Title: Chatterbox: A Real-time WebSocket Chat Application

Introduction

Chatterbox is a real-time web-based chat application developed using FastAPI and WebSockets. The primary purpose of this project is to demonstrate real-time bidirectional communication between clients and a server using persistent WebSocket connections. Unlike traditional HTTP communication, which operates on a request-response model, WebSockets maintain an open connection that allows instant message delivery without page refresh.

The application supports multiple users communicating simultaneously with features similar to modern chat platforms.

Objective of the Project

The objective of this project is to design and implement a fully functional real-time chat system that enables multiple authenticated users to send and receive messages instantly. The system ensures persistent storage of chat history, real-time updates, and enhanced user interaction features such as editing, deleting, typing indicators, online user tracking, emojis, date separators, and read receipts.

Technologies Used

The backend is developed using FastAPI, a modern asynchronous Python web framework. WebSockets are used to enable real-time communication between the client and server. SQLite is used as the database for storing chat messages and metadata. The frontend is developed using HTML, CSS, and JavaScript to create an interactive and responsive user interface.

Token-based authentication is implemented to ensure secure access to the chat system.

System Architecture

The system follows a client-server architecture. It consists of three primary components: the frontend interface, the FastAPI backend server, and the SQLite database.

When a user enters a username and logs in, a POST request is sent to the backend login endpoint. The backend generates a token for authentication and returns it to the client. Using this token, the client establishes a WebSocket connection with the server.

Once connected, the server registers the user in an active connections dictionary. A join message is broadcast to all users, and the online users list is updated in real time. The server continuously listens for incoming WebSocket messages and processes them accordingly.

FastAPI Backend Server module

The **FastAPI Backend Server module** forms the core of the application. It is responsible for creating and managing the WebSocket endpoint that enables real-time, bidirectional communication between clients and the server. This module handles incoming HTTP requests such as login and serves the frontend interface. It also manages the lifecycle of WebSocket connections, including accepting connections, receiving messages,

and sending responses. Since FastAPI supports asynchronous programming, the server efficiently handles multiple concurrent users without blocking other connections.

WebSocket Manager Module

The **WebSocket Manager Module** contains the logic required to manage multiple active users at the same time. It maintains a collection of active WebSocket connections and provides methods to connect a user, disconnect a user, and broadcast messages to all connected clients. This module enables real-time message delivery by sending incoming messages from one client to all other clients instantly. It follows a publish-subscribe communication pattern, where every connected client subscribes to message broadcasts from the server.

Authentication & Database Module

The **Authentication & Database Module** ensures secure access and data persistence within the system. Authentication is handled using token-based verification, where users receive a token upon login and must present it when connecting to the WebSocket endpoint.

The database component uses SQLite to store user information and chat messages. This module manages saving new messages, updating edited messages, deleting messages, and retrieving chat history when a user joins the chat. By integrating persistence, the application maintains message continuity even after users disconnect or refresh their browser.

Client Module

The **Client Module** represents the user-facing interface of the system. Initially described as a terminal-based client in the project outline, it has been implemented as a web-based frontend using HTML, CSS, and JavaScript. This module allows users to log in, send messages, receive real-time updates, and view chat history. It also handles advanced features such as typing indicators, message editing and deletion, date separators, emojis, and read receipts. The client communicates with the backend using WebSocket messages and processes special control messages such as typing status, seen notifications, and system join/leave alerts.

Features Implemented

1.Real-Time Messaging

Real-time communication is achieved using WebSockets. After accepting the connection, the server sends chat history to the newly connected user. The server then enters a continuous loop to receive messages.

When a user sends a normal chat message, the server generates a timestamp and date, stores the message in the database, and broadcasts it to all connected clients. Since WebSockets maintain persistent connections, messages appear instantly without refreshing the page.

2.Chat History Storage

All messages are stored in a SQLite database. The database schema includes message ID, username, message text, timestamp, date, and edited status.

When a new user connects to the chat, the complete chat history is retrieved from the database and sent in a structured format. This ensures that previous conversations are visible immediately after login.

3.Message Editing Feature

The system supports editing previously sent messages. When a user edits a message, a control message with a specific prefix is sent to the server. The backend identifies the edit request, updates the message in the database, and broadcasts the updated message to all connected users.

The frontend updates the message content dynamically and displays an "(edited)" indicator next to the message.

4.Message Deletion Feature

Message deletion is also supported. When a delete request is sent, the server removes the message from the database and broadcasts a delete event. The frontend replaces the original message with a placeholder text indicating that the message was deleted.

The deletion update occurs instantly across all connected clients without requiring a page refresh.

5.Online Users Tracking

Active users are tracked using a dictionary structure in the ConnectionManager class. When a user connects, they are added to this structure. When a user disconnects, they are removed.

A broadcast function sends the updated list of online users to all connected clients, ensuring the online users panel is dynamically updated.

6.Typing Indicator

Typing indicators are implemented using special control messages. When a user starts typing, a typing event is broadcast to other users. When typing stops, a stop event is sent.

This provides real-time feedback, displaying a message such as "User is typing..." in the chat interface.

7.Emoji Support

An emoji panel is integrated into the chat interface. Users can select emojis, which are appended to the message input field. This enhances user interaction and improves the overall chat experience.

8.Date Separators

The application dynamically inserts date separators such as "Today" and "Yesterday" when rendering messages. The date of each message is compared to the current date to determine the appropriate label. This improves readability and conversation organization.

9.Delivery and Read Receipts

The application implements delivery and read receipt indicators similar to professional messaging platforms.

When a message is sent, an empty circle symbol is displayed to indicate that the message has been delivered. When the recipient views the message, a seen event is triggered and broadcast to the sender. The empty circle is then updated to a filled circle symbol.

This feature enhances communication transparency by informing users about message status.

Work Flow

Login Flow

When the user enters a username, the login API is called and a unique authentication token is generated. Using this token, a secure WebSocket connection is established between client and server. After successful

connection, the user is added to the active connections list. The server then broadcasts that the user joined and sends previous chat history to that user.

Message Flow

When a user sends a message, it is transmitted through WebSocket to the server. The server saves the message in the SQLite database along with timestamp and date. Then the server broadcasts the formatted message to all connected users. The receiver renders the message in the UI and sends a seen event, which updates the sender's status indicator.

Edit Flow

When the user clicks edit, the client sends a special edit request containing message ID and updated text. The server updates the corresponding message in the database. After updating, the server broadcasts the edit event to all connected users. All clients update that specific message in real-time and display the edited label.

Delete Flow

When the user clicks delete, a delete request with the message ID is sent to the server. The server updates the database by marking the message as deleted. A delete event is then broadcast to all connected users. Every client immediately replaces the original message text with "This message was deleted" in the chat UI.

Exception Handling and Stability

WebSocketDisconnect exceptions are handled to prevent server crashes. When a user disconnects unexpectedly, the system removes the user from active connections and broadcasts a leave message. Proper exception handling ensures system stability and smooth real-time operation.

Final Output

The final output is a fully functional real-time chat application supporting multiple concurrent users. Messages are delivered instantly, chat history is stored permanently, and dynamic features such as editing, deleting, typing indicators, online users tracking, emojis, date separators, and read receipts are successfully implemented.

The project demonstrates practical implementation of WebSockets, asynchronous programming, real-time broadcasting, and database integration within a modern web application framework.