# **System Design Document for Messaging Service Prototype**

## **1. Introduction**

### **1.1 Purpose**

This document outlines the system architecture, design choices, and setup instructions for the messaging service prototype.

### **1.2 Scope**

This Chat App is a real-time messaging platform that enables seamless communication between users through both private and group chats. It offers a user-friendly interface along with robust functionality for text communication. Additionally, it features an AI-powered chatbot for enhanced user interaction.

## **2. System Overview**

The Chat App allows users to create accounts, send messages in real-time, participate in group chats, and engage with an AI-powered chatbot. The system is designed for scalability and ease of use.

## **3. Architecture Design**

### **3.1 Overall Architecture**

The architecture follows a client-server model:

* **Frontend**: Next.js (React-based framework)
* **Backend**: Node.js (with Express)
* **Database**: NoSQL (MongoDB)

### **3.2 Component Descriptions**

* **Frontend**: User interface built with Next.js for efficient rendering and routing.
* **Backend**: RESTful API developed in Node.js to handle business logic.
* **Database**: Stores user data, messages, and chat histories.

## **4. Database Design**

### **4.1 Entity-Relationship Diagram (ERD)**

* **User**: user\_id, username, password\_hash, email
* **Message**: message\_id, sender\_id, recipient\_id, content, timestamp
* **ChatGroup**: group\_id, group\_name, user\_ids

## **5. Technology Stack**

* **Frontend**: Next.js, Tailwind CSS
* **Backend**: Node.js, Express.js
* **Database**: MongoDB
* **WebSocket**: For real-time messaging
* **Email Notifications :** Nodemailer for sending email alerts
* **Encryption :** JWT encryption for secure data transmission

## **6. API Design**

### **6.1 Endpoints**

* **POST /api/register**: User registration
* **POST /api/login**: User authentication
* **GET /api/messages**: Retrieve messages
* **POST /api/messages**: Send a new message

## **7. Security Considerations**

* **Authentication**: JSON Web Tokens (JWT) for secure user sessions.
* **Data Protection**: Passwords hashed using bcrypt.

## **8. Scalability and Performance**

* **Caching**: Use Redis for caching frequently accessed data.
* **Load Balancing**: Plan for horizontal scaling by using multiple instances of the backend.

## **9. User Interface Design**

### **9.1 Wireframes**

* Login page
* Chat interface
* User settings

## **10. Testing and Quality Assurance**

### **10.1 Strategies**

* **Unit Testing**: For individual components.
* **Integration Testing**: To ensure components work together.

## **11. Deployment and Maintenance**

* **Environment**: Deploy on platforms like Heroku or AWS.
* **Monitoring**: Use tools like LogRocket for frontend and Winston for backend logging.

## **12. Future Enhancements**

* Implement more robust AI features for the chatbot.
* Improve video/audio calling capabilities.

# **Setup and Run Instructions**

## **Prerequisites**

* **Node.js** (version 14 or higher)
* **MongoDB** (if using NoSQL)

## **Step 1: Clone the Repository**

bash

Copy code

git clone https://github.com/himanshup18/Chat-App.git

cd Chat-App

## **Step 2: Install Dependencies**

### **For Frontend**

Navigate to the frontend directory and install dependencies:

bash

Copy code

cd client

npm install

### **For Backend**

Navigate to the backend directory and install dependencies:

bash

Copy code

cd backend

npm install

## **Step 3: Configure Environment Variables**

Create a .env file in the backend directory and add:

makefile

Copy code

NEXT\_PUBLIC\_LOCALHOST\_KEY="chat-app-current-user"

## **Step 4: Start the Services**

### **Frontend**

bash

Copy code

cd client

npm run dev

### **Backend**

bash

Copy code

cd server

npm start

## **Step 5: Access the Application**

Open your browser and navigate to http://localhost:3000 for the frontend.

## **Libraries and Dependencies**

* **Next.js**: For efficient server-side rendering and routing.
* **Express**: Lightweight framework for handling HTTP requests in Node.js.
* **MongoDB**: For flexible data storage.
* **WebSocket**: For enabling real-time messaging capabilities.
* **Bcrypt**: For securely hashing passwords.

### **Why These Technologies?**

* **Next.js** provides excellent performance and ease of use for building interactive UIs.
* **Node.js** allows for a JavaScript stack, making it easier to share code between client and server.
* **MongoDB** offers flexibility with unstructured data, while PostgreSQL provides strong consistency and relational features.