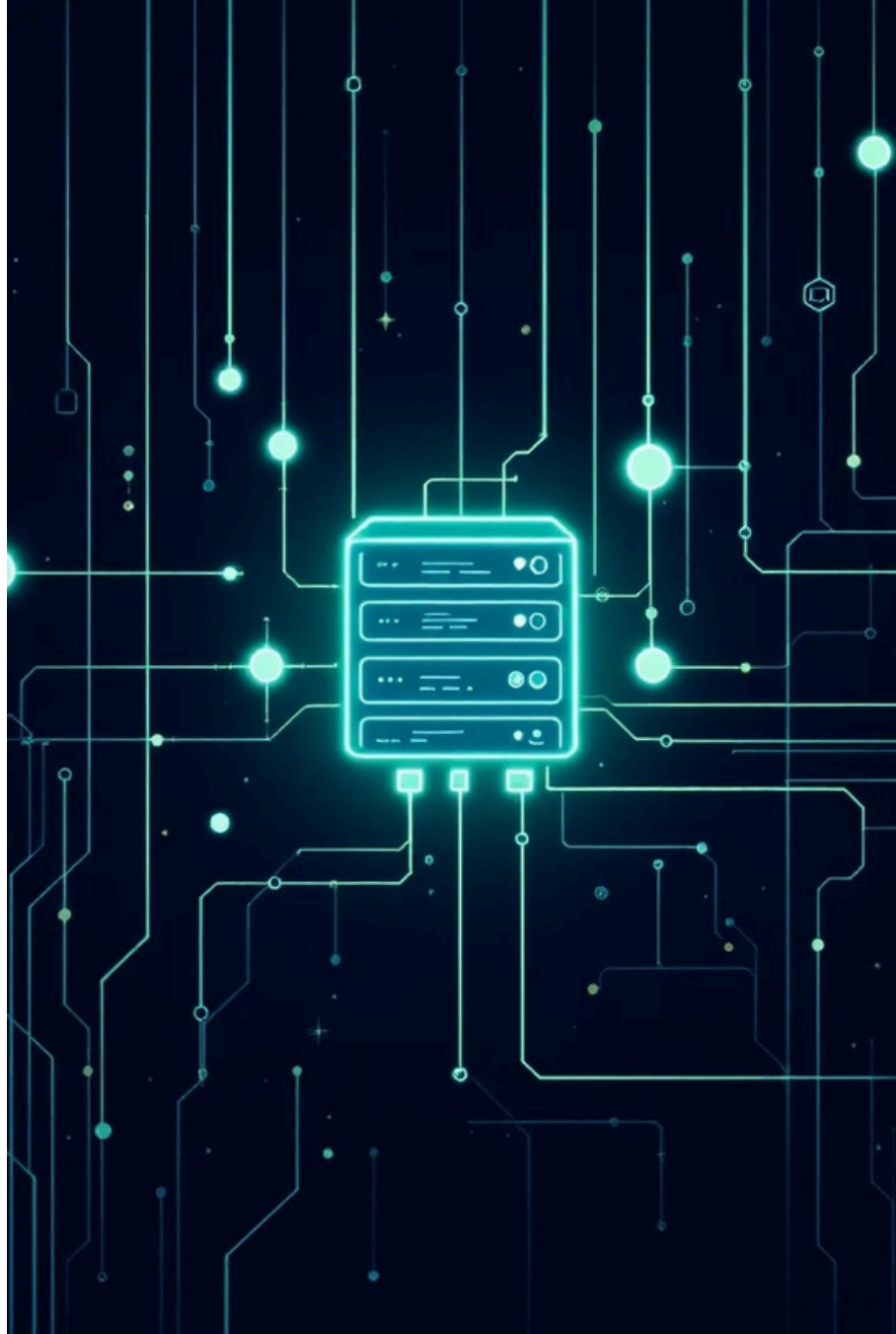
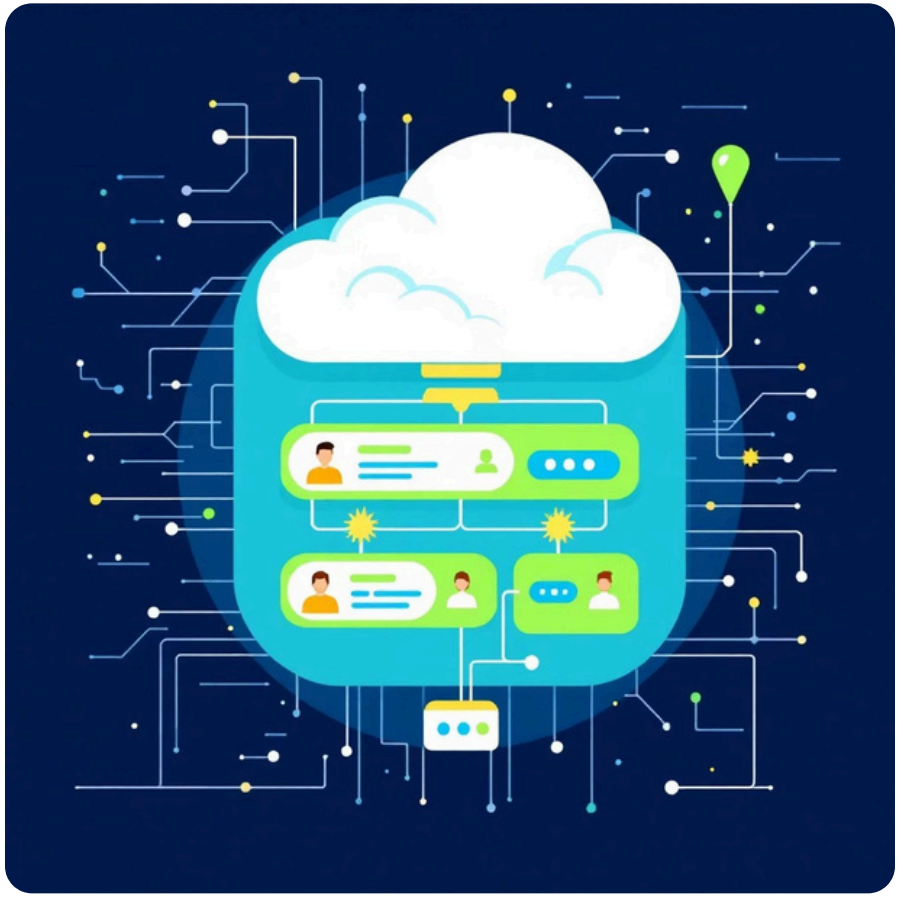
Chatterbox: Real-time WebSocket Chat Application

Building a high-performance, concurrent chat system with FastAPI and WebSockets



CHALLENGE

The Problem Statement



Real-time applications like chat services and live dashboards require persistent, bi-directional communication between servers and multiple clients. Traditional request-response architectures are ill-suited for this challenge.

WebSockets provide the solution: a protocol designed specifically for persistent connections, enabling truly real-time experiences.

Project Objectives

High Performance

Asynchronous FastAPI server handling thousands of concurrent WebSocket connections

Event-Driven Architecture

Real-time message broadcasting with pub-sub model for dynamic chat rooms

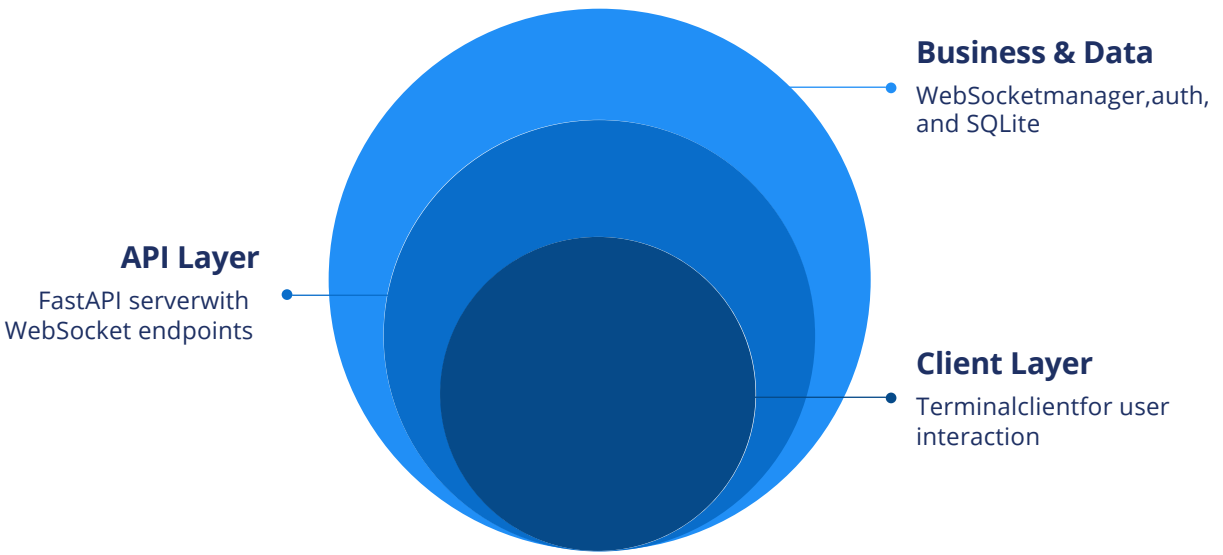
Secure Authentication

User authentication and session management for persistent, secure connections

Data Persistence

Relational database backend storing chat history and user information

System Architecture



Layered Design

Thearchitectureseparates concerns across four distinct layers, ensuring scalability and maintainability.

- Client layer for user interaction
- API layer managing WebSocket connections
- Business logic for broadcasting and auth
- Data persistence with SQLite

Technology Stack

FastAPI

High-performance async web framework for building the server and WebSocket endpoints

WebSockets

Protocol enablingpersistent, bi-directional communication between clients and server

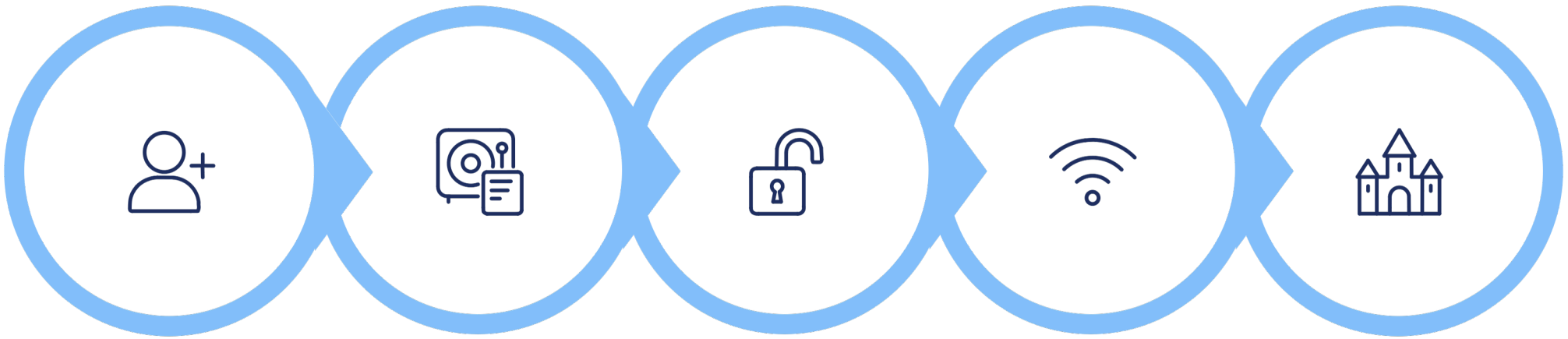
SQLite

Lightweight relational database for storing user data and chat message history

Asyncio

Python's asynchronous I/O library for handling concurrent connections ef ficiently

Authentication Flow



Register **Store** **Login** **Connect** **Chat**

Users must register and authenticate before accessing the chat room. Session tokens ensure secure, persistent connections throughout the chat experience.

WebSocket Message Flow

Broadcasting Architecture

TheWebSocket Managermaintains active connections and broadcasts messages to all participants in real-time.

Each message includes the sender's username and timestamp, creating a shared, synchronised chat experience across all connected clients.

Client Send | **Manager Broadcast** | **Store History** | **Sync New Clients**

Core Modules to Implement

1 2

FastAPI Backend Ser ver

Asynchronous application managing WebSocket endpoints and user authentication

WebSocket Manager

Corelogic for connection handling, message broadcasting, and pub-sub model implementation

3

Authentication & Database

Userregistration, login functionality, and persistent storage of user data and chat history

4

Terminal Client

Python scriptenabling users to connect, send messages, and receive real-time broadcasts

DATA PERSISTENCE

Database Design

TablesUsed

1. Users Table

- `id`(Primary Key)
- `username` (Unique)
- `hashed_password`

Purpose: Stores registered user credentials securely

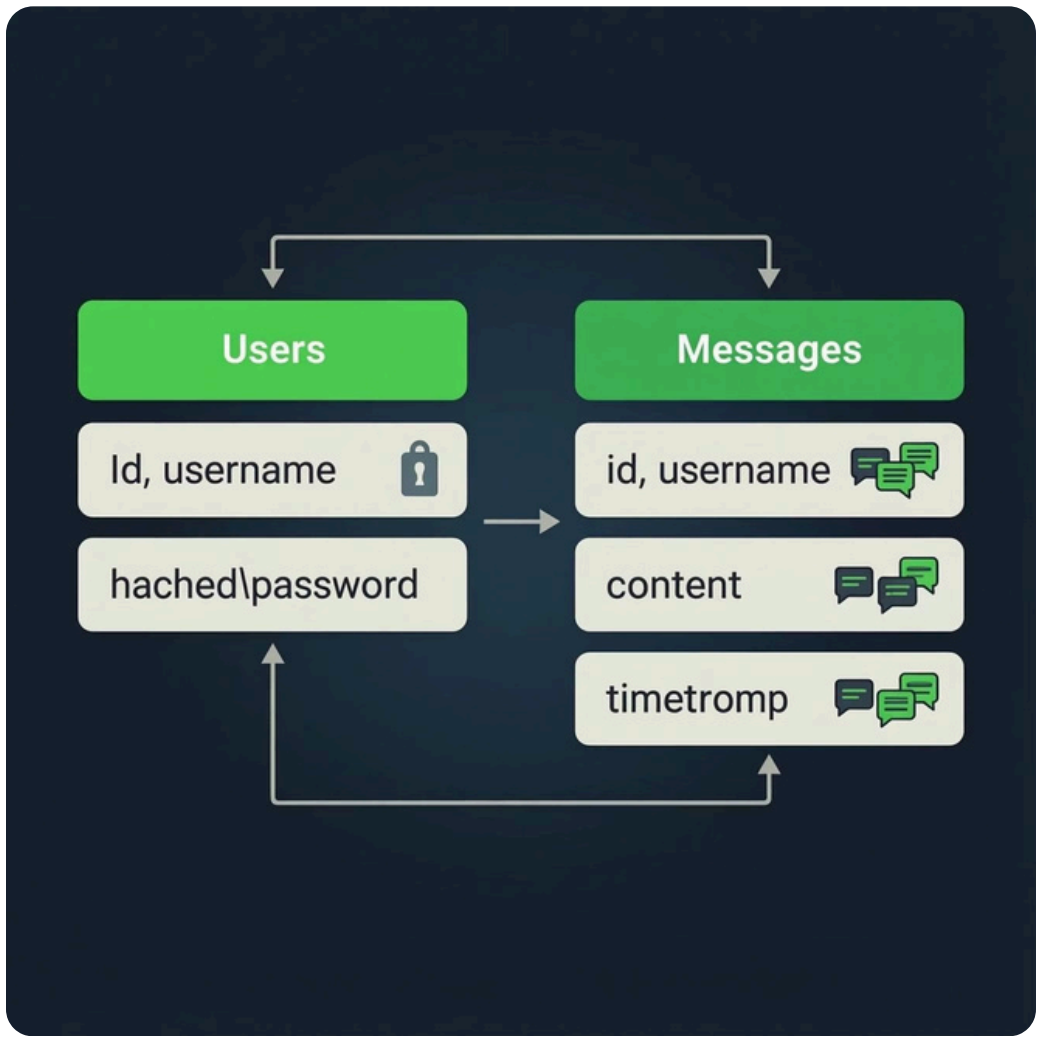
2. Messages Table

- `id` (Primary Key)
- `username`
- `content`
- `timestamp`

Purpose: Stores all chat messages with sender and time

Key Features

- Relationship:Oneuser can send multiple messages (One-to-Many relationship)
- Data Structure: Ensures structured data storage and efficient retrieval of chat history
- Security: Passwords stored in hashed format, Unique username constraint prevents duplication



Concurrency & Broadcasting

Connection Manager

Manages active WebSocket connections, tracks connected clients, handles connection lifecycle

Broadcasting

Distributes messages to all connected clients in real-time, implements pub-sub pattern

CLI Client

Command-line interface for users to connect, send messages, and receive broadcasts

API Endpoints Reference

POST /register	User registrationendpoint	username, password	User created successfully
POST /login	User authentication endpoint	username, password	Session token
WS /ws	WebSocket connection endpoint	Query parameter - token (session token)	Real-time message stream
GET /messages	Retrieve chat history	Optional - limit, offset	List of messages with timestamps
POST /logout	User logout endpoint	Session token	Session terminated

Future Enhancements

Private Messaging

One-to-oneencryptedconversations with separate message channels and notifications for new private messages.

User Presence Indicators

Real-time online/offlinestatus display, last seen timestamps, and typing indicators showing when users are composing mes s ages.

Message Encryption

End-to-endencryption for all messages with secure key exchange mechanisms for privacy-first communication.

User Profiles & Avatars

User profilepages withbioand avatar images, customizable user information, and profile visibility settings.

Thank You!