System Design Document for Messaging Service Prototype

1. System Requirements:

Functional Requirements

- User registration and login
- Real-time messaging between users
- Creation and management of chat rooms
- User profiles and settings
- Message history retrieval

Non-Functional Requirements

- Scalability to handle multiple users
- High availability and reliability
- Secure user authentication and data protection
- Fast response time for real-time messaging

2. Operating Environment

- Frontend: Next.js running in a web browser
- Backend: Node.js with Express.js
- Database: MongoDB
- Development Tools: VSCode, Postman, MongoDB
- Deployment: Vercel (Frontend), Heroku/AWS (Backend), MongoDB Atlas (Database)

3. System and Subsystem Architecture

High-Level Architecture

- Frontend (Next.js): Handles user interactions, rendering, and REST API requests.

- Backend (Node.js, Express.js): Processes requests via RESTful endpoints for user authentication and message management.
- Database (MongoDB): Stores user data, messages, and chat rooms.

Subsystem Architecture

- Authentication Subsystem: Manages user login, registration, and token generation through REST APIs.
- Messaging Subsystem: Handles message sending, receiving, and storage via REST APIs and real-time communication.
- Chat Room Subsystem: Manages chat room creation and participant management.

4. Files and Database Design

Database Design Collections

- Users:
  - Schema: `{ username, passwordHash, email, createdAt }`
- Messages:
  - Schema: `{ senderId, receiverId, content, timestamp, roomId }`
- Chat Rooms:
  - Schema: `{ roomId, participants: [userIds], createdAt }`

Files:

- Frontend Files:
  - Components: `Login.js`, `Chat.js`, `UserList.js`, etc.
  - API Routes: `api/auth.js`, `api/messages.js`
- Backend Files:
  - Server: `server.js`
  - Controllers: `userController.js`, `messageController.js`

- Models: `User.js`, `Message.js`, `ChatRoom.js`

5. Input Formats

- User Registration/Login:
  - Input: JSON
  - Example: `{ "username": "user", "password": "pass" }`

- Message Sending:
  - Input: JSON
  - Example: `{ "content": "Hello", "receiverId": "user2" }`

6. Output Layouts

- User Response:
  - Example: `{ "success": true, "token": "jwt_token" }`

- Message Retrieval:
  - Output: JSON Array
  - Example: `[ { "senderId": "user1", "content": "Hello", "timestamp": "2023-09-20T10:00:00Z" }, ... ]`

7. Human-Machine Interface

- User Interface:
  - Login form with fields for username and password.
  - Chat interface displaying message history and input box.
  - User list sidebar to select chat participants.
- Accessibility Considerations:
  - Use semantic HTML for better accessibility.
  - Keyboard navigation support for forms and chat.

8. Detailed Design

Frontend Components

- Login Component: Handles user authentication via REST APIs.

- Chat Component: Displays messages and input box.
- User List Component: Shows active users and chat rooms.

Backend Routes

- Authentication Routes (REST APIs):
  - POST `/api/auth/login` - Authenticates user and returns JWT.
  - POST `/api/auth/register` - Registers a new user.
- Messaging Routes (REST APIs):
  - GET `/api/messages/:roomId` - Retrieves messages for a specific chat room.
  - POST `/api/messages` - Sends a new message.

9. Processing Logic

Login Process

1. User submits a login form.
2. Frontend sends a request to the backend's REST API with credentials.
3. Backend validates credentials and generates a JWT token.
4. Token is sent back to the frontend for session management.

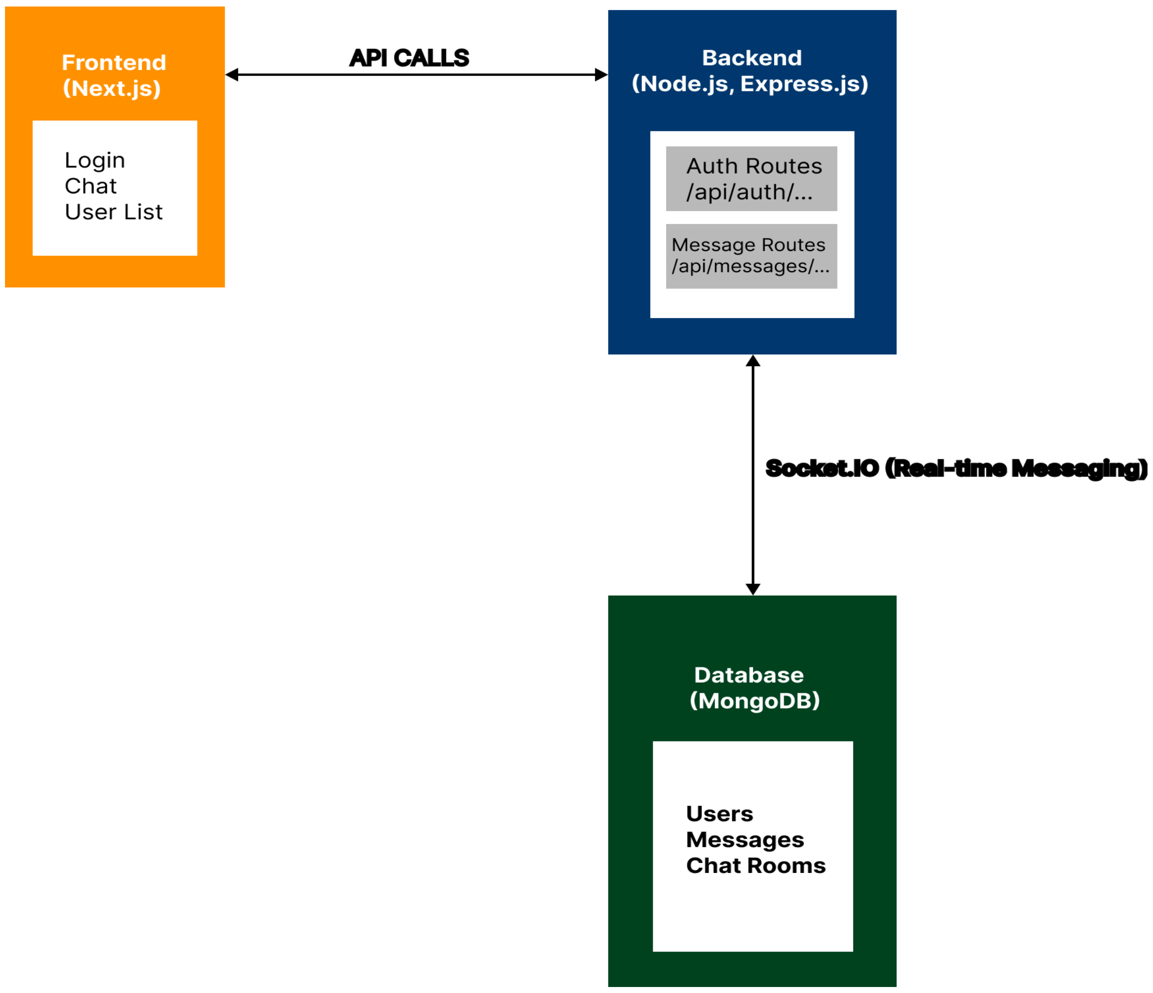
Messaging Process

1. User sends a message from the chat interface.
2. Frontend emits the message via Socket.IO to the server.
3. Server processes the message and stores it in the database.
4. Server broadcasts the message to all connected clients in the room.

Explanation of the Diagram:

- Frontend (Next.js): The user interface where users can log in, send messages, and view active chats. It communicates with the backend via REST APIs for authentication and message management.
- Backend (Node.js, Express.js): Contains routes for authentication and messaging. It processes API requests

- and manages real-time messaging through Socket.IO.
- Database (MongoDB): Stores user data, messages, and chat rooms, facilitating data persistence.


Frontend (Next.js)
Login
Chat
User List
API CALLS
Backend (Node.js, Express.js)
Auth Routes /api/auth/...
Message Routes /api/messages/...
Socket.IO (Real-time Messaging)
Database (MongoDB)
Users
Messages
Chat Rooms

10.External Interfaces:

REST APIs

- Authentication API: Used for user login and registration via RESTful endpoints.
- Messaging API: Used for sending and retrieving messages via RESTful endpoints.

Third-Party Services

- Socket.IO: For real-time communication.
- MongoDB Atlas: For database hosting.