Title

MSG YOU

September 2024

Contents

1 Introduction

1.1 Problem Statement

The challenge is to design and develop a full-stack messaging service prototype that not only enables text-based communication but also supports group chats, real-time updates, and audio/video call features. This solution must prioritize performance, scalability, and user experience while leveraging modern technologies such as Next.js, Socket.io, Node.js, Firebase, Tailwind CSS, Prisma, and PostgreSQL.

The key requirements include:

- **User Registration and Authentication:** Implementing a secure, efficient authentication system for users.
- **Real-time Messaging:** Ensuring smooth and instant message delivery using socket.io .
- **Voice and Video Calls:** Integrating reliable voice and video calling features to enhance user communication.
- **Scalability:** Building a robust backend that can handle increasing numbers of users and messages without performance degradation.

This Assignment will tackle these challenges by leveraging modern development frameworks and technologies to provide an optimized and comprehensive messaging platform.

1.2 System Design

1.2.1 High-Level Architecture Overview

Client-Server Architecture **Client:** User interface built with Next.js for a responsive web application that supports user interactions like registration, messaging, and group chats.

Server: REST APIs built with Node.js, handling requests from the frontend, processing data, and communicating with the database and real-time services.

1.2.2 User Authentication

Firebase Authentication: Used for managing user registration and authentication. This allows for secure and scalable user management without the need to build custom authentication systems.

1.2.3 Real-time Communication

Socket.io: Utilized for real-time message updates, enabling instantaneous communication between users as they send and receive messages within chats and groups.

1.2.4 Database

PostgreSQL: Relational database used to store structured data, including user profiles, messages, and group chat details. The choice of PostgreSQL allows for robust data management with strong integrity and relational capabilities.

1.2.5 Video and Voice Call Features

Zego Cloud: Integrated for video and audio calling functionalities, leveraging their APIs to facilitate real-time communication beyond text messaging.
Click the URL for the Zego Cloud Dashboard.

1.2.6 API Communication

RESTful APIs: The system uses RESTful API endpoints to manage interactions between the client and server. Key functionalities include:

- User Registration: Endpoint for user signup.
- User Login: Endpoint for authenticating users.
- Message Sending: Endpoint for sending messages between users.
- Group Chat Management: Endpoints for creating and managing group chats.

2 User Flow

2.1 User Registration/Signup

Action: User selects to sign up using Google.
Process:

- If Google sign-up is successful:
  - User is prompted to select a profile picture/avatar.
  - User enters their profile name.
  - User is directed to the chat interface.

2.2 User Login

Action: User selects to log in.
Process:

- User enters credentials (if applicable).
- On successful login, user is directed to the chat interface.

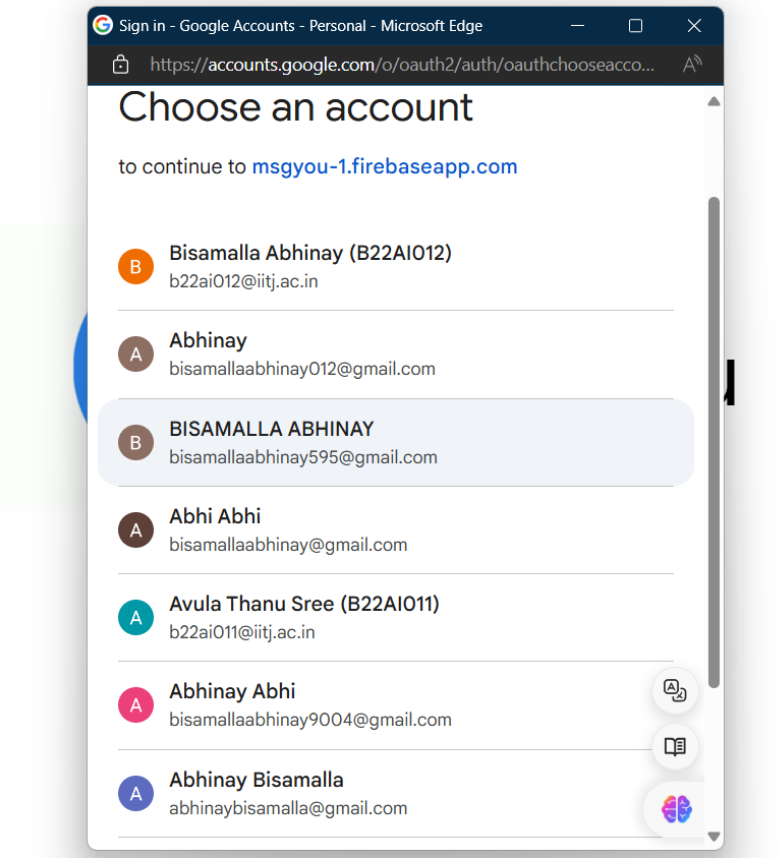


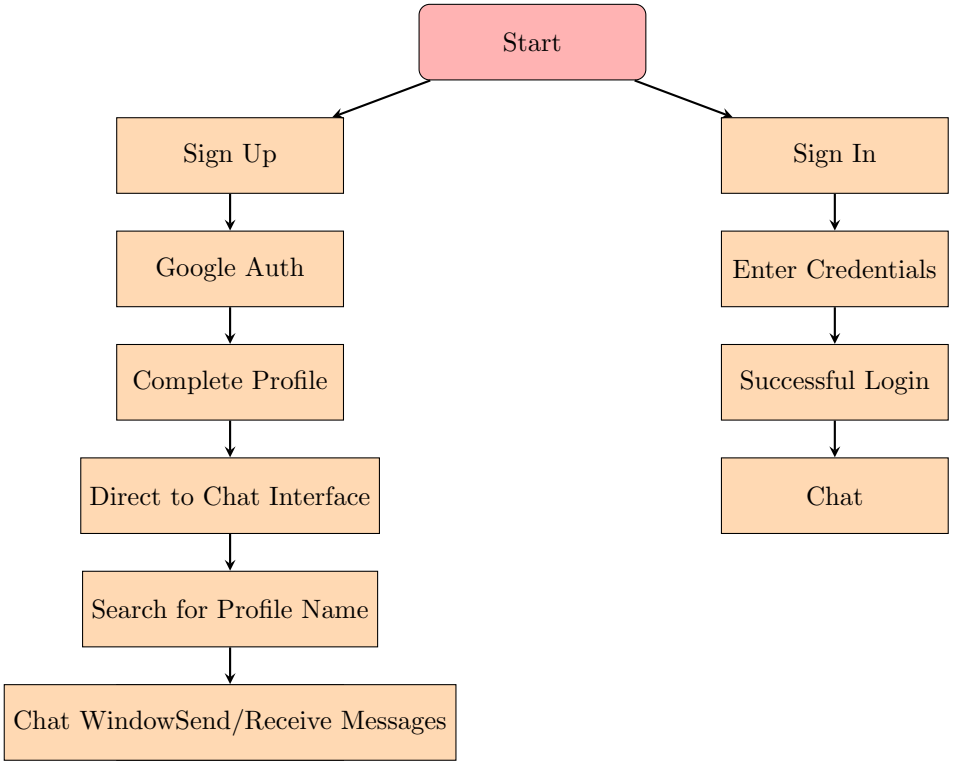
Figure 1: User Authentication

2.3 Starting a Chat

Action: User searches for a profile name.
Process:

- User types in the name of the contact they wish to chat with.
- User selects the contact from search results.
- User is taken to the chat window to send and receive messages.

2.4 User Flow Diagram


Start
Sign Up
Google Auth
Complete Profile
Direct to Chat Interface
Search for Profile Name
Chat WindowSend/Receive Messages
Sign In
Enter Credentials
Successful Login
Chat

3 Implementation

3.1 Registration and Authentication

The login functionality is implemented using Firebase for authentication, allowing users to sign in with their Google accounts. The application checks if the user is new or existing and manages their state accordingly.

3.1.1 Key Features

- **Google Sign-In:** Users can log in quickly using their Google credentials.
- **User State Management:** The application manages user information and login status using a state management context.
- **Conditional Routing:** Users are redirected based on their login status, enhancing the user experience by streamlining access to the chat interface.

3.2 Firebase Configuration

Firebase is initialized with a configuration that includes API keys and project identifiers. This setup allows for secure user authentication and management without the need for building custom authentication systems.

3.2.1 Key Features

- **Firebase Initialization:** The app is connected to Firebase services, enabling user authentication.
- **Exported Auth Instance:** The authentication instance is made available throughout the application for seamless login management.

3.3 Realtime Communication

3.3.1 5. Real-Time Messaging Implementation

The chat container is designed to display messages in real time, ensuring users have a seamless chatting experience. It dynamically updates as new messages arrive, automatically scrolling to the latest message.

3.3.2 Key Features

- **Message Rendering:** Supports multiple message types including text, images, and audio, allowing for a rich communication experience.
- **Automatic Scrolling:** The chat interface scrolls smoothly to the most recent message, keeping the conversation in view.
- **Message Status Indicators:** Displays message status (e.g., sent, delivered) for better user feedback.

3.3.3 5.2 Voice Message Component

The voice message functionality enhances the messaging experience by allowing users to send and receive audio messages.

3.3.4 Key Features

- **Audio Playback:** Users can play and pause audio messages with intuitive controls.
- **Waveform Visualization:** Integrates WaveSurfer.js to display an audio waveform, providing a visual representation of the audio message duration.
- **Playback Time Tracking:** Displays the current playback time and total duration of the audio message, giving users better context during playback.

Figure 2: Voice Msg

3.4 Integration with Real-Time Communication

The messaging system utilizes Socket.io for real-time message delivery. This allows messages to be sent and received instantaneously, creating a fluid chatting experience.

3.4.1 Key Benefits

- **Instant Notifications:** Users receive immediate notifications of new messages, enhancing communication efficiency.
- **Scalability:** The use of WebSockets ensures that the system can handle a growing number of users without performance issues.

3.5 Video and Voice Call Implementation

3.5.1 6.1 Video Call Component

The VideoCall component facilitates video calling functionality, enabling users to initiate and receive video calls in real time.

3.5.2 Key Features

- **Outgoing Call Handling:** When a user initiates a video call, the component emits an event to the server, signaling the call initiation to the intended recipient.
- **Dynamic UI Updates:** The component updates in real time based on the call state, ensuring users are informed about the status of the call.

3.5.3 6.2 Voice Call Component

Similar to the VideoCall component, the VoiceCall component manages voice calling capabilities.

3.5.4 Key Features

- **Real-Time Call Management:** The component handles outgoing voice call requests, communicating with the server to connect calls efficiently.
- **Integration with Socket.IO:** Enables real-time updates and event handling for voice call sessions.

3.5.5 6.3 Call Container

The Call Container component is central to managing the call's user interface and interactions.

3.5.6 Key Features

- **Stream Management:** Utilizes ZEGOCLOUD's SDK to create and manage local and remote streams for both audio and video calls. It handles user audio and video streams dynamically based on the call type.
- **Call Status Updates:** Monitors and updates the call status, including accepting calls and notifying when a call is ongoing.
- **End Call Functionality:** Provides users with the ability to end a call, managing clean-up of resources and updating the UI accordingly.

3.6 Integration with Real-Time Communication

The calling features leverage Socket.IO for real-time signaling and interaction, allowing for immediate connection and communication between users.

3.6.1 Key Benefits

- **Scalability:** The architecture supports multiple concurrent calls, making it suitable for larger user bases.
- **User Experience:** By providing clear visual feedback and seamless interactions, users can engage in calls with minimal friction.

4 Database Design

4.1 Database Model

Database Type: SQL (PostgreSQL)

4.2 User

4.2.1 Attributes

- `id`: Primary Key, Auto-incrementing Integer
- `email`: Unique String
- `name`: String
- `profilePicture`: String (default empty)
- `about`: String (default empty)

4.2.2 Relationships

- One-to-Many relationship with Messages (as sender and receiver).

4.3 Messages

4.3.1 Attributes

- `id`: Primary Key, Auto-incrementing Integer
- `senderId`: Foreign Key referencing User.id (the sender)
- `recieverId`: Foreign Key referencing User.id (the receiver)
- `type`: String (default "text")
- `message`: String
- `messageStatus`: String (default "sent")
- `createdAt`: DateTime (default current timestamp)

4.3.2 Relationships

- Each message is linked to a sender and a receiver.

4.4 Explanation of Tables

User Table: Stores user details, including their unique identifier, email, name, profile picture, and a brief bio. This table facilitates user identification and management.

Messages Table: Contains all messages exchanged between users. Each message references both the sender and the receiver, allowing for easy tracking of communication.

4.5 Justification for Database Choice

Rationale for Choosing SQL (PostgreSQL):

- **Structured Data:** The application requires a clear structure for user and message data, making a relational database suitable.
- **Complex Queries:** SQL databases are adept at handling complex queries and joining data across multiple tables, which is essential for retrieving message histories and user interactions.
- **Data Integrity:** Using foreign keys ensures referential integrity, which is critical for maintaining accurate relationships between users and their messages.
- **Scalability:** PostgreSQL offers scalability features that can support the growth of user data and messages over time without sacrificing performance.
- **ACID Compliance:** Ensures reliable transactions and data consistency, which is essential for messaging applications where data accuracy is crucial.

This design provides a solid foundation for a messaging application, ensuring both efficiency and reliability in data management.

5 App Features

5.1 User Profile Management

- **Search Contacts or Profiles:** Users can search for contacts or view profiles by entering names or other identifiers.
- **View User Profiles:** Clicking on a contact displays their profile details, including profile picture, about section, and status.

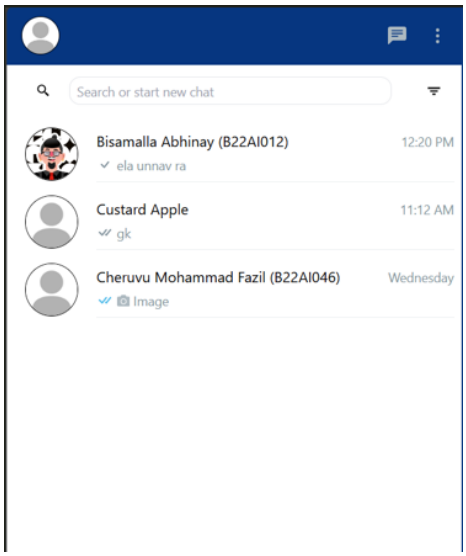


Figure 3: Search Profile

5.2 Message Management

- **Seen and Unseen Indicators:** Implement blue ticks to show if messages have been seen (similar to WhatsApp).
- **Search Messages:** Users can search through their message history to find specific conversations or messages.

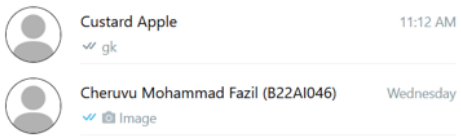


Figure 4: Read Unread Status

5.3 Communication Features

- **Send Files:** Users can attach and send files (images, documents, etc.) within chats.
- **Send Emojis:** Support for sending emojis in messages to enhance communication.
- **Audio Messages:** Users can record and send audio messages.

Figure 5: Type Container

- **Video Calls:** Initiate video calls with contacts.
- **Voice Calls:** Ability to make voice calls directly from the app.

Figure 6: Video And Audio Call Container

5.4 User Interaction

- **Direct Access to Profile List:** Double-clicking on a user shows all associated profiles or chat history with that user.
- **Exit Chat:** Users can exit or close chat windows easily.

5.5 User Status

- **Online/Offline Status:** Display user status (online or offline) to inform contacts of availability.
- **Last Seen:** Optionally show the last time a user was active.

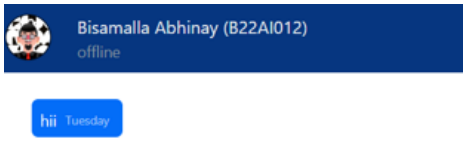


Figure 7: Online Offline Status

5.6 Account Management

- **Logout:** Users can log out of their accounts to secure their profiles and conversations.

5.7 Unread Message Count

- **Display Unread Message Badge:** For each user or contact in the chat list, show a small badge or icon on the right side that indicates the number of unread messages.
- **Badge Positioning:** Typically, this badge is positioned on the right side of the user's profile in the chat list, similar to WhatsApp.
- **Dynamic Count:** The number will update in real-time as new messages arrive and when messages are read.
- **Reset on Viewing:** Once a user opens the chat, the unread message count for that profile will reset to zero.

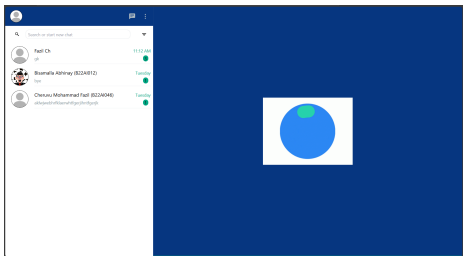


Figure 8: UnRead Msg

5.7.1 How This Feature Works

- **Message Status Tracking:** Each message can have a status such as "unread" or "read". The unread count for a user will increment each time a message is received but not opened.
- **UI Integration:** The unread count will appear next to the user's profile in the chat list, providing a quick view of how many unread messages are pending for each contact.
- **Real-Time Updates:** Using WebSockets or a similar real-time system (such as your current socket implementation), you can dynamically update the unread message count when new messages are sent and received.

6 Conclusion

This report outlines the key features and technical aspects of the messaging service prototype. The application includes user authentication, real-time messaging, and additional features such as voice and video calls to enhance the user experience. Technologies like Firebase and Socket.IO were chosen for their scalability and ease of use, while PostgreSQL was selected for managing structured data with complex relationships. The prototype is designed with an intuitive user interface and can scale with user growth, offering a robust communication platform.

Overall, the app's architecture is designed to be flexible and extensible, allowing for the integration of more advanced features like AI-powered chatbots or message thread archiving in the future.