**# Chat Application System Design Document**

## Table of Contents

**1. Introduction**

- 1.1 Purpose

- 1.2 Definitions, Acronyms, and Abbreviations

- 1.3 References

**2. System Overview**

- 2.1 System Architecture

- 2.2 Technologies Used

- 2.3 Key Features

**3. System Design**

- 3.1 Frontend

- 3.2 Backend

- 3.3 WebSocket Integration

**4. API Design**

**5. Security Considerations**

- 5.1 Authentication and Authorization

- 5.2 Data Encryption

**6. Conclusion**

**## 1. Introduction**

**### 1.1 Purpose**

The purpose of this document is to provide a comprehensive overview of the system design for a real-time chat application developed using the MERN (MongoDB, Express.js, React, Node.js) stack, Socket.IO for real-time communication, and Chakra UI for the user interface. This document outlines the architectural decisions, key features, API design, security considerations, deployment strategies, and maintenance procedures for the chat application.

**### 1.2 Definitions, Acronyms, and Abbreviations**

- MERN: MongoDB, Express.js, React, Node.js

- REST: Representational State Transfer

- API: Application Programming Interface

- UI: User Interface

- JWT: JSON Web Token

**### 1.3 References**

- MERN Stack: https://mern.io/

- Socket.IO: https://socket.io/

- Chakra UI: https://chakra-ui.com/

- MongoDB: <https://www.mongodb.com/>

**## 2. System Overview**

**### 2.1 System Architecture**

The chat application follows a client-server architecture. It utilizes the MERN stack for the development of both frontend and backend components. The frontend is built using React with Chakra UI for the user interface, while the backend is implemented using Node.js and Express.js. Real-time communication is facilitated through Socket.IO, and MongoDB serves as the database for storing chat data.

**### 2.2 Technologies Used**

- Frontend:

- React

- Chakra UI

- Socket.IO Client

- Backend:

- Node.js

- Express.js

- Socket.IO

- MongoDB (via Mongoose)

**### 2.3 Key Features**

- User registration and authentication.

- Creation and management of chat rooms.

- Real-time chat messaging using Socket.IO.

- Retrieval of chat history via RESTful APIs.

- Notifications for new messages.

**## 3. System Design**

**### 3.1 Frontend**

The frontend is built using React and Chakra UI for a responsive and visually appealing user interface. Key components include:

- User Authentication: Users can sign up, log in, and manage their profiles.

- Chat Interface: Allows users to create chat rooms, send messages, and view chat history.

- Real-time Messaging: Utilizes Socket.IO for real-time messaging.

- Notifications: Provides notifications for new messages.

**### 3.2 Backend**

The backend is implemented using Node.js and Express.js, serving as the core of the application. Key components include:

- User Management: Handles user registration, login, and profile management.

- Chat Room Management: Manages the creation and leaving of chat rooms.

- Messaging Service: Enables real-time messaging through Socket.IO.

- Database Interaction: Utilizes MongoDB with Mongoose for storing user data, chat rooms, and messages.

- RESTful API: Exposes RESTful endpoints for retrieving chat history.

**### 3.3 WebSocket Integration**

WebSocket integration is achieved using Socket.IO, allowing real-time communication between clients and the server. This enables instant message delivery and updates in the chat interface.

**## 4. API Design**

**### 4.1 /api/chat**

For fetching all chats of a specific user.

**### 4.2 /api/message/chat\_id**

For fetching all messages of a specific group/ single chat using chat id.

**### 4.3 /api/user/search**

For searching users in the database by querying to the database.

**### 4.4 /api/chat/group**

For creation of new group.

**## 5. Security Considerations**

**### 5.1 Authentication and Authorization**

- User authentication is performed using JWT tokens.

- Authorization is enforced to ensure users can only access their own data and chat rooms.

**### 5.2 Data Encryption**

- Sensitive data (e.g., passwords, messages) is encrypted during transmission and storage using bcrypt.

**## 6. Conclusion**

This chat application is designed to offer real-time messaging capabilities, user authentication, chat room management, and RESTful API endpoints like /api/chats for accessing chat history.

By leveraging technologies such as Socket.IO for real-time communication and Chakra UI for a user-friendly interface, we aim to create a robust and responsive chat platform. The system design encompasses frontend and backend components, WebSocket integration, database design, API definitions, security measures, deployment strategies, and maintenance procedures.