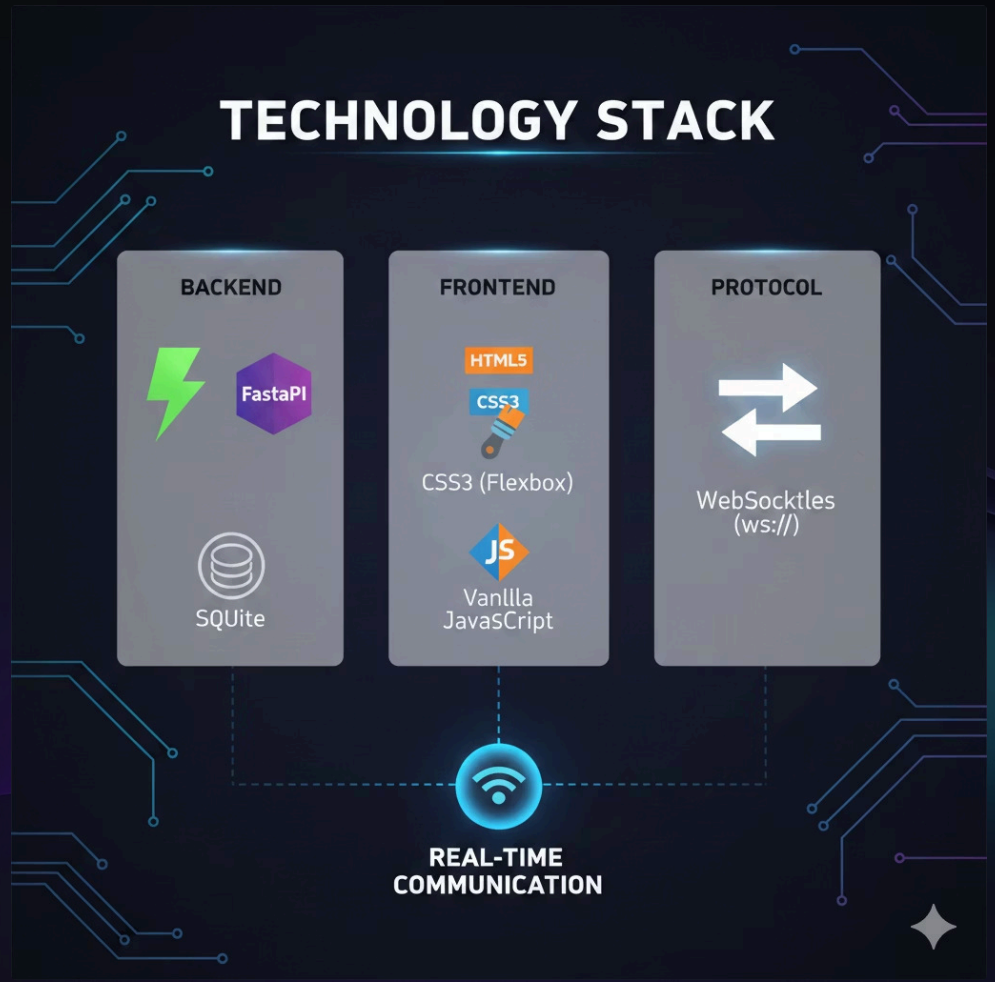
Chatterbox:

A Real-time WebSocket Chat Application

Presented by: POORNIMA H U

Project Overview

Chatterbox is a full-stack real-time chat system built with FastAPI and WebSockets. Objectives: - Zero-latency two-way messaging, Accurate user presence and typing indicators, Persistent message storage and retrieval and amodern responsive UI.



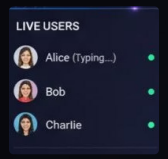
Technical Stack

- Backend: FastAPI — WebSocket routing & REST endpoints
- Database: SQLite — durable relational storage for messages
- Frontend: HTML5 + CSS3 (Flexbox) + Vanilla JS — lightweight, dependency-free
- Protocol: WebSockets (ws://) — full-duplex, low latency

Key Features

Message CRUD

Edit, delete (placeholder replacement), and persist messages with edit flag and timestamps.



Presence & Typing

Broadcast online status, typing indicators, and reliable join/leave events.

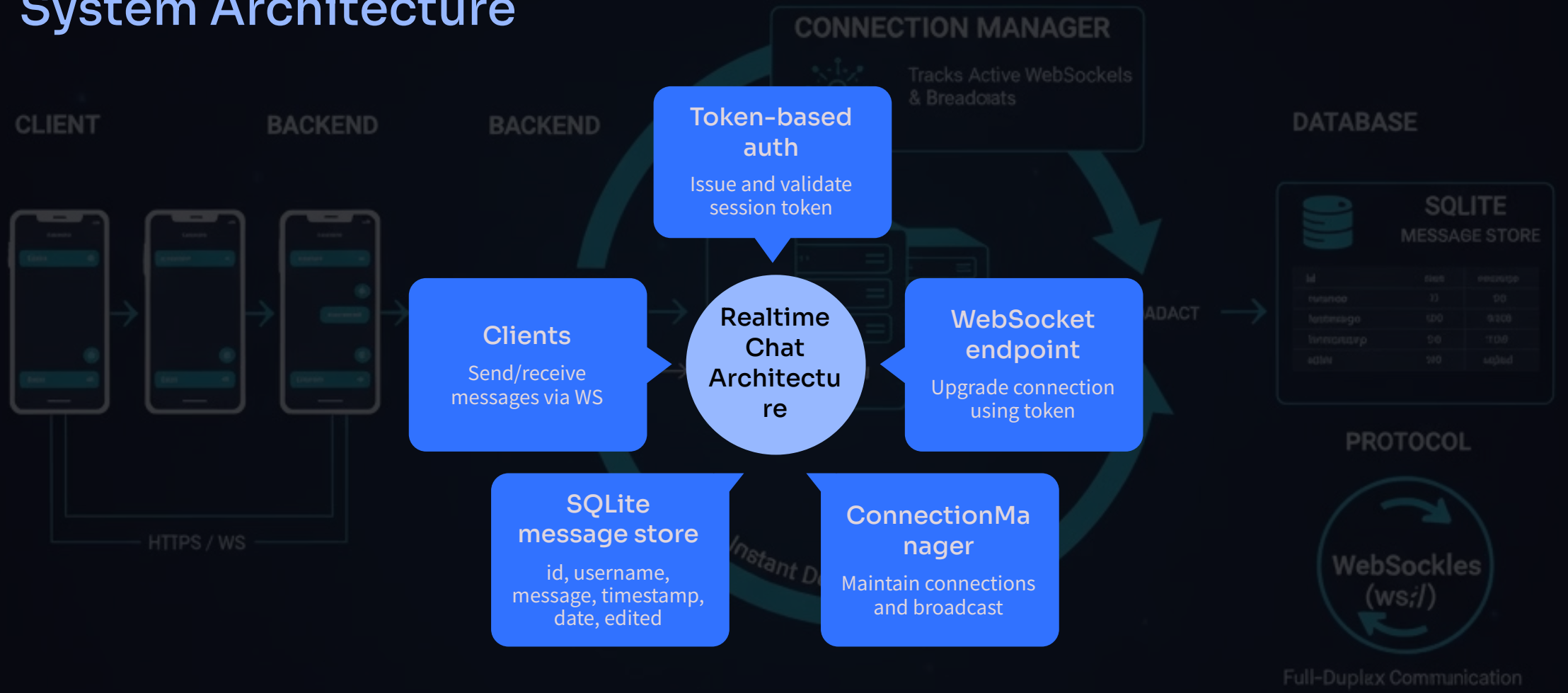
Read Receipts

Delivered vs seen state: empty circle = delivered, filled = seen; synchronized across clients.

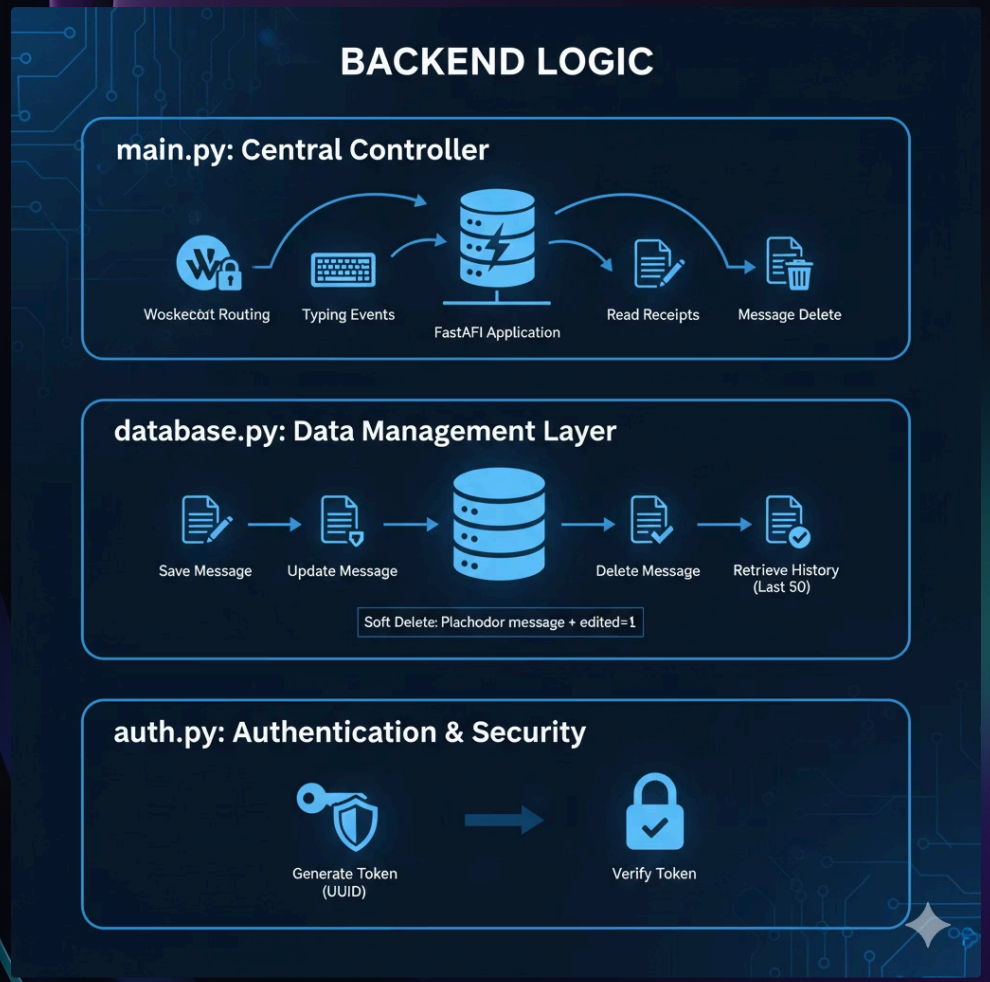
UX Polishes

Emoji support, date grouping (Today / Yesterday), Enter to send, and accessible keyboard interactions.

System Architecture



ConnectionManager centralizes connection state and broadcasting. Authentication issues a session token validated on WebSocket upgrade. SQLite provides durable history retrieval for new joiners.



Backend Logic

main.py

Routes WebSocket connections, parses events (typing, edit, delete, seen), delegates to ConnectionManager and DB layer.

database.py

CRUD: save, update (edit flag), delete (placeholder), and paginate retrieval by date for efficient joins.

auth.py

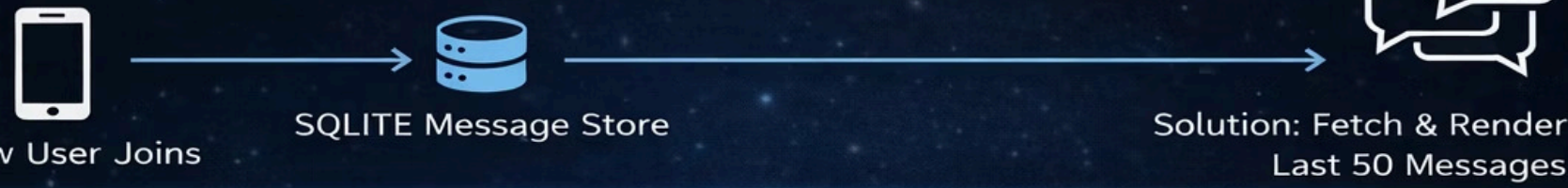
Generates/verifies session tokens used for WebSocket auth and simple session management.

Frontend Interface

- Login: username input → token created → WebSocket open.
- Rendering: left/right bubbles, date separators, emoji picker, dynamic user list.
- Events: keydown (Enter to send), WebSocket message handlers for broadcasted events, optimistic UI updates.
- Accessibility: semantic HTML, focus management for inputs and emoji control.

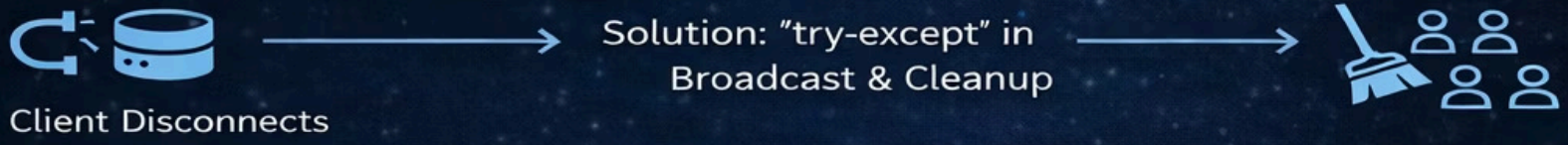
CHALLENGES AND SOLUTIONS

HISTORY SYNC



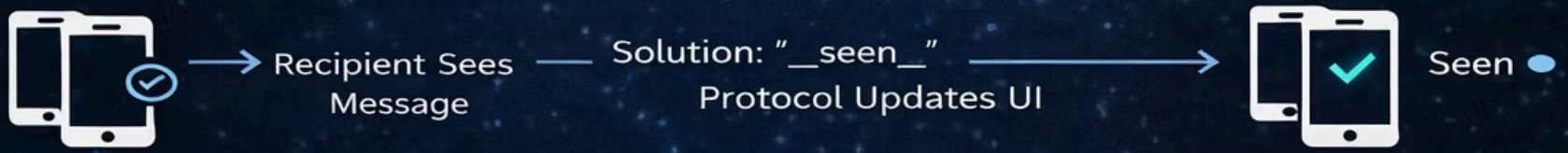
> On join, server streams stored messages grouped by date to ensure consistent timeline and ordering.

DISCONNECTION HANDLING

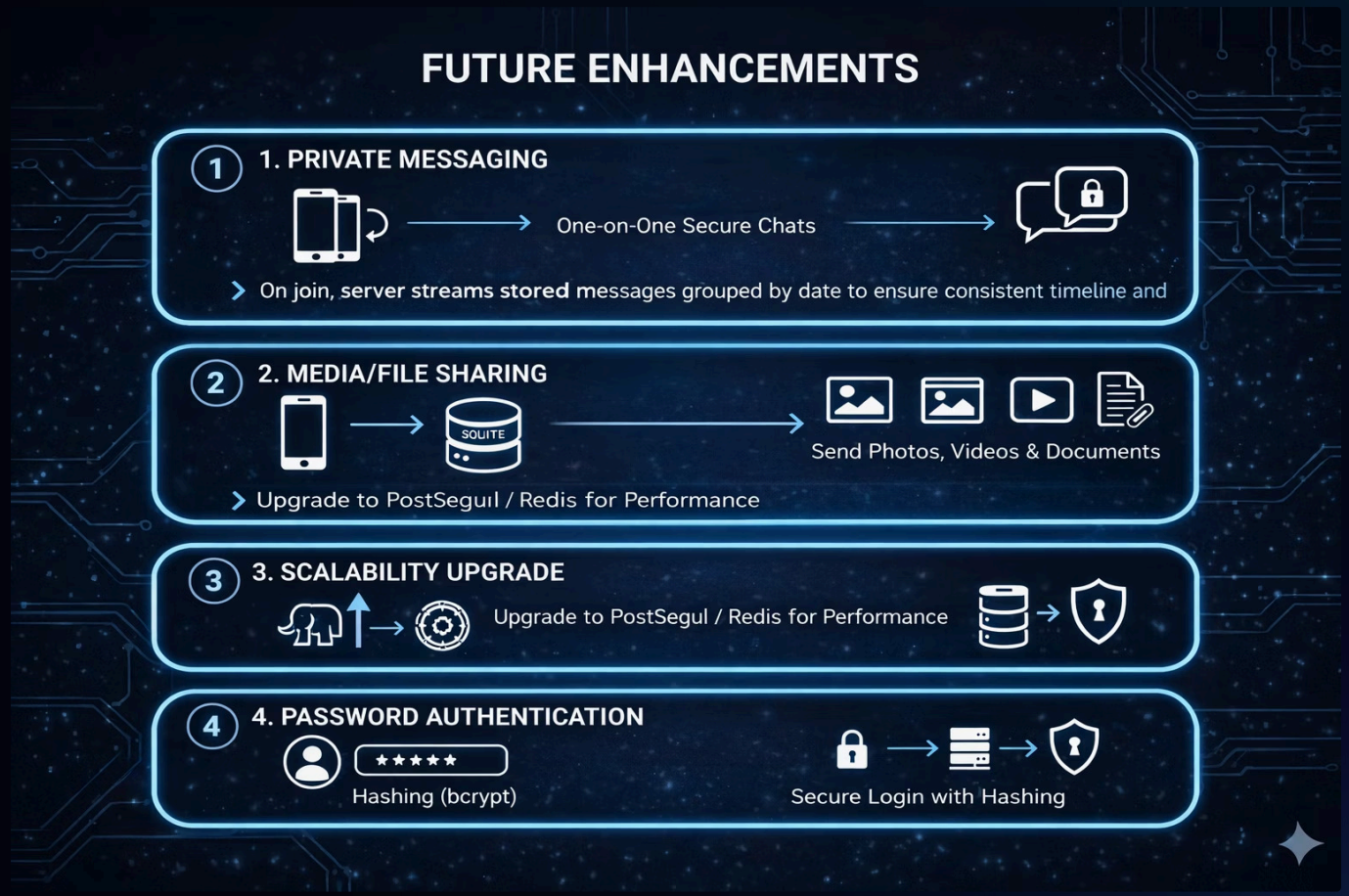


> Use WebSocketDisconnect events to remove stale connections; heartbeat/keepalive optional for robustness.

ACCURATE READ RECEIPTS



> Clients send explicit 'seen' events when message enters viewport; server validates before broadcasting seen state to avoid false positives.



Future Enhancements

- Private/one-to-one messaging and room permissions.
- Media/file attachments with CDN and virus scanning pipeline.
- Scalability: replace SQLite with PostgreSQL or Redis (pub/sub) for horizontal scaling.
- Authentication: password hashing, OAuth integration, refresh tokens.

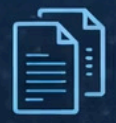
CONCLUSION

CHATTERBOX: A REAL-TIME COMMUNICATION SYSTEM

Key Features

Demonstrates Real-Time Communication
WebSockets & Persistent Storage

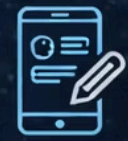
Modular Backend Architecture
main.py, database.py, auth.py



Scalability & Future

Scalability Potential
Database Upgrades (PosTOGOQL/Redis)

Interactive UI Features
Typing, Read Receipts, Edit/Delete



THANK YOU

ANY QUESTIONS? ?