Chatterbox: A Real-Time WebSocket Chat Application

1. Project Overview

Chatterbox is a real-time, multi-user chat application built using **FastAPI and WebSockets**.

Unlike traditional HTTP request-response systems, this application uses persistent WebSocket connections to enable **bi-directional, real-time communication** between the server and multiple connected clients.

The system supports:

- User registration & login
- Token-based authentication
- Real-time message broadcasting
- Chat history persistence
- Modern frontend chat interface
- Message differentiation (Me vs Others)

---

2. Problem Statement

Traditional web applications operate using a request-response model. This model is inefficient for real-time applications such as chat systems because:

- It requires repeated polling.
- It increases server load.
- It does not support instant message broadcasting efficiently.

This project solves the above problem by:

- Using WebSockets for persistent connections.
- Implementing asynchronous FastAPI backend.
- Broadcasting messages to all connected clients instantly.

---

3. Objectives

- Build an asynchronous FastAPI backend.
- Implement WebSocket-based real-time messaging.

- Develop secure user authentication system.
- Store chat history in SQLite database.
- Design a user-friendly frontend interface.
- Implement message differentiation:
    - Sender messages on right.
    - Other users' messages on left.

---

4. System Architecture

🏗️ High-Level Architecture

Frontend (HTML + JS)

↓

REST API (/register, /login)

↓

FastAPI Backend

↓

WebSocket (/ws?token=...)

↓

Connection Manager

↓

SQLite Database

---

5. Modules Implemented

5.1 FastAPI Backend Server

- Handles HTTP endpoints:
    - /register
    - /login
- WebSocket endpoint:
    - /ws

- Uses asynchronous event loop.
- Manages concurrency efficiently.

5.2 WebSocket Manager Module

Responsible for:

- Accepting new connections.
- Maintaining list of active connections.
- Broadcasting messages.
- Handling disconnections.

Core functions:

- connect()
- disconnect()
- broadcast()

5.3 Authentication & Database Module

- Password hashing using bcrypt.
- Token-based authentication (JWT).
- SQLite database for:
    - User storage
    - Message history storage

Tables:

Users Table

- id
- username
- hashed_password

Messages Table

- id
- username

- message
- timestamp

5.4 Frontend Module

Frontend consists of:

index.html → Login / Register page / Chat interface

Features:

- Token stored in localStorage.
- Redirect logic after login.
- WebSocket connection using token.
- UI differentiation:
    - Me → Right side (blue bubble)
    - Others → Left side (gray bubble)

---

6. Features Implemented

✅ **User Authentication**

- Register new users.
- Secure login.
- JWT-based session management.

✅ **Real-Time Messaging**

- WebSocket persistent connection.
- Instant broadcast to all users.
- Asynchronous handling.

✅ **Chat History Persistence**

- Messages saved in SQLite.
- New user receives previous messages.

✅ **UI Message Differentiation**

- Current user messages aligned right.

- Other users aligned left.
- Clean responsive layout.

✅ **Multi-User Support**

- Multiple clients can connect simultaneously.
- Incognito testing supported.

---

7. API Endpoints

POST /register

Registers a new user.

Request:

{

"username": "user1",

"password": "1234"

}

Response:

{

"message": "User registered successfully"

}

POST /login

Authenticates user and returns token.

Response:

{

"access_token": "jwt_token_here"

}

WebSocket /ws?token=...

- Validates JWT token.

- Establishes persistent connection.
- Enables message exchange.

---

8. How to Run the Project

Step 1 — Backend

cd server

uvicorn server:app –reload

Runs on:

http://127.0.0.1:8000

Step 2 — Frontend

Inside frontend folder:

python -m http.server 5500

Open:

http://127.0.0.1:5500/index.html

---

9. Technologies Used

- Python 3.14.0
- FastAPI
- WebSockets
- SQLite
- bcrypt
- JWT
- HTML
- JavaScript
- CSS

---

10. Advantages

- Highly scalable architecture.

- Asynchronous and efficient.
- Real-time communication.
- Secure authentication.
- Clean modular structure.

---

11. Limitations

- SQLite limits very high-scale deployment.
- No message encryption beyond transport.
- No file/image sharing yet.
- No private messaging feature.

---

12. Future Enhancements

- Typing indicators.
- Private chat rooms.
- Message encryption.
- Deployment on cloud.
- React-based frontend.
- Docker containerization.

---

13. Conclusion

Chatterbox successfully demonstrates:

- Real-time WebSocket communication.
- Concurrent multi-user handling.
- Secure authentication.
- Persistent storage.
- Clean frontend-backend separation.

The project meets all requirements and serves as a scalable foundation for production-level real-time communication systems.