

ChatApp: Real-Time Messaging Web Application

Built with ASP.NET 8.0, SignalR, and JavaScript

1.Introduction

ChatApp is a modern, real-time web-based messaging platform developed using **ASP.NET Core MVC**, **SignalR**, and **JavaScript/jQuery**. The application enables seamless real-time communication between multiple users in a group chat environment, supporting rich features like replies, edits, deletions, file sharing, message pinning, and more.

2.Key Objectives

- **Real-time group communication using SignalR**, enabling instant message broadcast across connected clients without the need for manual refresh or polling mechanisms.
- **Dynamic group-based messaging support**, allowing users to switch between chat groups while preserving and retrieving their respective conversation histories from persistent storage.
- **Reply, edit, delete, forward, and pin functionality**, offering a comprehensive suite of interaction options to manage conversations effectively and maintain clarity during communication threads.
- **File upload support (image, PDF, Word, Excel, etc.)**, including inline preview or secure download links depending on the file type, thereby enriching message content and facilitating document sharing.
- **User interface enhancements such as typing indicators, real-time search, and message notifications**, contributing to an intuitive, interactive, and seamless chat experience across various devices and screen sizes.
- **Chat logging and persistence**, using JSON-formatted `.txt` files per group to ensure that all messages—including file links and reply chains—are stored securely and can be reloaded as chat history.
- **Clean separation of concerns with scalable architecture**, incorporating a SignalR hub, controller logic, and service layer to ensure modularity, maintainability, and easy extensibility.

Table of Contents

1. **Introduction**
2. **Key Objectives**
3. **Architecture**
 - 3.1 Client Tier (Presentation Layer)
 - 3.2 Application Tier (Server Layer)
 - 3.3 Data Tier (Storage Layer)
4. **Functional Diagram**
5. **Conceptual Diagram**
6. **Functionalities**
7. **Dependencies**
 - 7.1 Framework
 - 7.2 Client Libraries
 - 7.3 Backend Libraries
8. **Conclusion**
9. **Diagrams and Screenshots**

3. Architecture

The application follows a 3-tier architecture consisting of:

3.1 Presentation Layer (Client Tier)

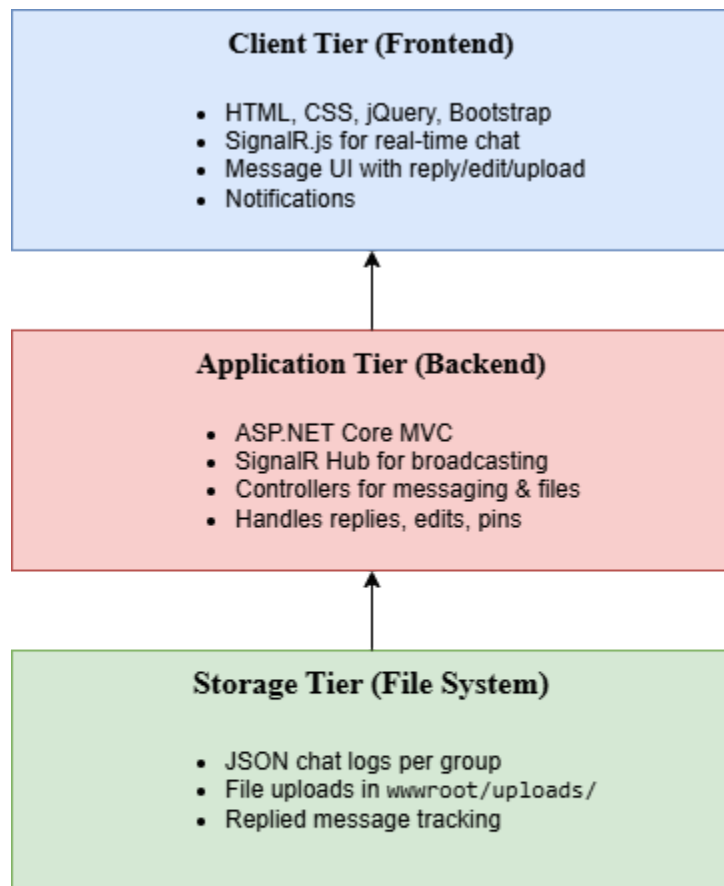
- Built using **HTML**, **CSS**, **Bootstrap**, and **JavaScript/jQuery**.
- Communicates with the backend via AJAX and SignalR WebSockets.
- Dynamic rendering of messages with proper alignment (sender/receiver).
- UI includes message box, chat history, file uploader, emoji picker, search bar, and notification bubble.

3.2 Application Layer (Server Tier)

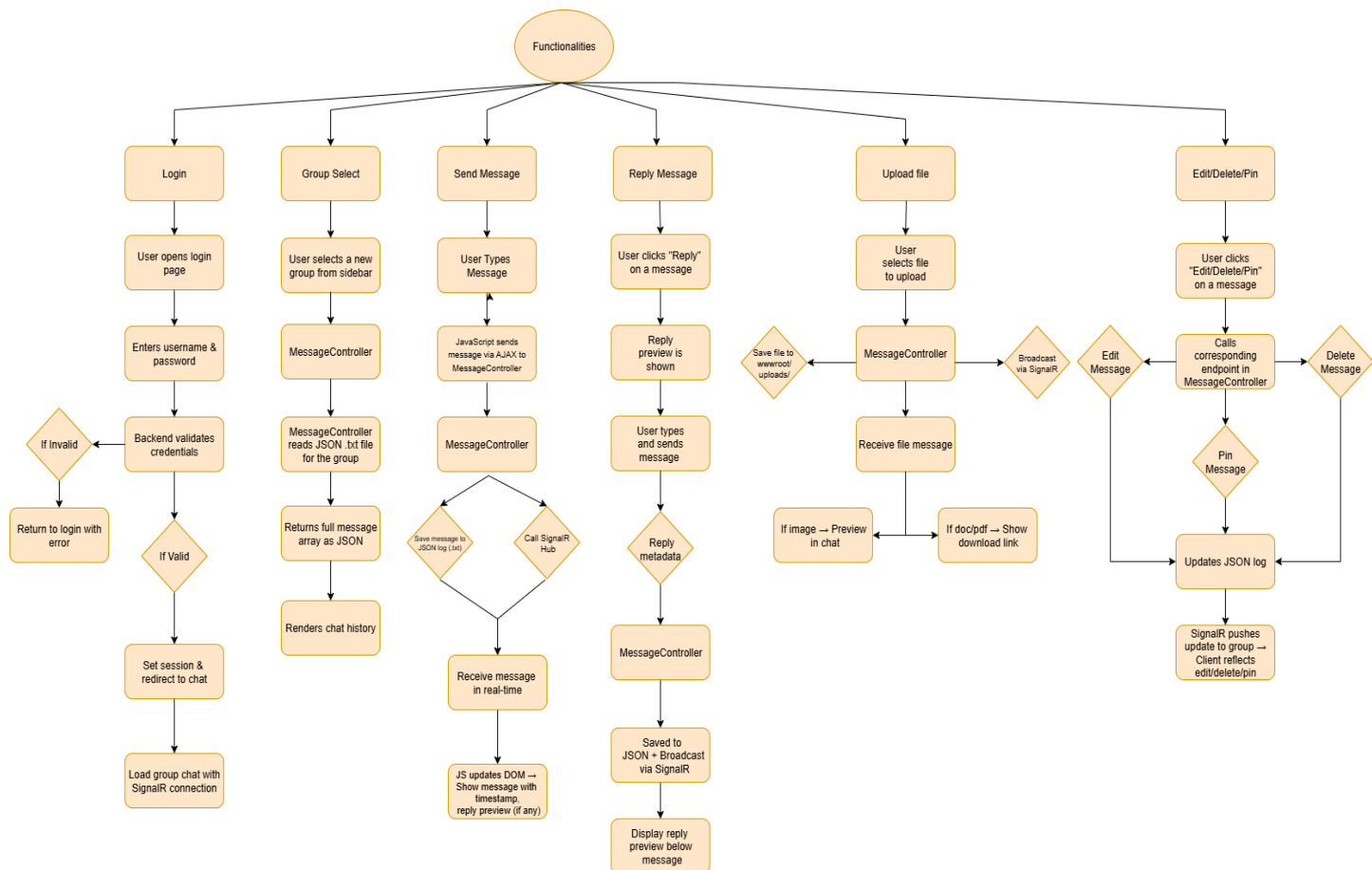
- Developed using **ASP.NET Core MVC** and **SignalR** hub classes.
- Handles message broadcasting, group communication, file path creation, message replies/edits, and tracking chat state.
- Maintains message structure and business logic via controllers and models.

3.3 Data Layer (Storage Tier)

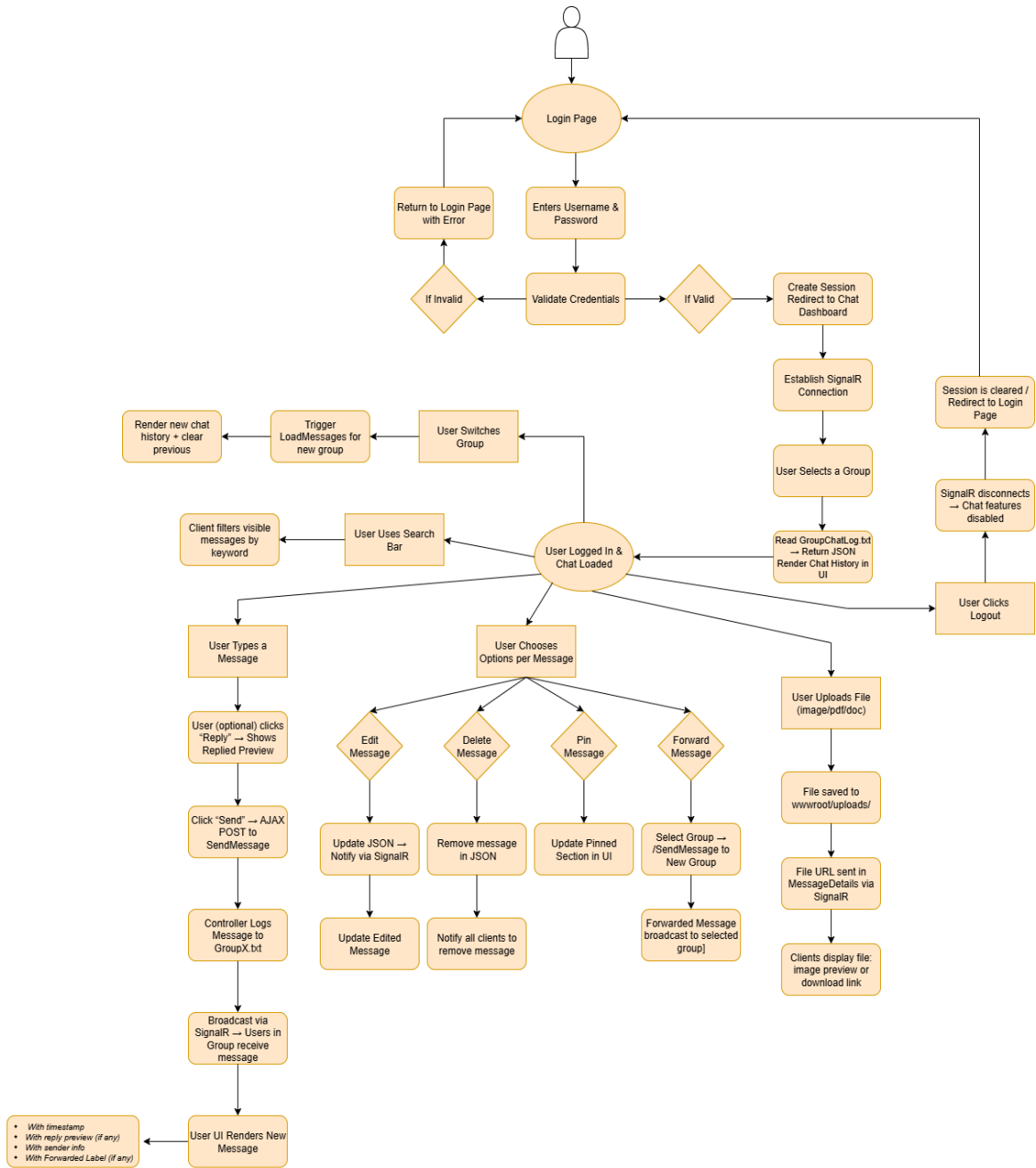
- Stores messages in a **JSON-formatted .txt file** per group chat.
- Files are saved in the `wwwroot/uploads/` directory.
- Chat history is read from the text file and broadcasted on group selection.



4.Functional Diagram:



5. Conceptual Diagram:



6. Functionalities

1. Real-Time Messaging

The application enables real-time communication using SignalR. Messages are instantly broadcasted to all users in the selected group without requiring a page refresh. This ensures a seamless and dynamic user experience, simulating modern messaging apps like WhatsApp or Slack.

2. Group-Based Chat

Users can participate in multiple chat groups. Each group maintains an independent message history stored in a `.txt` file using JSON format. When a user switches groups, the corresponding chat history is loaded automatically.

3. Message Reply with Preview

Users can reply to specific messages. The replied message displays in a smaller preview above the new message, including the original sender's name and message content, maintaining clarity in threaded discussions.

4. Edit and Delete Messages

Sent messages can be edited or deleted by the sender. When edited, a subtle "(edited)" label is shown. Deleted messages are either removed or replaced with a placeholder, depending on implementation.

5. Forward Messages

Users can forward messages from one group to another, maintaining content integrity. This promotes message reusability without copy-paste.

6. Pinned Messages

Important messages can be pinned by the user. Pinned messages are stored in a dedicated section and remain accessible at the top or in a toggleable list.

7. File Uploads and Sharing

Users can upload files, including images, PDFs, Word, and Excel documents.

- **Images** are shown as previews within the chat.

- **Other files** appear as clickable download links.


Files are stored in the `wwwroot/uploads/` directory and their links are sent through SignalR.

8. Search Messages

A search bar is available for users to search through the loaded chat history using

keywords. Matching messages are dynamically highlighted or filtered.

9. Notification Alerts

Notifications (e.g., " New Message Received") appear when a new message is sent in a minimized or inactive chat panel. This keeps users informed of incoming messages.

10. Chat History Retrieval

All messages, including their metadata (timestamp, sender, reply context, file link), are stored in JSON format in a `.txt` file for each group. On group selection, this file is parsed and the full conversation is displayed.

7. Dependencies

7.1 Framework

- **ASP.NET 8.0 or higher**
Provides the foundation for building scalable MVC applications and REST APIs. Manages routing, controller actions, model binding, and serves static files.
- **SignalR (Microsoft.AspNetCore.SignalR)**
Enables real-time server-client communication using WebSockets or fallbacks. Used to broadcast messages instantly to all users in a selected group.

7.2 Client-Side Libraries:

Library	Purpose
jQuery	Simplifies DOM manipulation, AJAX communication, and event handling.
SignalR.js	SignalR JavaScript client that connects to the backend hub for real-time updates.
Bootstrap	Provides responsive design, layout, and UI components like buttons and modals.
Font Awesome(optional)	Used for icons in message controls (reply, edit, delete, pin, etc.)

7.3 Backend Libraries:

Library	Purpose
System.IO	Handles reading/writing chat messages and file uploads to/from disk.
System.Text.Json	Serializes and deserializes messages in JSON format for logging and retrieval.
System.Threading.Tasks	Enables asynchronous operations (e.g., file uploads and background tasks).
Microsoft.AspNetCore.Mvc	Handles routing, controllers, and model binding within the ASP.NET Core MVC framework.
Microsoft.AspNetCore.Http	Used for handling file uploads and HTTP context information.

7.4 Storage:

- **Local File System ([wwwroot/uploads/](#))**
Used to store uploaded files such as images, documents, and media.
 - **JSON [.txt](#) Logs**
Each group has a dedicated [.txt](#) file storing chat history in JSON format. These logs are used for loading previous messages.
-

8. Conclusion

The **TeamChat** application delivers a robust, real-time communication platform built with **ASP.NET Core MVC** and **SignalR**, providing seamless and dynamic group messaging functionalities. By integrating modern messaging features such as **message replies**, **edits**, **deletions**, **forwards**, **pinning**, and **file uploads**, the system ensures a user-friendly and interactive experience across web environments.

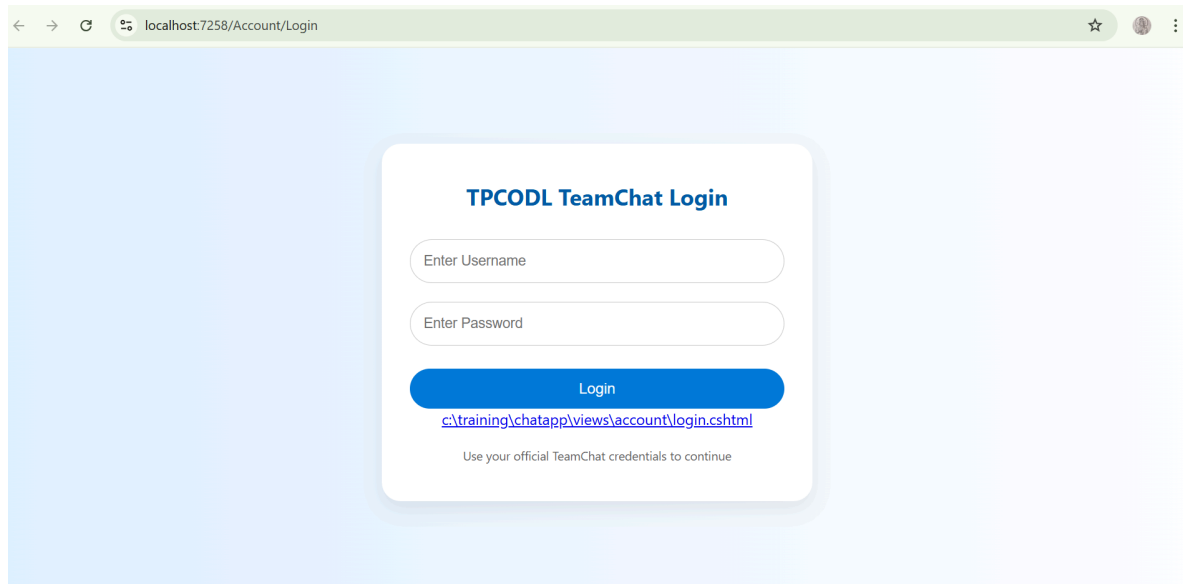
The backend architecture is designed for **scalability and modularity**, enabling real-time broadcasting, structured message logging, and smooth group-based message management. Messages are persistently stored in JSON format within **.txt** files, allowing for lightweight yet effective chat history retrieval. Additionally, features like **typing indicators**, **search**, and **notification alerts** enhance usability and bring the experience closer to that of modern professional chat platforms.

With support for **rich media sharing**, structured chat history, and real-time synchronization, TeamChat serves as a foundational system that can be further extended to include user authentication, role-based access control, message reactions, and database-backed storage for enterprise-grade deployments.

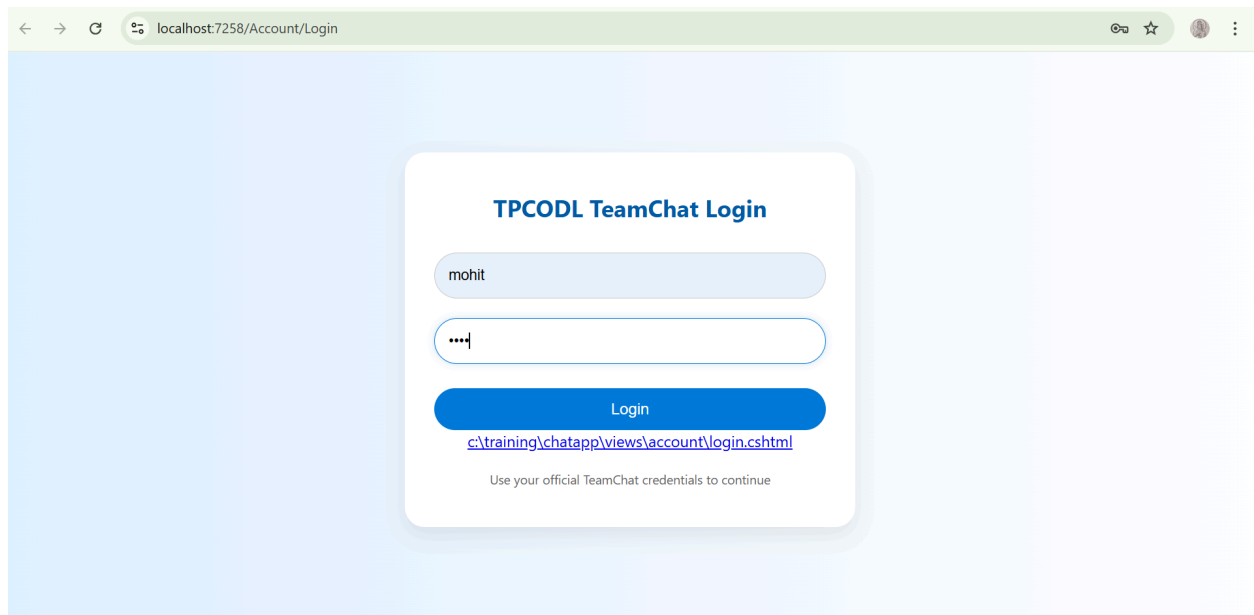
Overall, the project demonstrates a complete and reliable chat communication framework with real-time capabilities, built using industry-standard technologies and best practices.

9.Snippets:

Login:

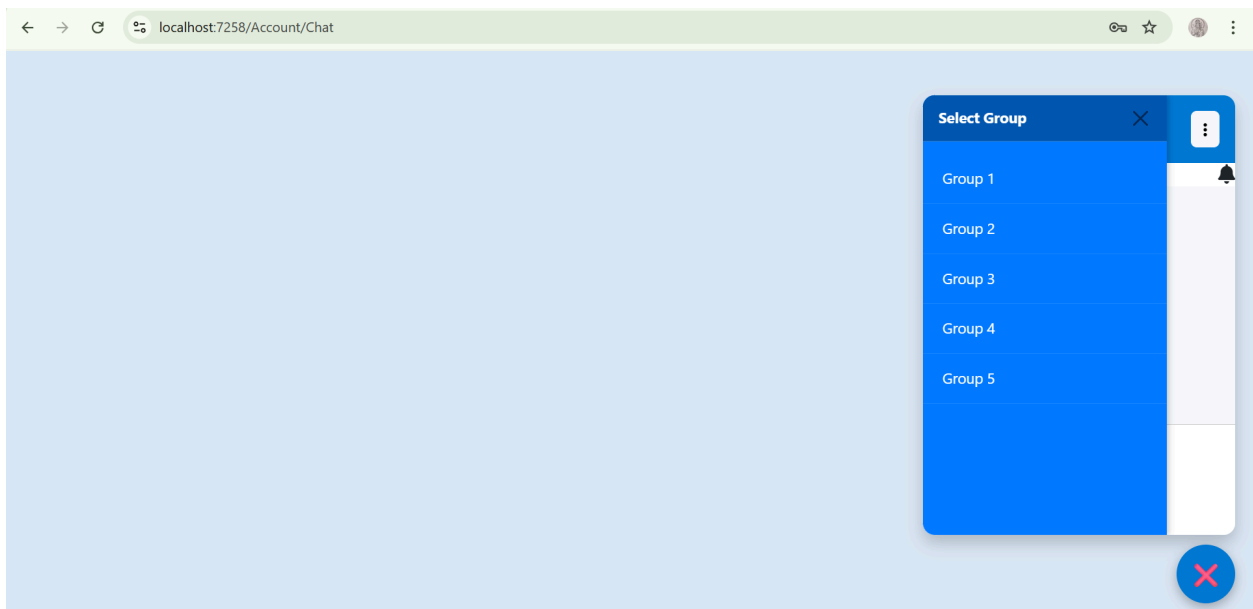
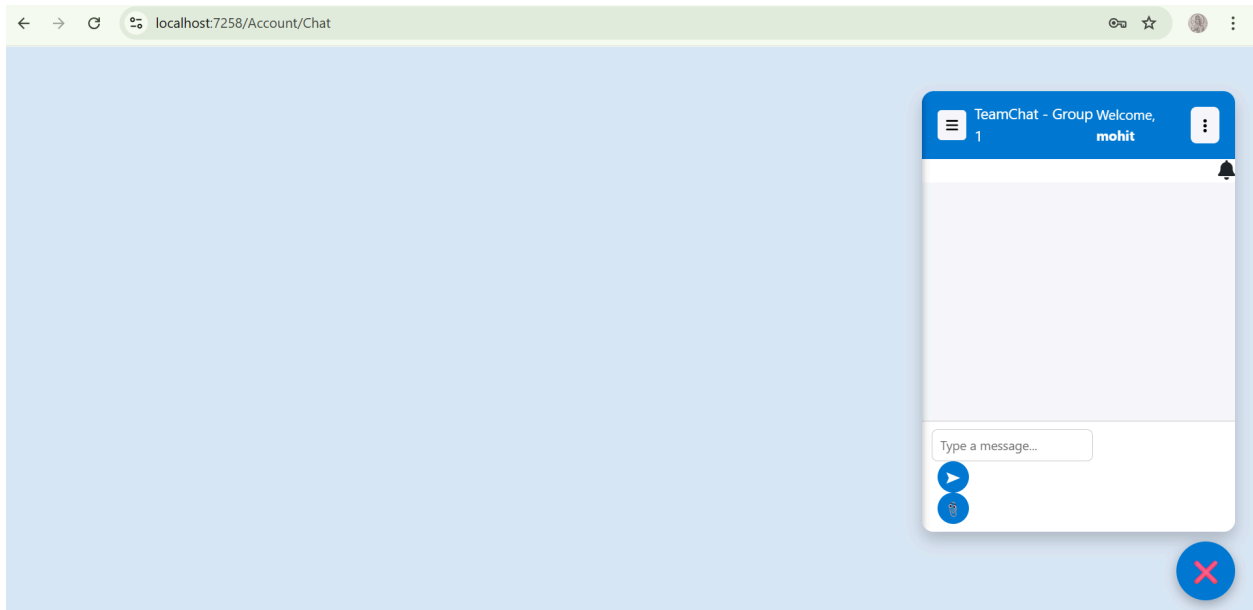


A screenshot of a web browser displaying the TPCODL TeamChat Login page. The browser's address bar shows "localhost:7258/Account/Login". The page has a light blue background. In the center, there is a white rounded rectangle containing the title "TPCODL TeamChat Login" in bold blue text. Below the title are two input fields: "Enter Username" and "Enter Password", both with rounded ends. A blue "Login" button is positioned below the password field. Underneath the button is a blue hyperlink: <c:\training\chatapp\views\account\login.cshtml>. At the bottom of the white box, there is a small line of text: "Use your official TeamChat credentials to continue".

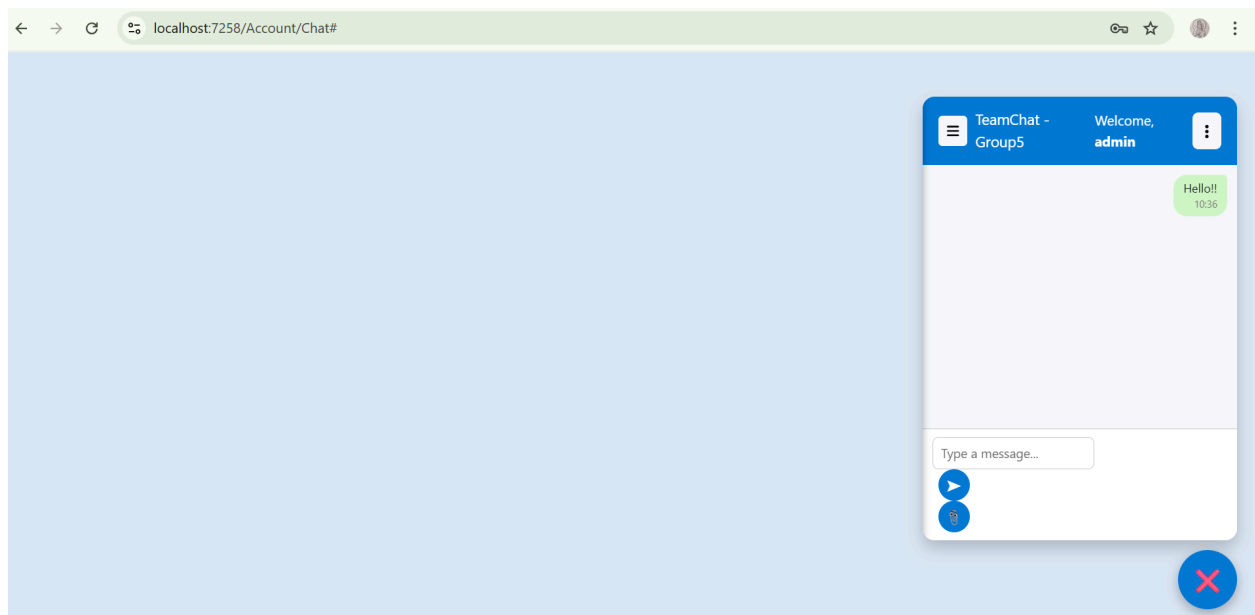
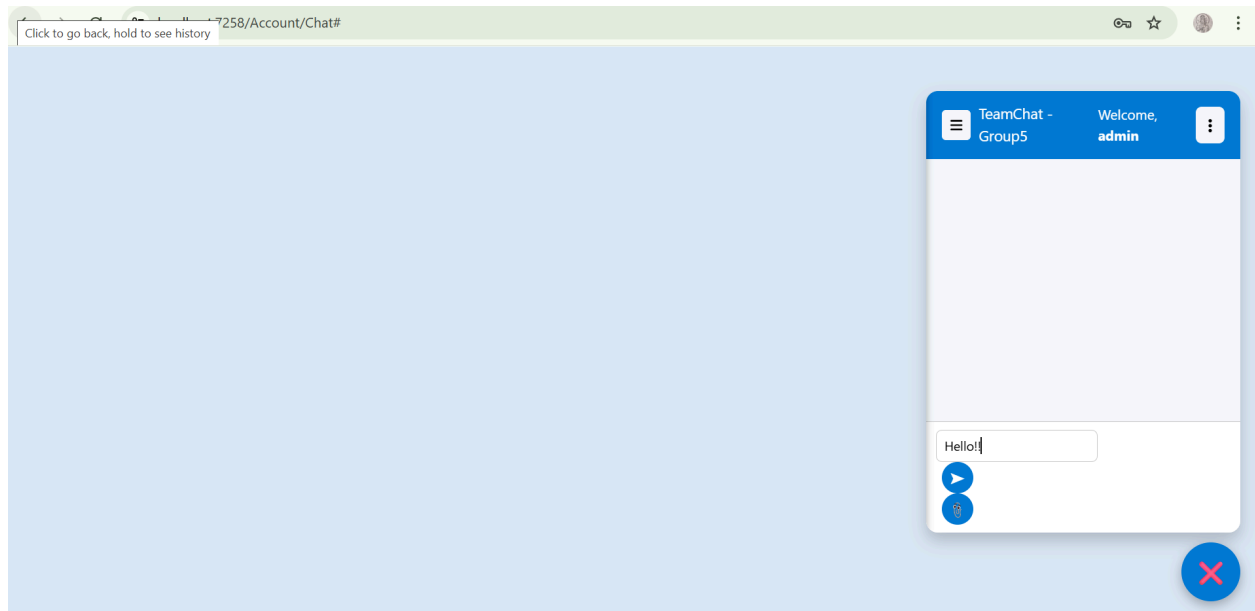


A second screenshot of the same TPCODL TeamChat Login page. The browser address bar remains "localhost:7258/Account/Login". The "Enter Username" field now contains the text "mohit". The "Enter Password" field is empty, showing only the placeholder dots. The blue "Login" button and the blue hyperlink <c:\training\chatapp\views\account\login.cshtml> are still present. The text "Use your official TeamChat credentials to continue" is at the bottom of the white login box.

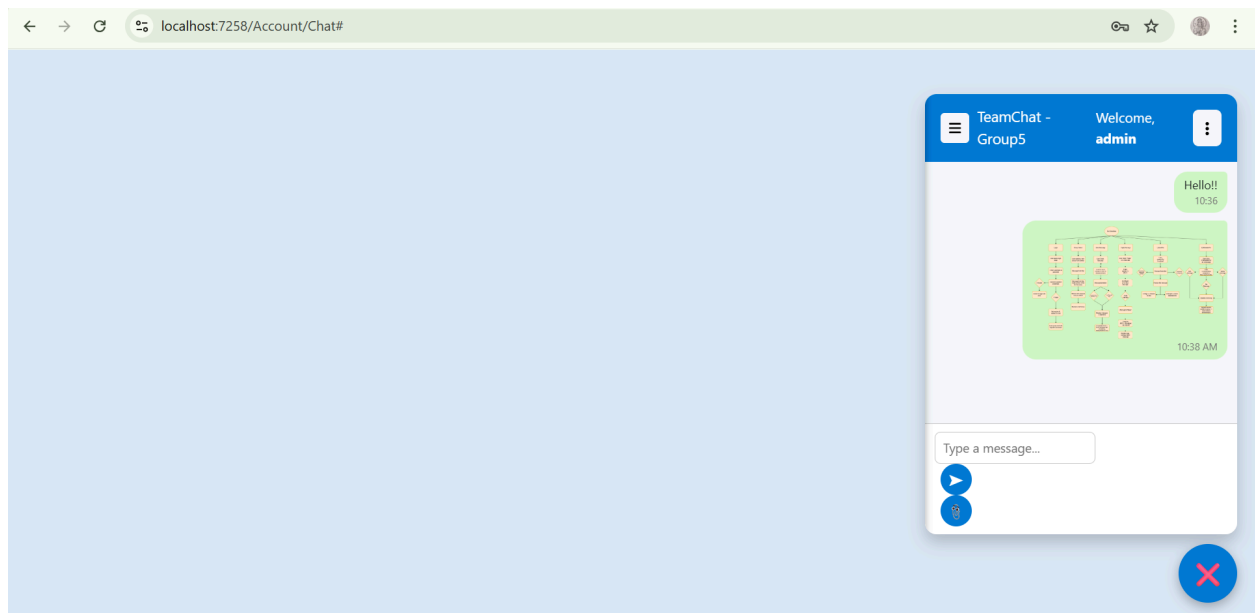
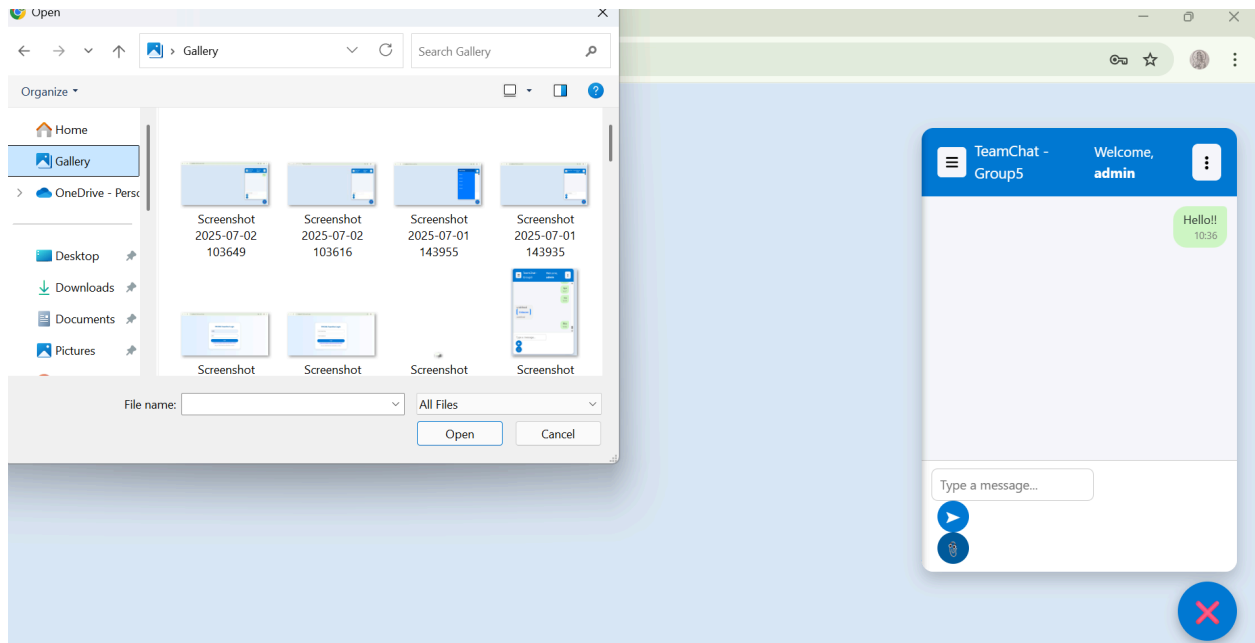
Group Select:



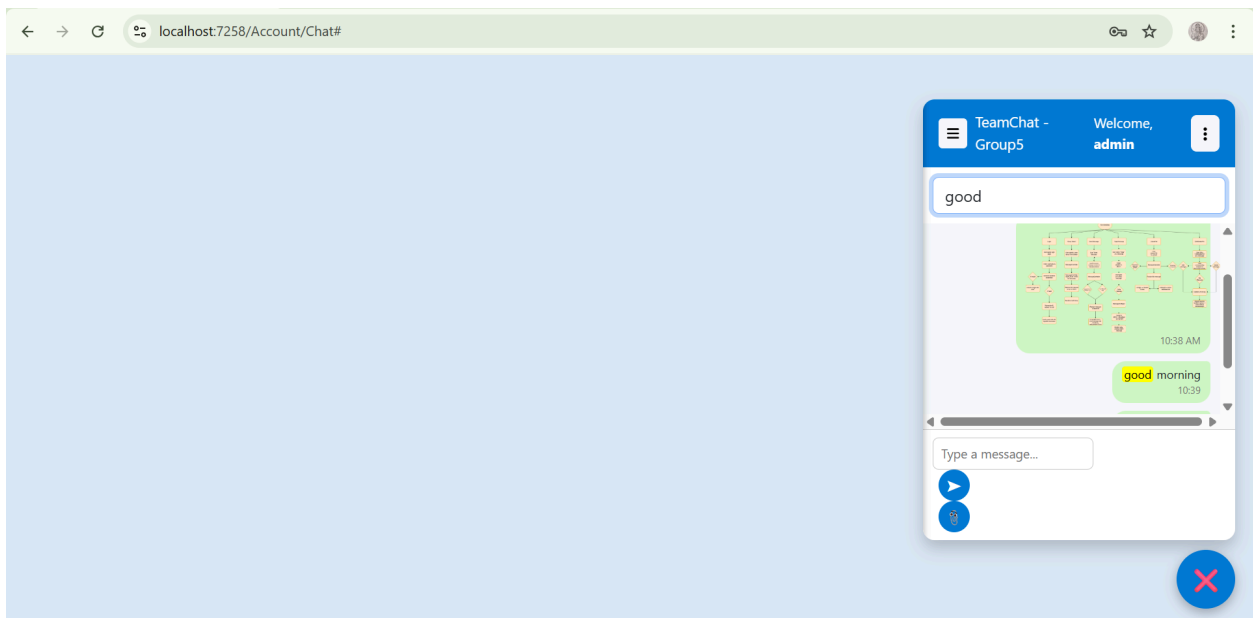
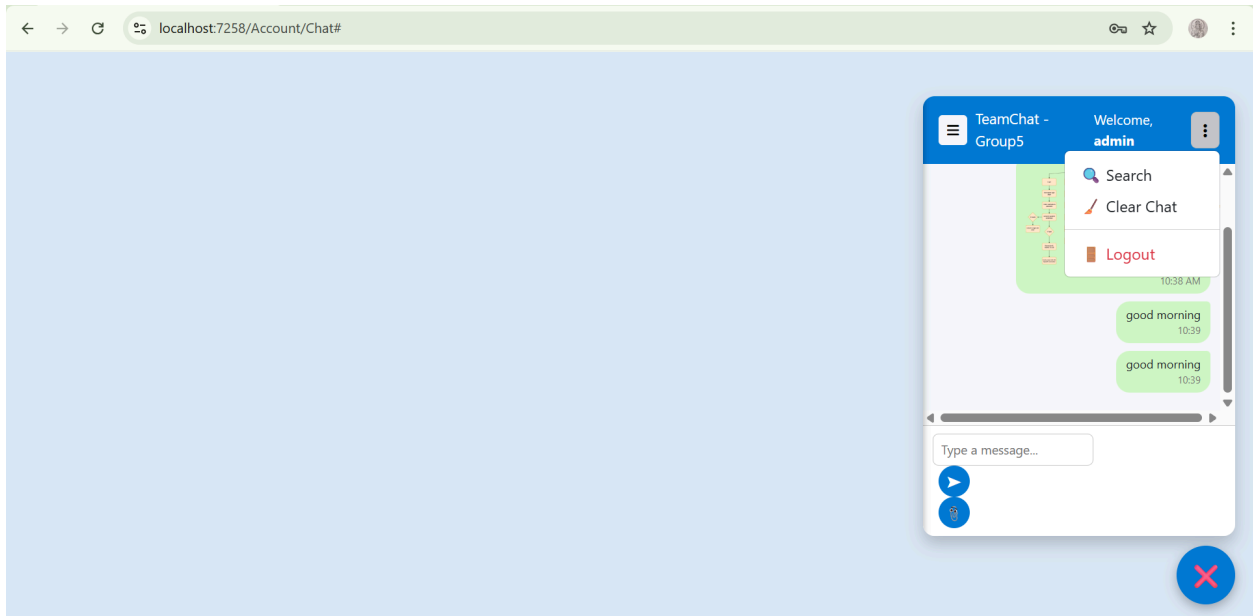
Send Message:



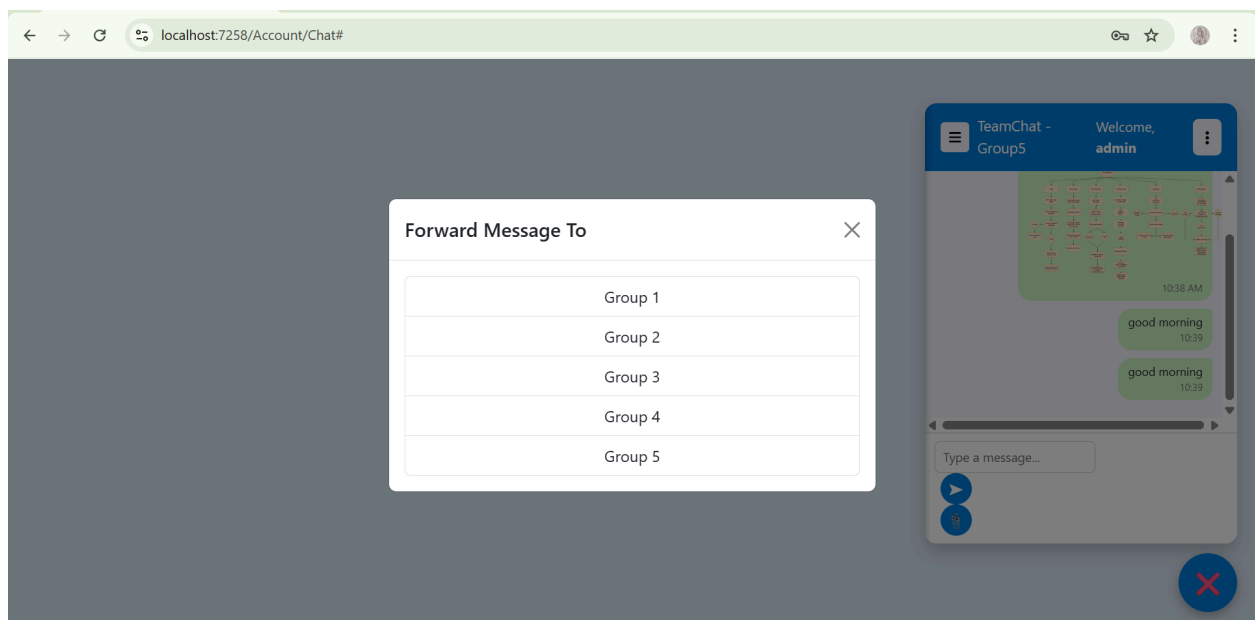
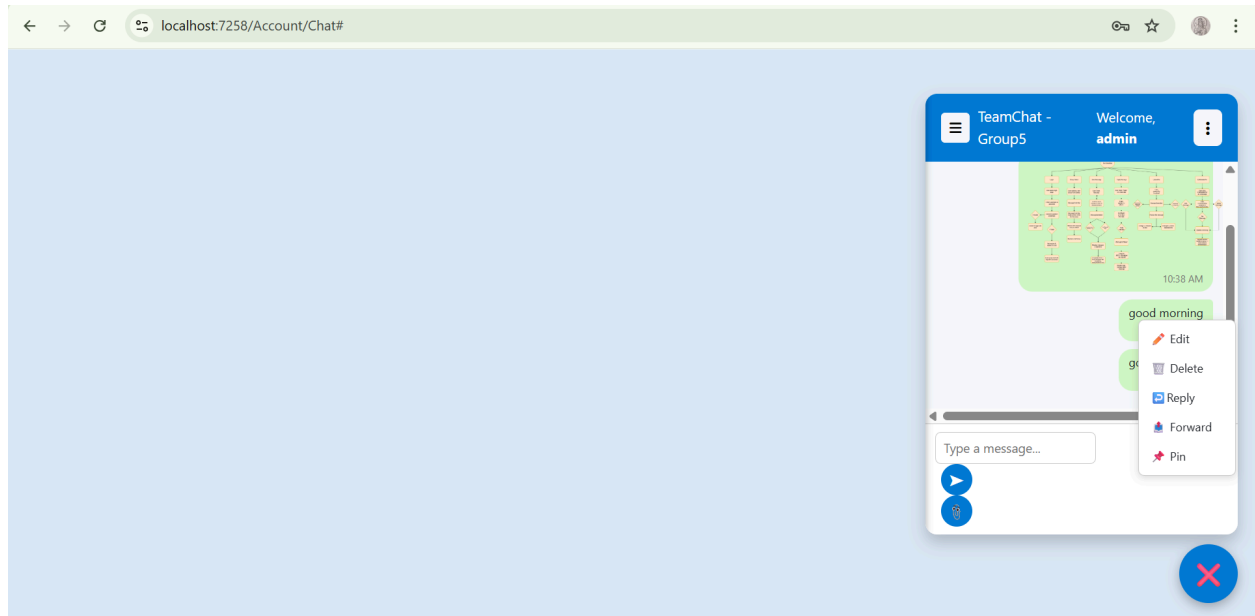
Upload photo/document:

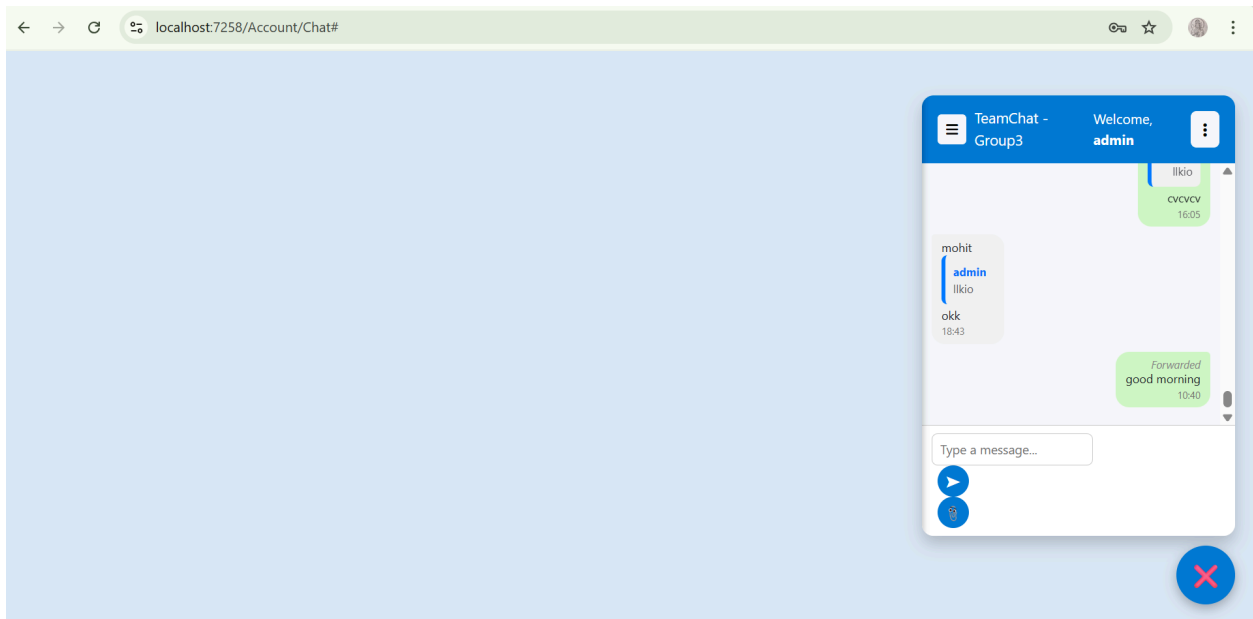


Search Message:

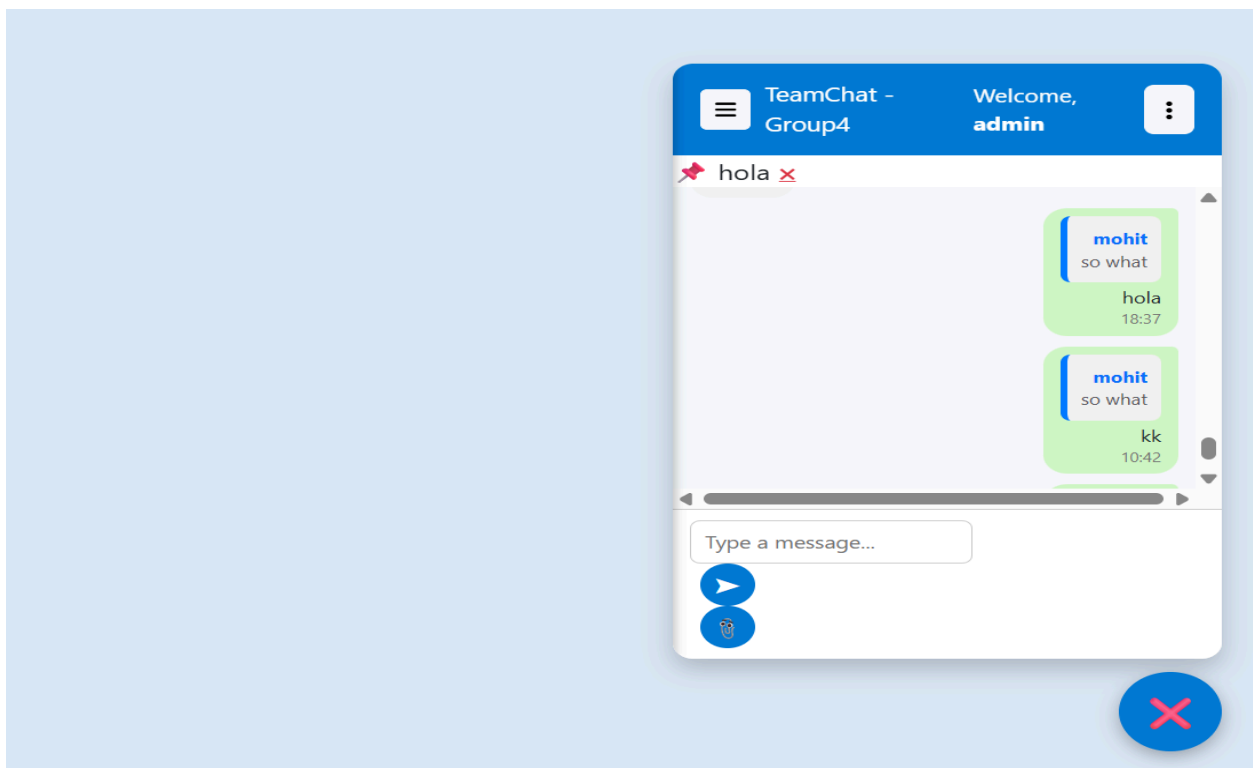
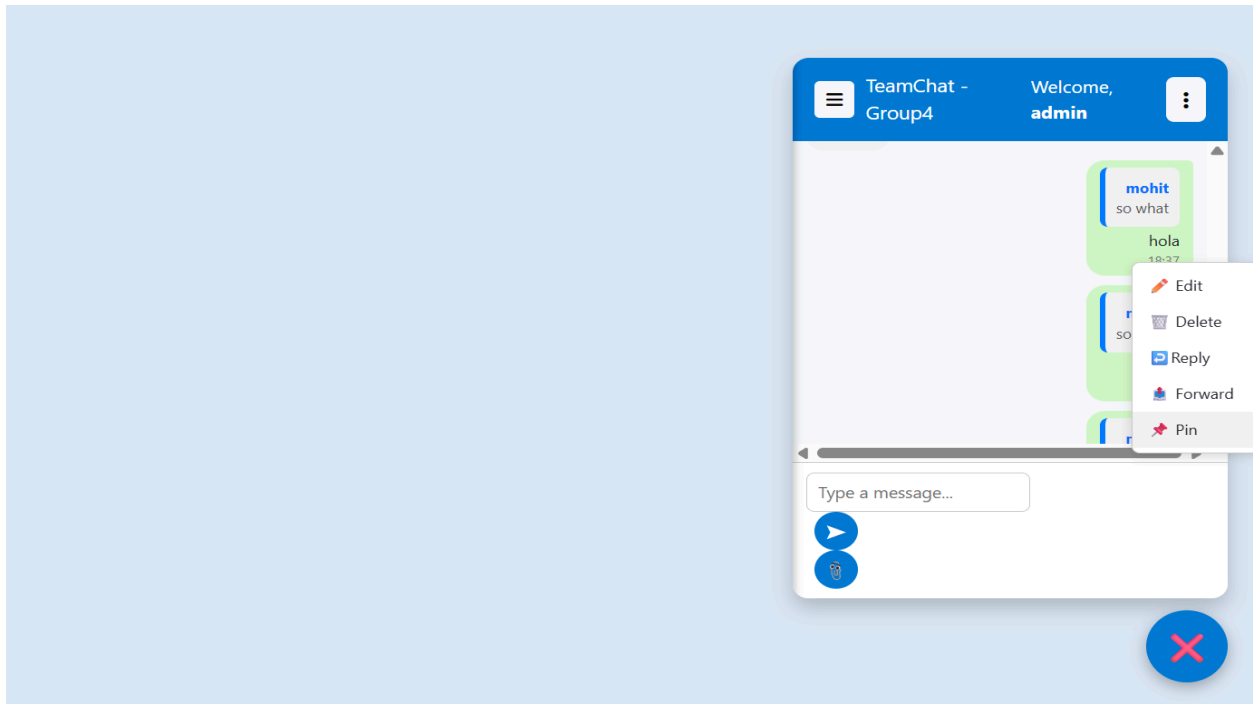


Forward Message:

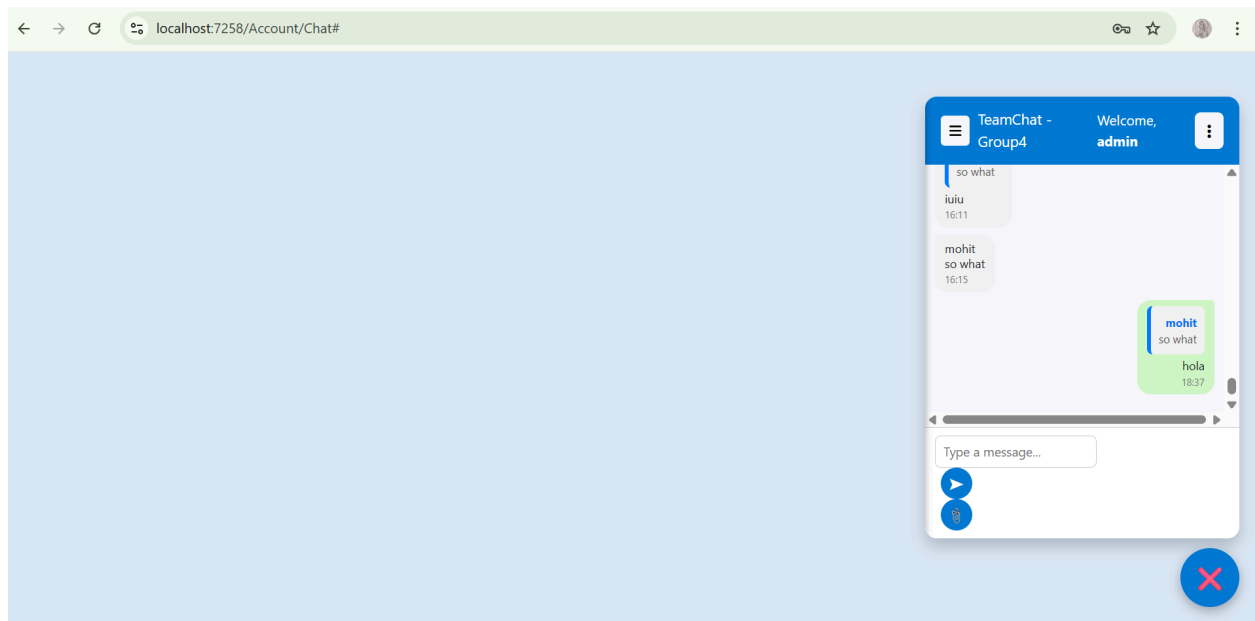




Pin Message:



Reply Message:



Edit Message:

