Real-Time Chat Application

1. System Architecture Overview

High-Level Architecture

The application follows a three-tier architecture designed for scalability, performance, and real-time communication capabilities.

CLIENT LAYER

(React Frontend, Socket.io Client)

|

APPLICATION LAYER

(Auth Module, Chat Module, User Module)

|

DATA LAYER

(MongoDB Database, Redis Cache, File Storage)

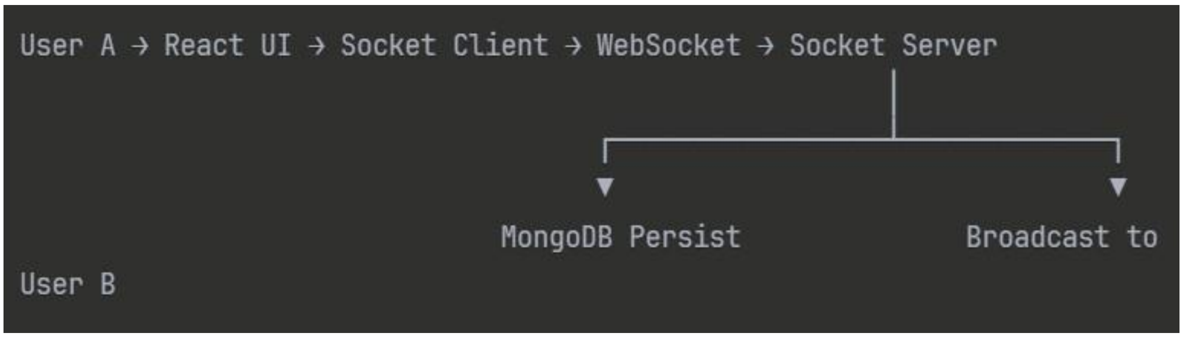
Technology Stack

Frontend: React.js provides component-based architecture with Socket.io-client enabling real-time bidirectional communication. React Router handles navigation while Context API manages application state.

Backend: Node.js with Express.js offers non-blocking I/O perfect for concurrent connections. Socket.io manages real-time events through WebSocket protocol. MongoDB provides flexible NoSQL storage for messages and user data. JWT enables stateless authentication supporting horizontal scaling.

Communication Architecture

Message Transmission Flow:



```
User A → React UI → Socket Client → WebSocket → Socket Server
                                                     |
                                    ┌────────────────┴────────────────┐
                                    ▼                                 ▼
                             MongoDB Persist                     Broadcast to
User B
```

2. Database Schema & Data Management

Collections Structure

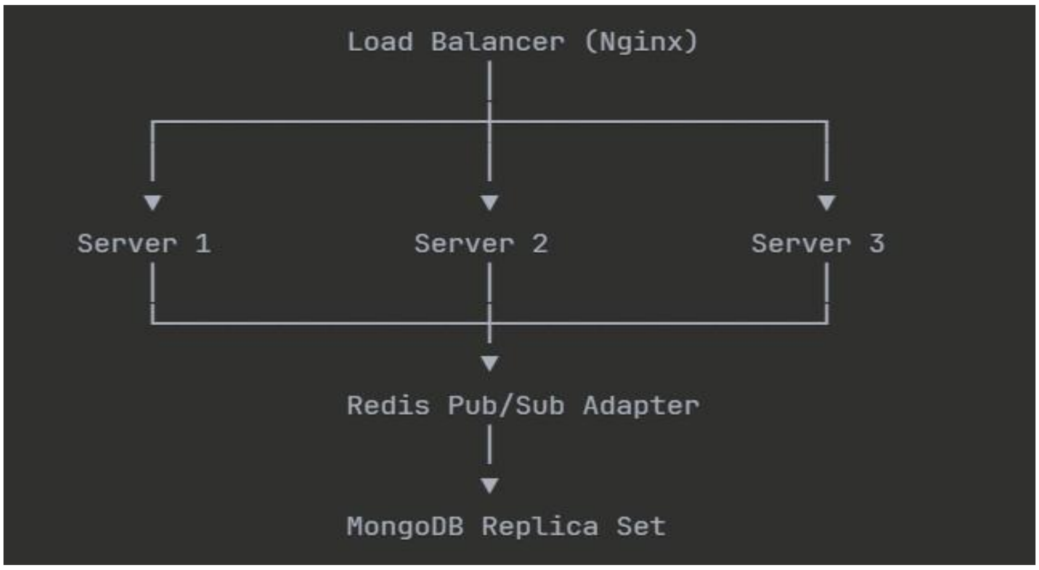
Users Collection stores account information including unique identifier, name, email (indexed and unique), hashed password, avatar URL, online status flag, last seen timestamp, and creation metadata.

Conversations Collection manages chat sessions with unique identifier, array of participant IDs (indexed), conversation type (direct or group), optional name for groups, embedded last message preview (content, sender, timestamp), and creation metadata.

Messages Collection contains individual messages with unique identifier, conversation reference (indexed), sender ID, message content, type indicator (text/image/file), delivery status (sent/delivered/read), array of users who read the message, and timestamp information.

3. Scalability & Performance Architecture

Horizontal Scaling Design



Redis Adapter enables Socket.io across multiple servers through publish-subscribe mechanism. When a user on Server 1 sends a message, Redis broadcasts to all servers, allowing Server 2 to deliver to its connected clients.

MongoDB Replication provides read scaling with primary node handling writes and replica nodes serving reads. Automatic failover ensures high availability. Sharding available for extreme scale scenarios.

4. Security Architecture

Authentication & Authorization

JWT Token System implements two-token strategy: short-lived access tokens (15 minutes) for API requests and long-lived refresh tokens (7 days) for session maintenance. Passwords use bcrypt hashing with 12 salt rounds. Token refresh mechanism prevents frequent re-authentication.

API Security Layers include CORS configuration restricting trusted domains, rate limiting preventing brute force (100 requests/minute), input validation sanitizing user data, and protection against SQL injection and XSS attacks through parameterized queries.

WebSocket Security requires authentication during handshake, validates tokens for every connection, implements rate limiting on messages (10 per second), and checks permissions before broadcasting events.

Data Protection enforces HTTPS encryption for all API communication, WSS (WebSocket Secure) for real-time connections, never exposes passwords in responses, and encrypts sensitive data at rest in database.

PROJECT DEVELOPMENT MODULE (Frontend and Backend(Spring Boot))

1. **Module 1 (Frontend – UI Design):**
   In this module you design the complete chat application interface using HTML and CSS, creating login pages, chat windows, message bubbles, contact lists, and making everything responsive so it works smoothly on mobile and desktop screens.

2. **Module 2 (Frontend – JavaScript & Interaction):**
   Here you make the chat app dynamic by handling user actions like sending messages, switching conversations, showing typing indicators, and updating the interface instantly using JavaScript without reloading pages.

3. **Module 3 (Frontend – API Integration):**
   This module connects the frontend to the backend by sending requests for login, fetching chat history, and posting new messages through REST APIs, allowing real data to flow between the UI and the server.

4. **Module 4 (Frontend – Real-Time Messaging):**
   You implement real-time communication on the frontend using WebSockets so messages appear instantly for all users, along with live status updates like online indicators and instant notifications.

5. **Module 5 (Backend – Spring Boot APIs):**
   In this module you build the core server using Spring Boot by creating REST controllers for user authentication, chat creation, and message handling, forming the main logic of the application.

6. **Module 6 (Backend – Database & JPA):**
   Here you design and manage the database using JPA and Hibernate to store users, messages, and conversations efficiently with proper relationships for fast and reliable data retrieval.

7. **Module 7 (Backend – Real-Time Engine):**
   This module sets up WebSocket communication on the server side to broadcast messages instantly between users, ensuring real-time chat functionality without repeated API polling.

8. **Module 8 (Backend – Security & Deployment):**
   In the final module you secure the application using JWT authentication, encrypt passwords, protect APIs, and deploy the complete chat system to a cloud platform so it can be accessed publicly.

Implementation: Add infinite scroll and virtual scrolling. Create animations and toast notifications. Implementdark mode and accessibility features. Optimize with code splitting. Deploy to Vercel/Netlify and configureCORS.

Conclusion

This system design delivers a scalable, secure, and performant real-time chat application. The architecture supports horizontal scaling for millions of concurrent users, maintains sub-100ms message delivery, implements multi-layered security protection, and provides 99.9% uptime reliability.