Real-Time Chat Application

Full-Stack Learning Proposal with Spring Boot

System Design Overview

The application follows a three-tier architecture with React frontend communicating with Spring Boot backend via HTTPS for REST APIs and WebSocket Secure (WSS) for real-time messaging. The backend uses MySQL/PostgreSQL for data persistence and Redis for caching. Technology stack includes Spring Boot with Spring Security (JWT), Spring Data JPA, Spring WebSocket (STOMP), React with Context API, and Socket.io client.

BACKEND DEVELOPMENT (Spring Boot)

Level 1: Spring Boot Fundamentals & REST API

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

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

Level 2: Spring Security & JWT Authentication

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

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

Level 3: WebSocket Configuration & Real-Time Messaging

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

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

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 message pagination. Create @ControllerAdvice for exception handling. Add @Valid for validation. Configure logging and optimize database queries.

FRONTEND DEVELOPMENT (React)

Level 5: React Fundamentals & Component Architecture

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

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

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.

Level 7: WebSocket Integration & Real-Time Features

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

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

Level 8: Advanced UI/UX & Production Deployment

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

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

Key Learning Outcomes

Backend: Spring Boot architecture, dependency injection, JPA/Hibernate, Spring Security, JWT authentication, WebSocket/STOMP, Redis caching, RESTful design, exception handling, database optimization.

Frontend: React components and hooks, React Router, Context API, WebSocket integration, responsive design, performance optimization, production deployment.

Full-Stack: Client-server architecture, real-time communication, authentication flow, state synchronization, API integration, CORS, cloud deployment.

Success Metrics

- Real-time messaging with sub-200ms latency
- Secure JWT authentication with refresh mechanism
- Responsive UI for mobile and desktop
- Message persistence and history
- Typing indicators and presence system
- Production deployment on cloud platform

This progressive learning path builds full-stack expertise from fundamentals to production-ready features.