Chat Application Requirement Document

Project Overview

Objective:

Develop a real-time chat application using the MERN stack with Socket.io for seamless communication.

Target Audience:

Users looking for a simple and efficient platform for real-time messaging and communication.

Functional Requirements

Authentication:

- Implement user authentication for secure login and registration.
- Include basic profile management features (update, delete, sign-out).

Chat Functionality:

- Users can create and join chat rooms.
- Real-time messaging with support for text, emojis.
- Implement basic message editing and deletion features.

User Interaction:

- Users can add friends and initiate private conversations.
- Implement notifications for new messages and friend requests.

Frontend:

- Use React.js for frontend development.
- Implement a responsive and user-friendly design.
- Utilize Socket.io-client for real-time communication.

Non-Functional Requirements

Performance:

- Ensure low latency and high responsiveness in real-time messaging.
- Optimize the application for a smooth user experience.

Security:

- Implement secure authentication and protect user data.
- Ensure secure transmission of messages using Socket.io.

Technology Stack

- MongoDB for the database
- Express.js as the backend framework
- React.js for the frontend
- Node.js as the runtime environment for the backend
- Socket.io for real-time communication
- JWT for authentication

Deployment

- Deploy the application to a hosting service (e.g., Heroku, Versel).

System Design

System Architecture:

- MERN stack architecture with Socket.io for real-time communication.
- WebSockets relies on ‘ws’ (websockets API) for WebSockets, Engine.IO to establish a long-polling connection.

Database Schema:

- UserModel: id, name, email, password, picture.
- ChatModel: id, chatName, isGroupChat, users, latest Message, Admin(only for group chat)
- MessageModel: sender, content, chat.

Authentication Flow:

- User registration with JWT and bcrypt.

User Interface Design:

- Design a clean and intuitive UI using React.js and CSS.
- Include pages for chat rooms, private messages, user profile, etc.

API Endpoints:

- Define API endpoints for user authentication, CRUD operations on chat rooms, messaging, etc.

Error Handling:

- Implement a consistent error-handling mechanism for API requests and form submissions.

Deployment Strategy

- Deploy the frontend and backend as separate services.
- Configure environment variables for easy deployment and configuration changes.

Testing

- Implement unit testing for critical functionalities.

Scalability

- Design the system to handle increased user loads efficiently.