Talky

Introduction

Welcome to our MERN stack-based real-time chat and video/audio call application, a modern solution for seamless communication. In this application, we leverage the power of cutting-edge technologies to deliver a feature-rich and responsive experience.

Key highlights of our application:

- **Real-Time Chat Conversations**: We utilize Socket.io to ensure that chat conversations happen in real-time, allowing users to exchange messages instantly and effortlessly.
- **Crystal Clear Audio and Video Calls**: Integrated with Zego Cloud, our application supports high-quality audio and video calls, enabling users to connect face-to-face and voice-to-voice from anywhere in the world.
- **Elegant React JS Frontend:** We've designed a beautiful and intuitive frontend using React JS, providing a user-friendly interface that's both visually appealing and highly functional.
- **Scalable NoSQL MongoDB:** For the backend database, we rely on MongoDB, a NoSQL database that excels in scalability. This choice is essential to accommodate the high volume of data generated by our users, ensuring smooth and efficient operations.

Our application is not just about text-based communication; it's a dynamic platform that brings people closer through real-time chat and high-quality video/audio calls. Whether it's for personal connections, business meetings, or social interactions, our application is your gateway to seamless and scalable communication.

Design Chat Application

1. Requirements

- Group chat: users can participate in group conversation
- Direct messaging: two users can chat with each other
- Video and audio call with one to one user.
- Group video and audio call
- Join/leave groups.
- send notifications to the user
- User status: whether you are online or offline

2. High-level Design

2.1. Database Design

Read Operations

- Given group G, retrieve all messages
- Given user A and user B, retrieve messages.
- find users by name or email

Write Operations

- Authentication
- Save a new message by user A in group G
- Save a message between user A and user B
- Add/delete user A to/from group G

In our case, it is obvious that the database is used primarily as a key-value store. No complex relational ops such as join are needed. In addition to the access pattern, keep in mind that the database must be horizontally scalable and tuned for writes.

In our case, we could use NoSQL database MongoDb

Schema

User

_id (Primary Key)	name (String)	email (String, Unique)	password (String)	pic (String, Default:	isAdmin (Boolean)

Message

_id (Primary Key)	sender (Foreign Key, References: User._id)	content (String)	chat (Foreign Key, References: Chat._id)	readBy (Array of Foreign Keys, References: User._id)

Foreign Keys: - sender (References User._id) ,chat (References Chat._id) , readBy (References User._id)

Chat

_id (Primary Key)	chatName (String)	isGroupChat (Boolean, Default: false)	latestMessage (Foreign Key, References: Message._id)	groupAdmin (Foreign Key, References: User._id)

Foreign Keys: - latestMessage (References Message._id) , groupAdmin (References User._id)

2.2. API Design

Chat API:

- POST /chat: Access a chat.
- GET /chat: Fetch user's chats.
- POST /chat/group: Create a group chat.
- PUT /chat/rename: Rename a group chat.
- PUT /chat/groupremove: Remove a user from a group.
- PUT /chat/groupadd: Add a user to a group

User API:

- POST /users: Register a user.
- POST /users/login: Authenticate a user.
- GET /users: Get all users.
- GET /users/access-token: Get an access token

Message API:

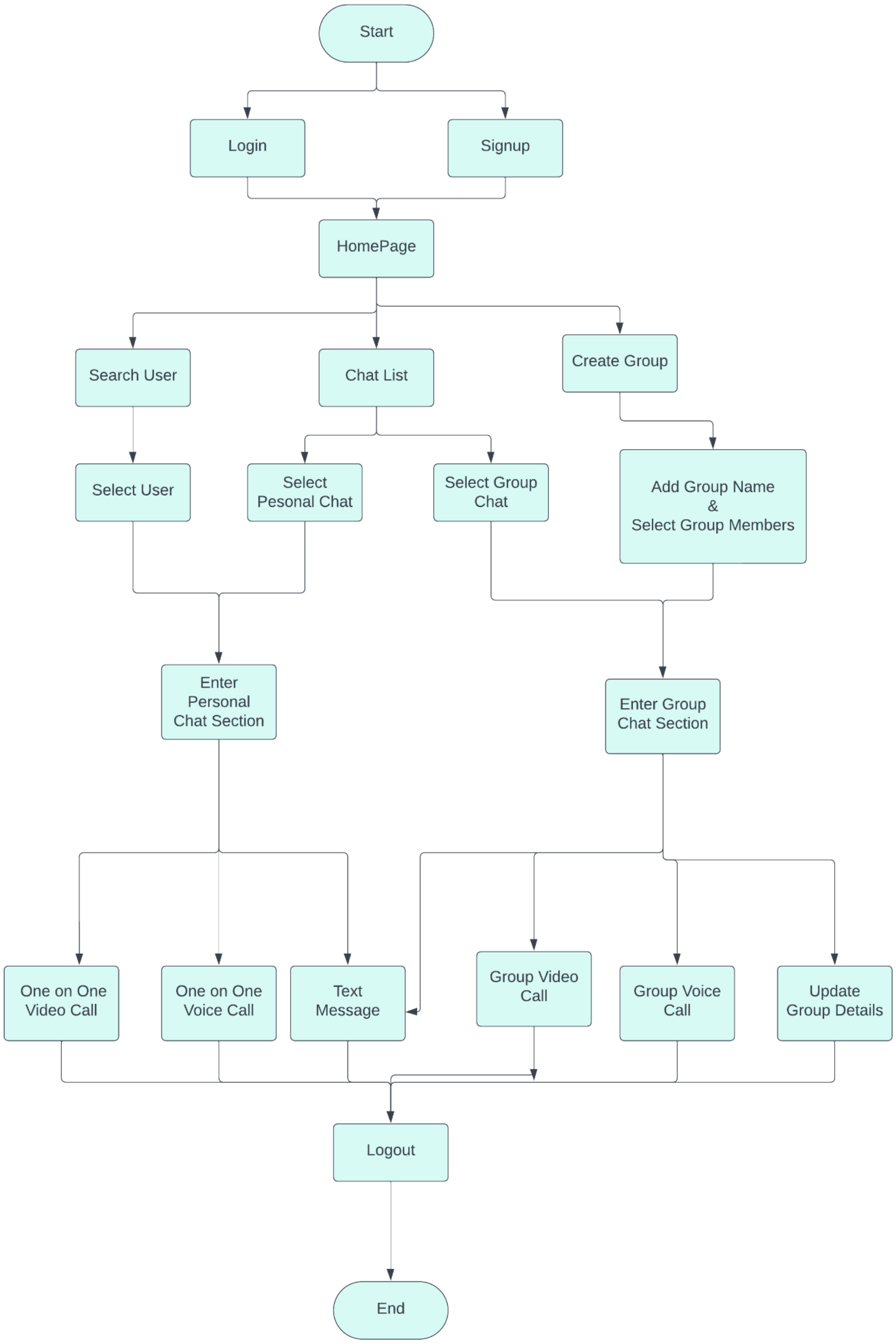
- GET /messages/:chatId: Get chat messages.
- POST /messages: Send a message

2.3. Architecture

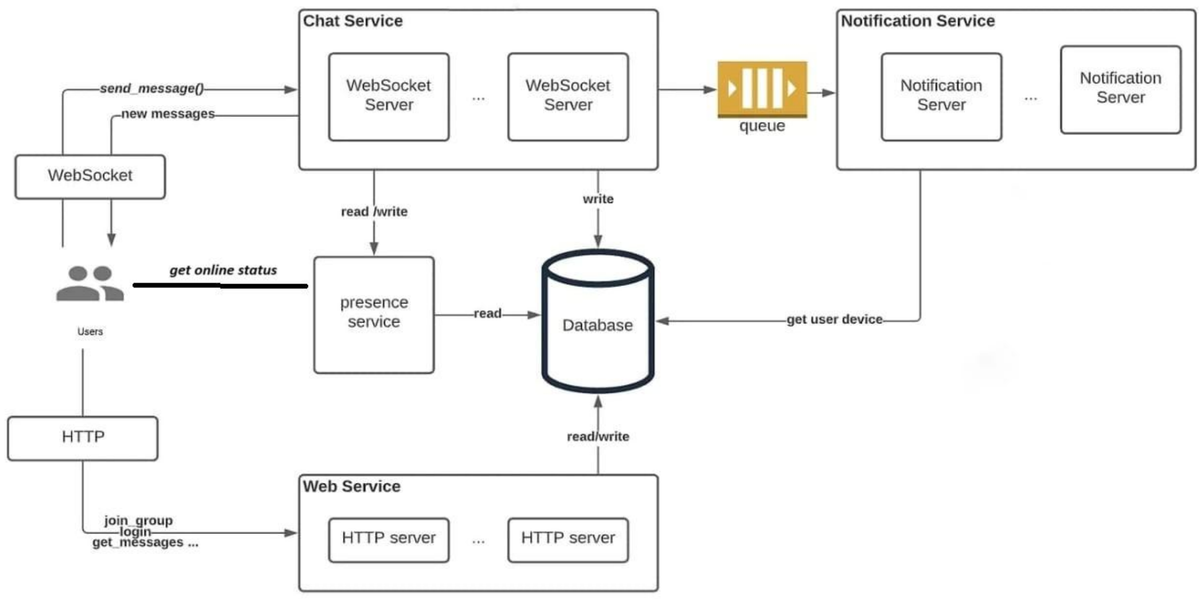
Chat Service: each online user maintains a WebSocket connection with a WebSocket server in the Chat Service. Outgoing and incoming chat messages are exchanged here.

Web Service: It handles all RPC calls except send_message(). Users talk to this service for authentication, join/leave groups, etc.

Flowchart of Application

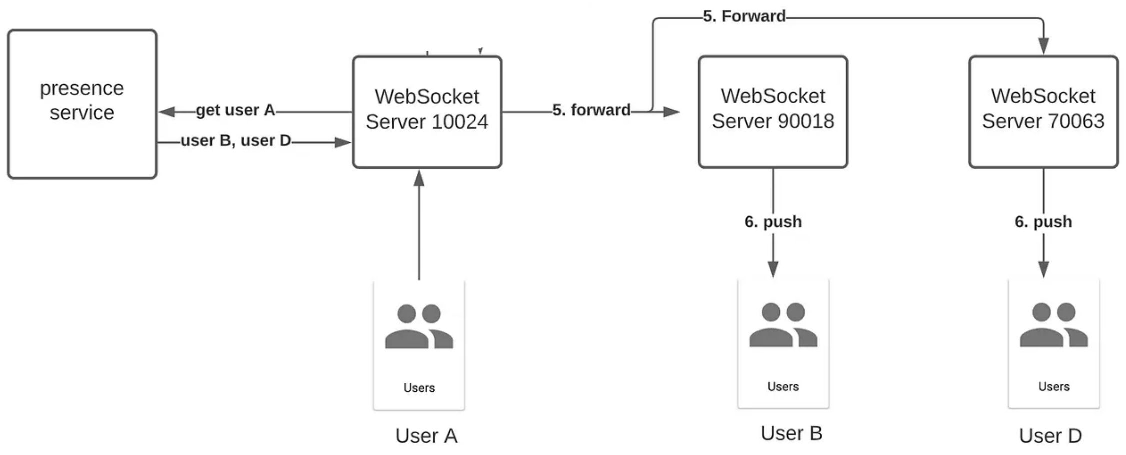


High-Level Architecture



Presence Detection

the WebSocket server contacts the Presence Service who returns a list of contacts for broadcast



In conclusion, the technical decisions encompassed the choice of technology stack, real-time communication integration, database selection and schema design, API structure, user authentication, scalability considerations, and prioritization of a seamless user experience. These decisions collectively contribute to a modern and scalable communication platform.