:** UI → Validate the responsiveness of the UI.
- **Unit Test 25:** Performance → Test application performance with high user load.
- **Unit Test 26:** Compatibility → Verify application compatibility across different OS and Java versions.

2. Regression Test Plan:

- **Regression testing 1:** Retest User registration → U1
- **Regression testing 2:** Retest User registration → U2

- **Regression testing 3:** Retest User registration → U3
- **Regression testing 4:** Retest User login → U4
- **Regression testing 5:** Retest User login → U5
- **Regression testing 6:** Retest User login → U6
- **Regression testing 7:** Retest Chat interface → U7
- **Regression testing 8:** Retest Chat interface → U8
- **Regression testing 9:** Retest Chat interface → U9
- **Regression testing 10:** Retest Chat interface → U10
- **Regression testing 11:** Retest Chat interface → U11
- **Regression testing 12:** Retest Chat interface → U12
- **Regression testing 13:** Retest Chat interface → U13
- **Regression testing 14:** Retest Chat interface → U14
- **Regression testing 15:** Retest Server GUI → U15
- **Regression testing 16:** Retest Server GUI → U16
- **Regression testing 17:** Retest Server GUI → U17
- **Regression testing 18:** Retest Server GUI → U18
- **Regression testing 19:** Retest Database → U19
- **Regression testing 20:** Retest Database → U20
- **Regression testing 21:** Retest Database → U21
- **Regression testing 22:** Retest Security → U22
- **Regression testing 23:** Retest Security → U23
- **Regression testing 24:** Retest UI → U24
- **Regression testing 25:** Retest Performance → U25
- **Regression testing 26:** Retest Compatibility → U26

3. Integration Test Plan:

- **Integration Test 1:** User registration and login → U1 & U2 & U3 & U4 & U5 & U6
- **Integration Test 2:** Chat interface → U7 & U8 & U9 & U10 & U11 & U12 & U13 & U14
- **Integration Test 3:** Server GUI → U15 & U16 & U17 & U18
- **Integration Test 4:** Database and Security → U19 & U20 & U21 & U22 & U23
- **Integration Test 5:** UI and Performance → U24 & U25 & U26

4. Overall Test Plan:

- Unit Test 1
- Unit Test 2
- Regression testing 1

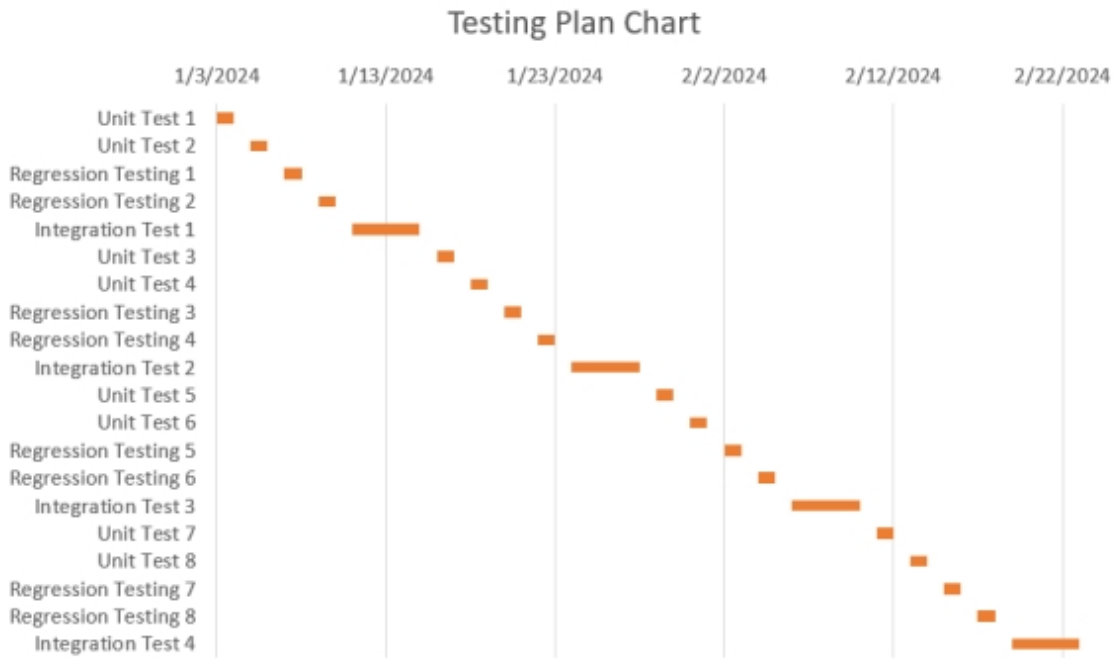
- Regression testing 2
- Unit Test 3
- Unit Test 4
- Regression testing 3
- Regression testing 4
- Integration Test 1
- Unit Test 5
- Unit Test 6
- Regression testing 5
- Regression testing 6
- Unit Test 7
- Unit Test 8

5. Regression Test Plan:

- Regression testing 8
- Integration Test 2
- Unit Test 9
- Unit Test 10
- Regression testing 9
- Regression testing 10
- Unit Test 11
- Unit Test 12
- Regression testing 11
- Regression testing 12
- Integration Test 3
- Unit Test 13
- Unit Test 14
- Regression testing 13
- Regression testing 14
- Unit Test 15
- Unit Test 16
- Regression testing 15
- Regression testing 16
- Integration Test 4
- Unit Test 17
- Unit Test 18
- Regression testing 17

- Regression testing 18
- Unit Test 19
- Unit Test 20
- Regression testing 19
- Regression testing 20
- Integration Test 5
- Unit Test 21
- Unit Test 22
- Regression testing 21
- Regression testing 22
- Unit Test 23
- Unit Test 24
- Regression testing 23
- Regression testing 24
- Unit Test 25
- Unit Test 26
- Regression testing 25
- Regression testing 26
- Integration Test 6

GANTT CHART


Testing Plan Chart
1/3/2024
1/13/2024
1/23/2024
2/2/2024
2/12/2024
2/22/2024
Unit Test 1
Unit Test 2
Regression Testing 1
Regression Testing 2
Integration Test 1
Unit Test 3
Unit Test 4
Regression Testing 3
Regression Testing 4
Integration Test 2
Unit Test 5
Unit Test 6
Regression Testing 5
Regression Testing 6
Integration Test 3
Unit Test 7
Unit Test 8
Regression Testing 7
Regression Testing 8
Integration Test 4