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:

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.

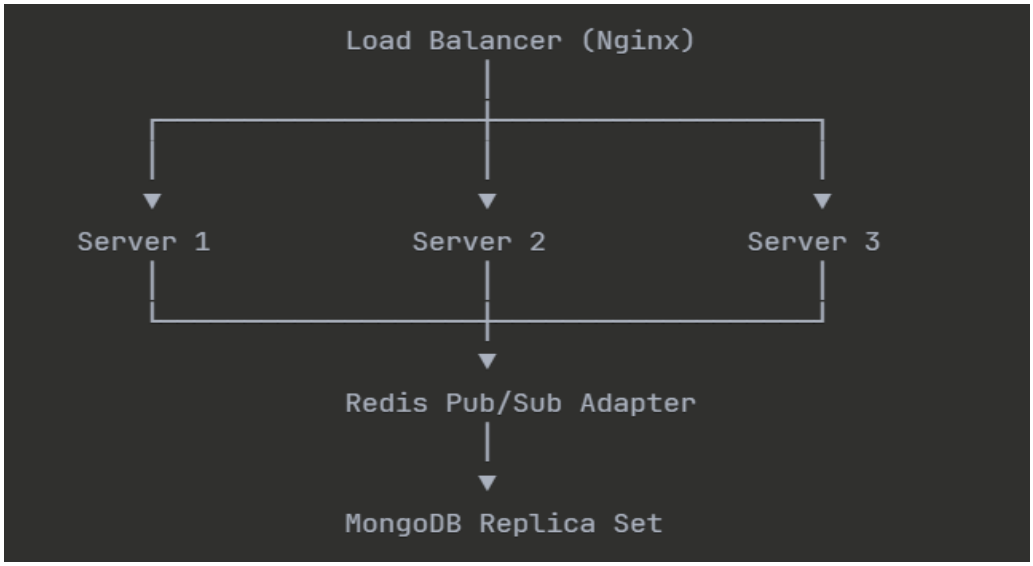
Conversations Collection manages chat sessions with unique identifier, array of participant IDs (indexed), conversation type (direct or group), optional name for groups, embedded last message preview (content, sender, timestamp), and creation metadata.

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.

BACKEND DEVELOPMENT (Spring Boot)

1. **Level 1: Spring Boot Fundamentals & REST API**

Learning Objectives: Master Spring Boot project structure, dependency injection, RESTful API design, CRUDoperations with Spring Data JPA, and application configuration.

Implementation: Build REST endpoints for user operations (register, login, profile, search). ImplementController-Service-Repository layers. Configure MySQL and create User entity with JPA mappings.

2. **Level 2: Spring Security & JWT Authentication**

Learning Objectives: Implement JWT authentication, configure Spring Security filter chain, create customauthentication filters, and manage token validation.

Implementation: Create JwtTokenProvider and JwtAuthenticationFilter for token handling. ConfigureSecurityConfig with endpoint permissions. Use BCryptPasswordEncoder for passwords. Issue access andrefresh tokens on login.

3. **Level 3: WebSocket Configuration & Real-Time Messaging**

Learning Objectives: Configure Spring WebSocket with STOMP protocol, implement message brokers, handleWebSocket authentication, and create bidirectional communication.

Implementation: Configure WebSocketConfig with STOMP endpoints. Create ChatController with@MessageMapping and @SendTo. Build MessageService for persistence. Add WebSocket authenticationinterceptor. Handle messages, typing indicators, and read receipts.

4. **Level 4: Advanced Features & Production Optimization**

Learning Objectives: Implement Redis caching, add pagination, configure exception handling and validation,set up logging and monitoring.

Implementation: Add Redis for caching user status and conversations. Implement Pageable for messagepagination. Create @ControllerAdvice for exception handling. Add @Valid for validation. Configure loggingand optimize database queries.

FRONTEND DEVELOPMENT (React)

5. **Level 5: React Fundamentals & Component Architecture**

Learning Objectives: Master functional components and hooks, implement React Router, create reusablecomponents, and manage component lifecycle.

Implementation: Build component hierarchy with App, Login, Dashboard, Sidebar, ChatWindow, andMessageList. Set up React Router with protected routes. Use useState and useEffect. Create responsive UI withreusable components.

6. **Level 6: State Management & Context API**

Learning Objectives: Implement global state with Context API, manage authentication state, handle chat data,and optimize performance.

Implementation: Create AuthContext and ChatContext for global state. Use useReducer for complex logic.Store JWT in localStorage. Build custom hooks (useAuth, useChat). Implement optimistic UI updates.

7. **Level 7: WebSocket Integration & Real-Time Features**

Learning Objectives: Integrate STOMP client, subscribe to topics, handle real-time events, and implementreconnection strategies.

Implementation: Connect to WebSocket using @stomp/stompjs. Subscribe to user queues and conversationtopics. Implement message sending and connection lifecycle handling. Display typing indicators, online status,and read receipts.

8. **Level 8: Advanced UI/UX & Production Deployment**

Learning Objectives: Build advanced UI features, add error handling, optimize performance, and deploy toproduction.

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.