System Design Document for Messaging Service Prototype

1. Introduction

This document outlines the system design for a messaging service prototype that supports real-time messaging, user registration, and authentication. The system is designed as a web application, utilizing modern front-end and back-end technologies to create an efficient and scalable messaging platform.

2. Key Features

- **User Registration & Authentication**: Allows users to sign up, log in, and securely access the chat service.
- **Real-Time Messaging**: Users can send and receive messages in real-time.
- **Group Chat Functionality**: Multiple users (for now , tabs) can be added to a group chat where they can see each other's messages.

3. System Components

3.1 Front-End (Client-Side)

- **Next.js** : Used for building the user interface.
  - **Reason**: Next.js allows for server-side rendering, improving performance.
- **Socket.IO (Client)**: For real-time, bi-directional communication between the browser and the server.
  - **Reason**: Socket.IO is easy to implement and provides real-time communication with low latency.
- **JWT (JSON Web Tokens)**: For user authentication and secure communication.
  - **Reason**: JWT is a widely used, lightweight method for securely transmitting information between parties as a JSON object.

3.2 Back-End (Server-Side)

- **Node.js**: The main server-side environment that executes JavaScript code.
  - **Reason**: Node.js is non-blocking and well-suited for real-time applications like chat services.
- **Express.js**: A minimalist web framework for Node.js.
  - **Reason**: Express.js simplifies routing, handling requests, and creating RESTful APIs.
- **Socket.IO (Server)**: Enables real-time, event-based communication between users and the server.
  - **Reason**: This allows for real-time messaging functionality.
- **MongoDB**: NoSQL database for storing user credentials and chat messages.
  - **Reason**: MongoDB is highly scalable and flexible, making it ideal for storing chat data, which can have varying structures.
- **Mongoose**: Object Data Modeling (ODM) library for MongoDB.

- **Reason**: Mongoose provides a straightforward way to model MongoDB documents in Node.js.

Dependency	Version	Reason for Usage
Next.js	Latest	Provides server-side rendering and static site generation.
Ionic React	Latest	For building cross-platform mobile and web applications.
Socket.IO	Latest	Enables real-time, bi-directional communication.
Express.js	Latest	Simplifies the routing and handling of HTTP requests.
MongoDB	Latest	NoSQL database for storing unstructured data.
Mongoose	Latest	Provides schema-based solutions to model MongoDB documents.
JWT	Latest	Used for secure authentication and token generation.
bcrypt	Latest	Password hashing for secure storage of user credentials.

Steps to Set Up and Run

1. Clone the Repository
2. Install Dependencies
3. Environment Variables
4. Start the Back-End Server - node server.js
5. Start the Front-End Client - npm run dev