Phase – 1:

**Real-Time Chat Application with React, Flask, Firebase, and Google Authentication**

**Introduction**:

We propose to build a real-time chat application using React for the frontend, Flask for the backend, Firebase as the database, and Google Authentication for user authentication. The application will allow users to create accounts, join chat rooms, and exchange messages in real-time. Users will be able to login using their Google accounts, and their authentication will be handled by Firebase.

**Features**:

**User Authentication**: Users will be able to create accounts and login using their Google accounts for authentication. Firebase will handle the user authentication process, including secure storage of user credentials.

**Profile Management**: Users will be able to manage their profile information, including their profile picture and display name. Users will be able to update their personal information and view their activity history.

**Chat Rooms**: Users will be able to join chat rooms based on different channels, such as news, sports, food, dogs, and announcements. They will be able to exchange messages with other users in real-time within the chat rooms.

**Announcement Feature:** Admins of the application will be able to create announcements and broadcast them to all users. Users will be able to view announcements and see the time and date they were posted.

**Logout Functionality**: Users will be able to logout from their accounts, which will securely invalidate their session and protect their account information.

Database:

Firebase, a NoSQL cloud-based database, will be used to store the data for the application, including user profiles, chat messages, and announcements. Firebase provides real-time synchronization and scalable storage, making it suitable for a real-time chat application.

Backend:

Flask, a micro web framework, will be used for the backend of the application. Flask will handle the user authentication process with Firebase, as well as handle the creation and management of chat rooms and announcements. Flask will also establish an instance to enable real-time communication between clients within the chat rooms.

Frontend:

React, a popular JavaScript library for building user interfaces, will be used for the frontend of the application. React will handle the rendering of the chat room interface, user profile management, and login/logout functionality. React components will communicate with the Flask backend to send and receive data from Firebase.

Responsibilities:

Suraj Mandal will be responsible for developing the frontend using React and integrating it with the Flask backend.

Vedavyas Venkata Narasimha Vedavyas Muppavarapu will be responsible for developing the backend using Flask, integrating with Firebase for user authentication and database management.

Conclusion:

This project aims to create a real-time chat application using React, Flask, Firebase, and Google Authentication. The application will allow users to create accounts, join chat rooms, and exchange messages in real-time. The use of Firebase for the database and Google Authentication for user authentication will provide a secure and scalable solution. The combination of React and Flask will result in a modern and responsive web application with real-time chat functionality.

Phase -2:

## Tables and Fields:

|  |  |
| --- | --- |
| **Tables** | **Fields** |
| **Users**: stores user information, such as username, email, password. | id INT(11) ,name VARCHAR(50), email VARCHAR(50), password VARCHAR(255), profile\_picture VARCHAR(255), UNIQUE KEY (email) |
| **Messages**: This table will store direct messages sent between users. It will include the sender and receiver's user ID, the message content, and the timestamp. | id INT(11), sender\_id INT(11), receiver\_id INT(11) , content TEXT, timestamp DATETIME |
| **Groups**: This table will store information about each group, including the group name and description. | id INT(11),name VARCHAR(50), description VARCHAR(255) |
| **Groups Members**: This table stores the user's task groups, including task group ID, user ID, task group name, and description. | id INT(11), group\_id INT(11), user\_id INT(11) |
| **Announcements:** This table will store announcements created by admins. It will include the announcement content and the timestamp. | id INT(11) ,content TEXT , timestamp DATETIME |

## Tables and Usage

|  |  |
| --- | --- |
| **Table** | **Usage** |
| Users | This table will store user information, including their name, email, password and date of birth. |
| Messages | This table will store direct messages sent between users. It will include the sender and receiver's user ID, the message content, and the timestamp. |
| Groups | This table will store information about each group, including the group name and description. |
| Group Members | This table will store the relationship between groups and their members. It will include the group ID |
| Announcements | This table will store announcements created by admins. It will include the announcement content and the timestamp. |

An SQL scripts are created as below for each table with necessary data types as shown below –

-CREATE TABLE Users ( id INT(11) NOT NULL AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

email VARCHAR(50) NOT NULL,

password VARCHAR(255) NOT NULL,

profile\_picture VARCHAR(255),

PRIMARY KEY (id),

UNIQUE KEY (email) );

- CREATE TABLE Messages ( id INT(11) NOT NULL AUTO\_INCREMENT,

sender\_id INT(11) NOT NULL,

receiver\_id INT(11) NOT NULL,

content TEXT NOT NULL,

timestamp DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

PRIMARY KEY (id),

FOREIGN KEY (sender\_id) REFERENCES Users(id),

FOREIGN KEY (receiver\_id) REFERENCES Users(id) );

-CREATE TABLE Groups ( id INT(11) NOT NULL AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

description VARCHAR(255) NOT NULL,

PRIMARY KEY (id) );

-CREATE TABLE GroupMembers ( id INT(11) NOT NULL AUTO\_INCREMENT,

group\_id INT(11) NOT NULL,

user\_id INT(11) NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (group\_id) REFERENCES Groups(id),

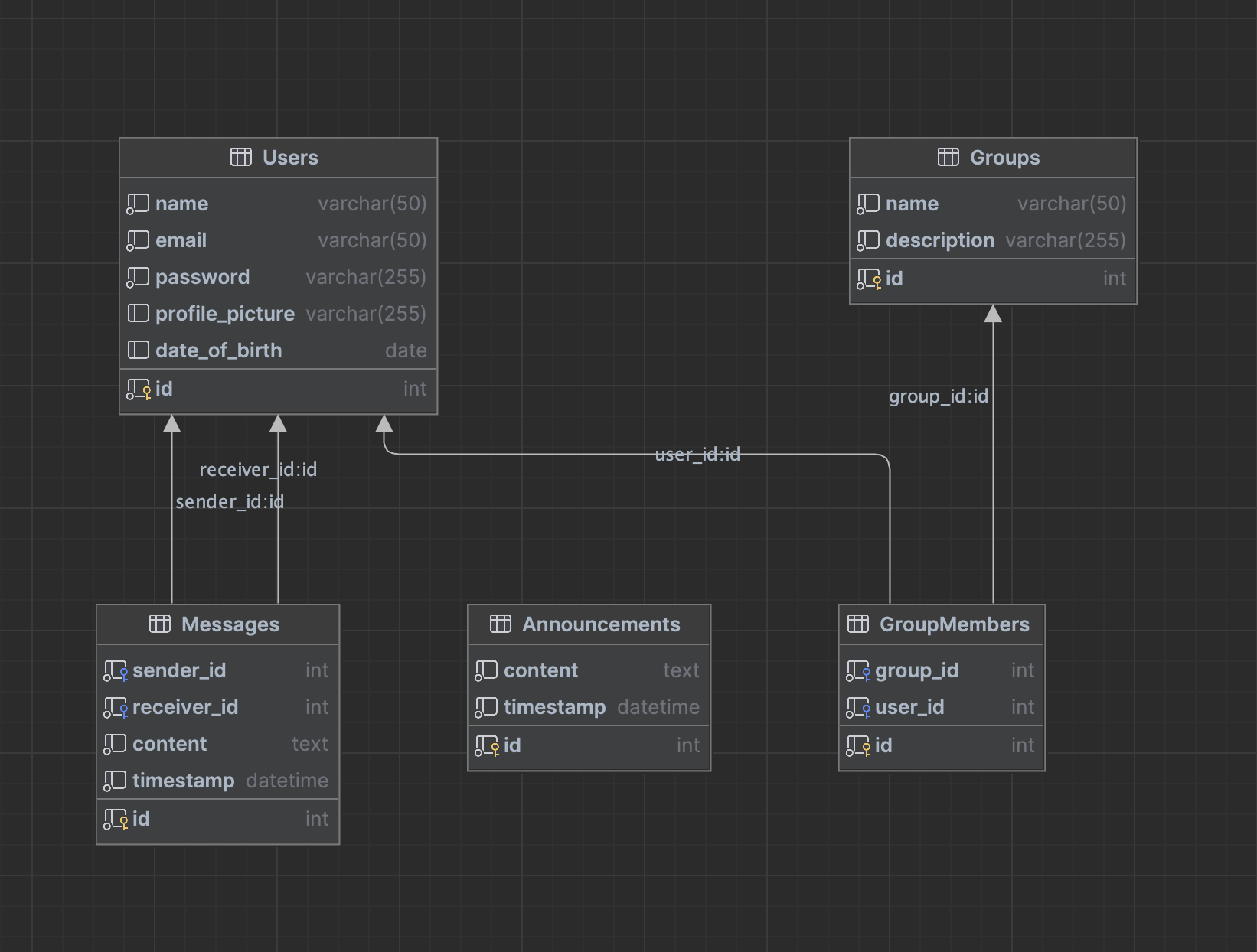
FOREIGN KEY (user\_id) REFERENCES Users(id) );

-CREATE TABLE Announcements ( id INT(11) NOT NULL AUTO\_INCREMENT,

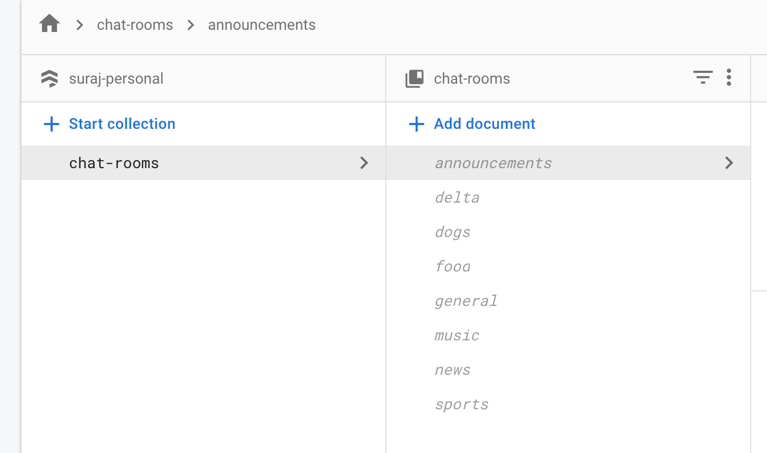
content TEXT NOT NULL,

timestamp DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP, PRIMARY KEY (id) );

## ER-diagram

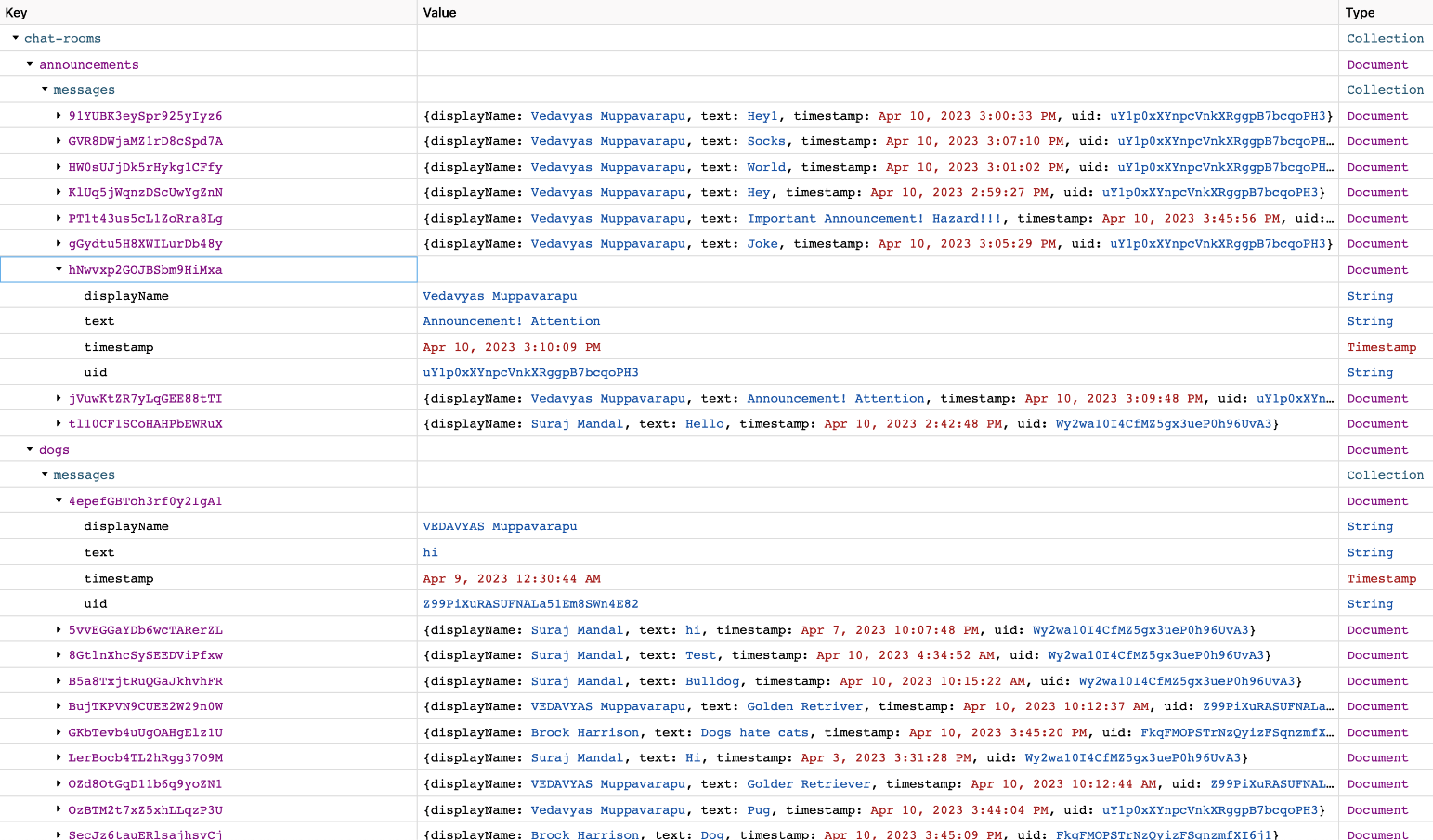


## Firebase with collections as known as tables in relational database.



## Test Scenarios with Sample data

The below screenshots show the statements for sample data insertion into the above created tables.



The below screen captures shows the collections after data insertion.

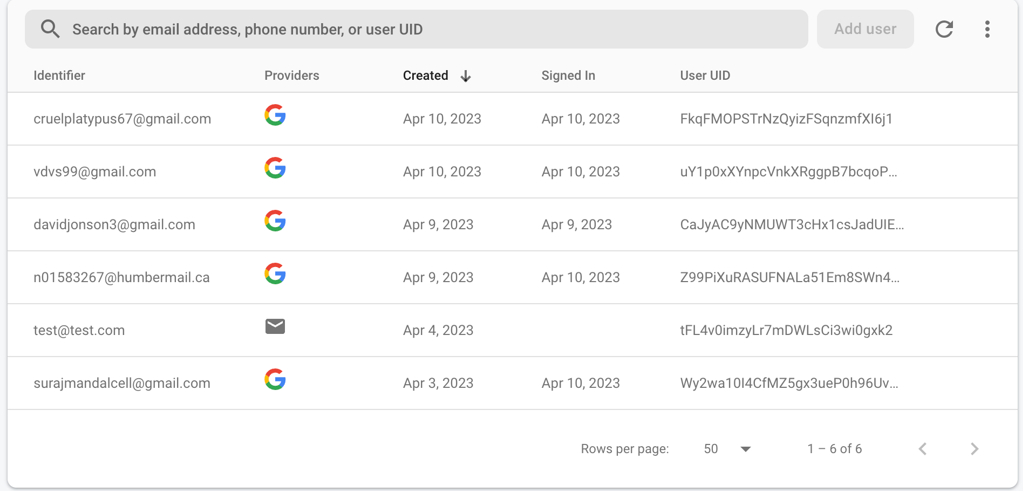






## Test cases for various scenarios

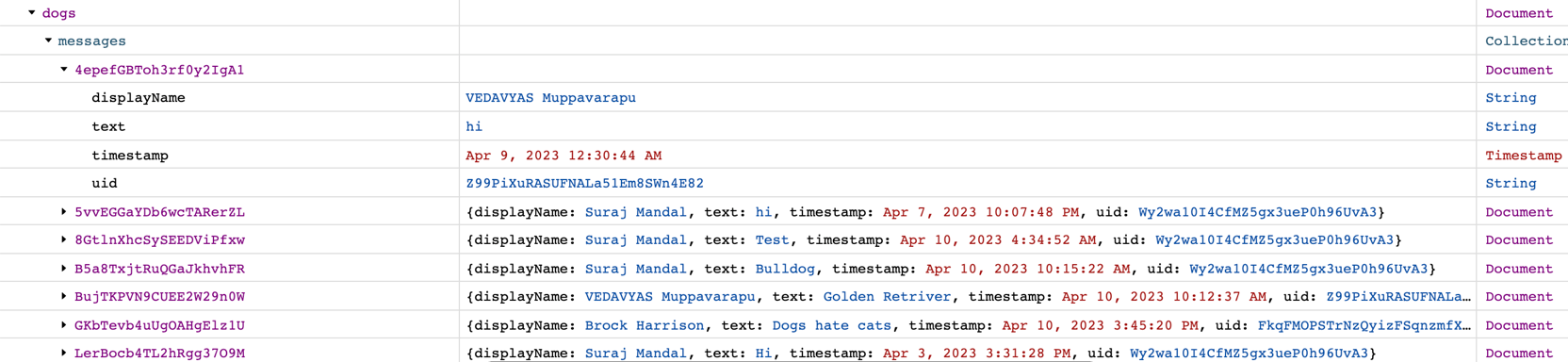
1. User Registration and Login:



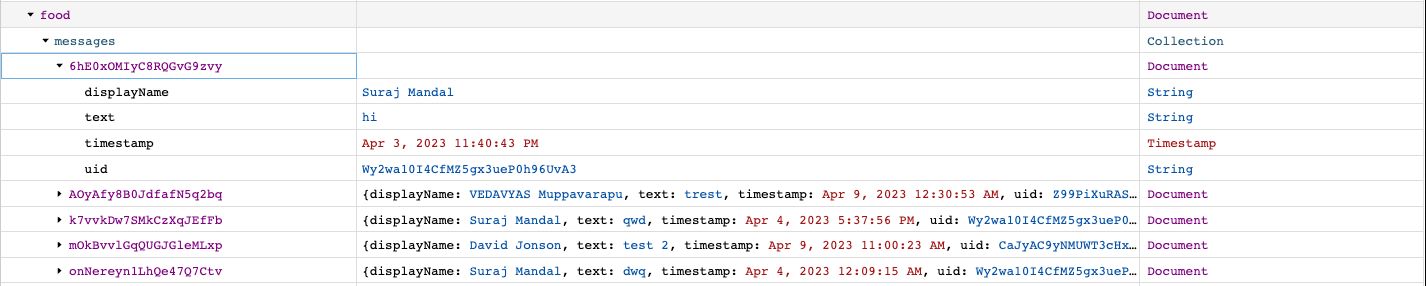
1. announcements screenshot:



1. Messages for dogs collection:



1. Messages for food collection:



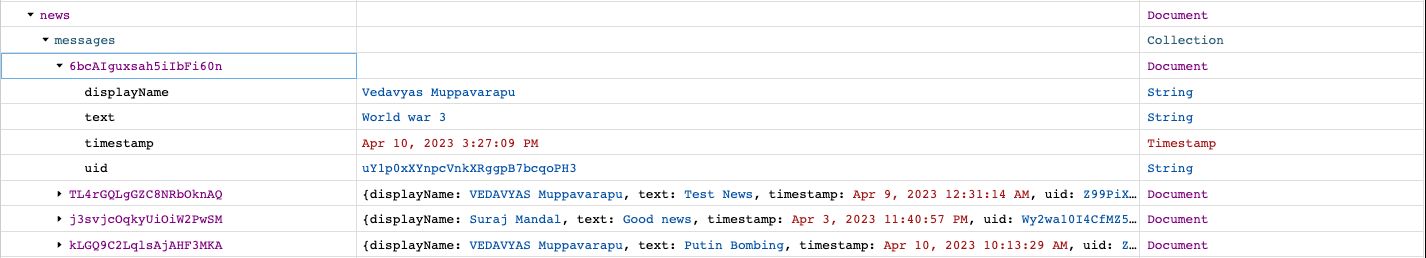
1. Messages for general collection:



1. Messages for music collection:



1. Messages for news collection:



1. Messages for sports collection:



## Team member contribution

**Venkata Narasimha Vedavyas Muppavarapu**

Worked on 3 tables creation, E-R diagram, inserting data to the tables and test case scenarios.

I will be working on project setup and routing integration and implementing the code. Also, will be working on creating UI for foreseeing templates.

**Suraj Mandal**

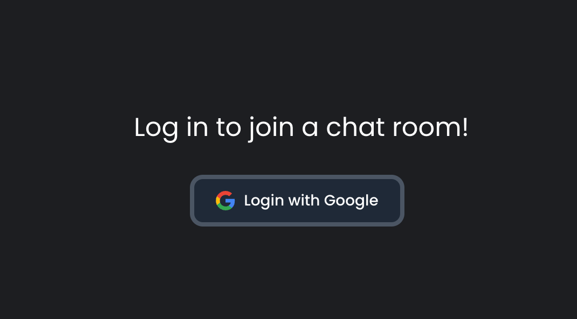
Worked on 5 tables creation, E-R diagram, inserting data to the tables and test case scenarios.

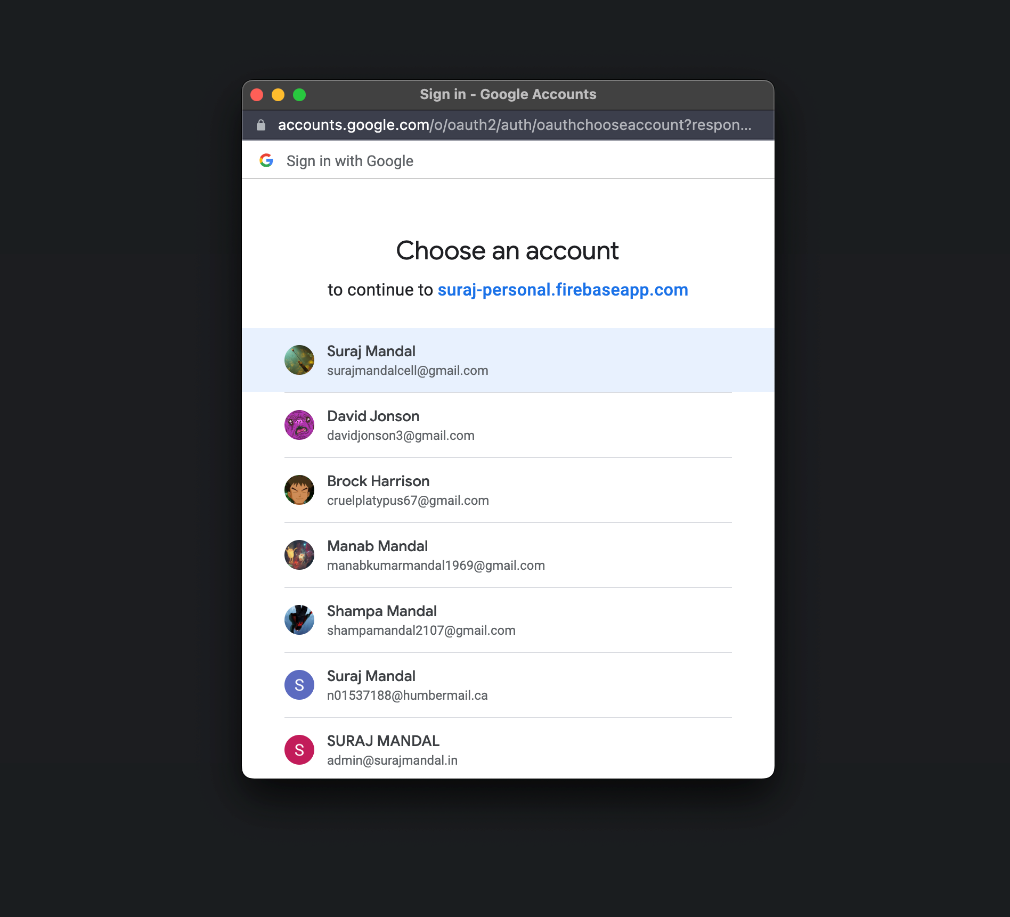
I will be working on connecting database to the application and testing with multiple scenarios and UI development for login page using HTML.

Phase -3:

## Login Page:

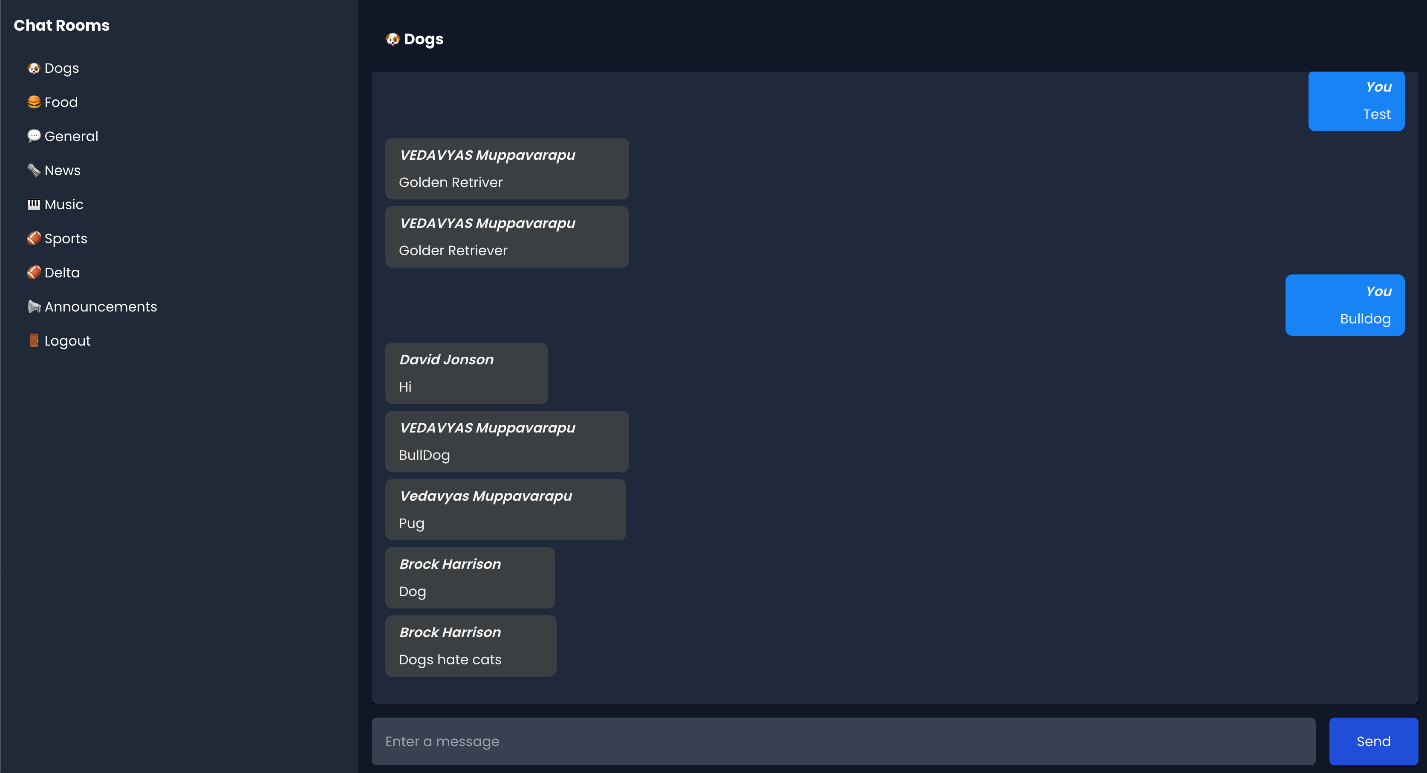
This is the landing page of our chat application. Once the user has created an account, they can log in to the application directly. The login functionality, logs in the user to chat app, the information stored in the firebase database to verify their identity via google authentication.



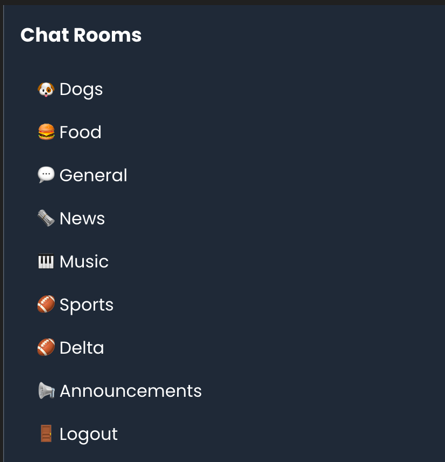


**Valid username and password (Login success)**

If the user's login credentials are valid, the application would create a session for the user and redirect them to chatroom page.

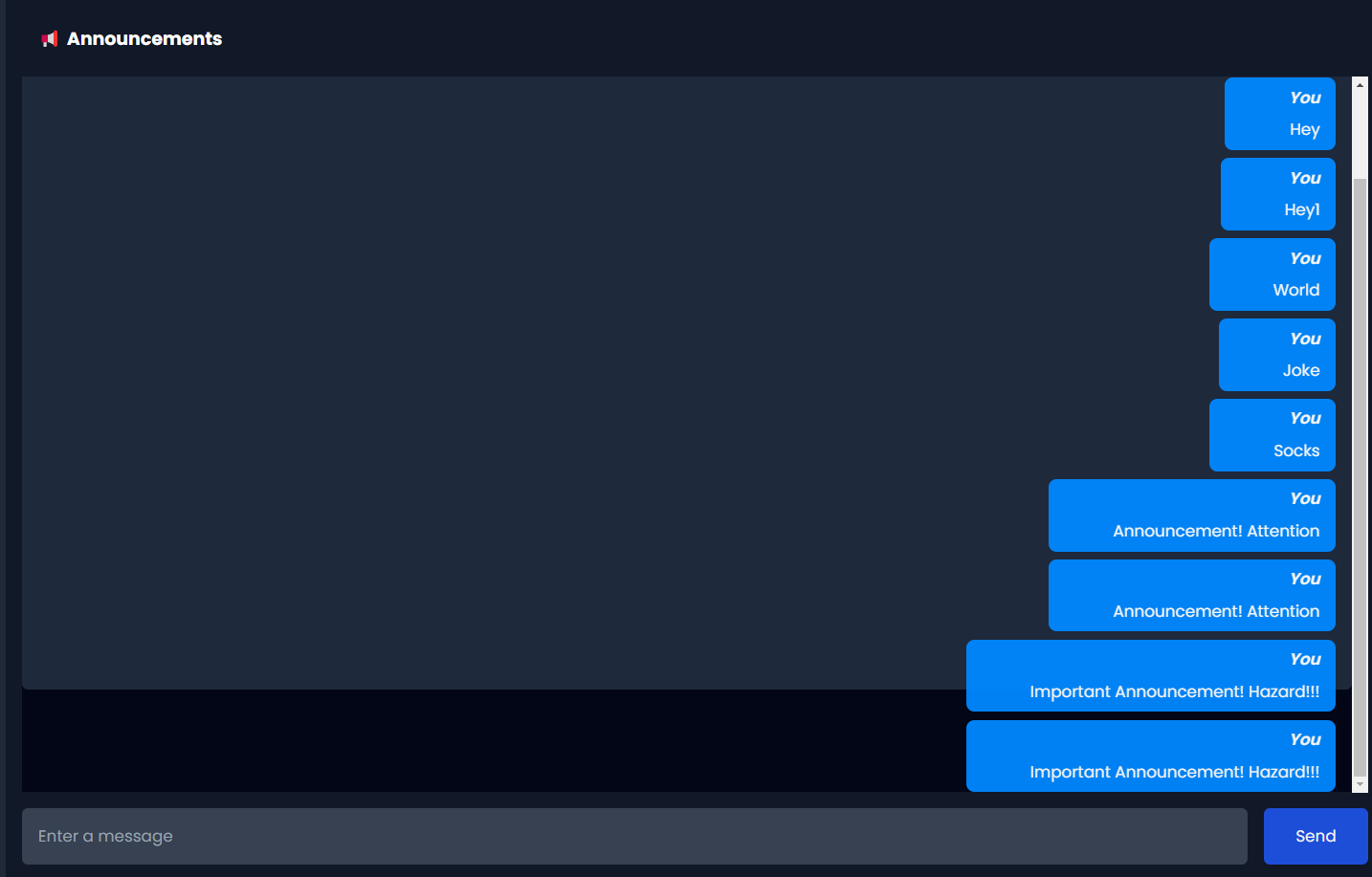


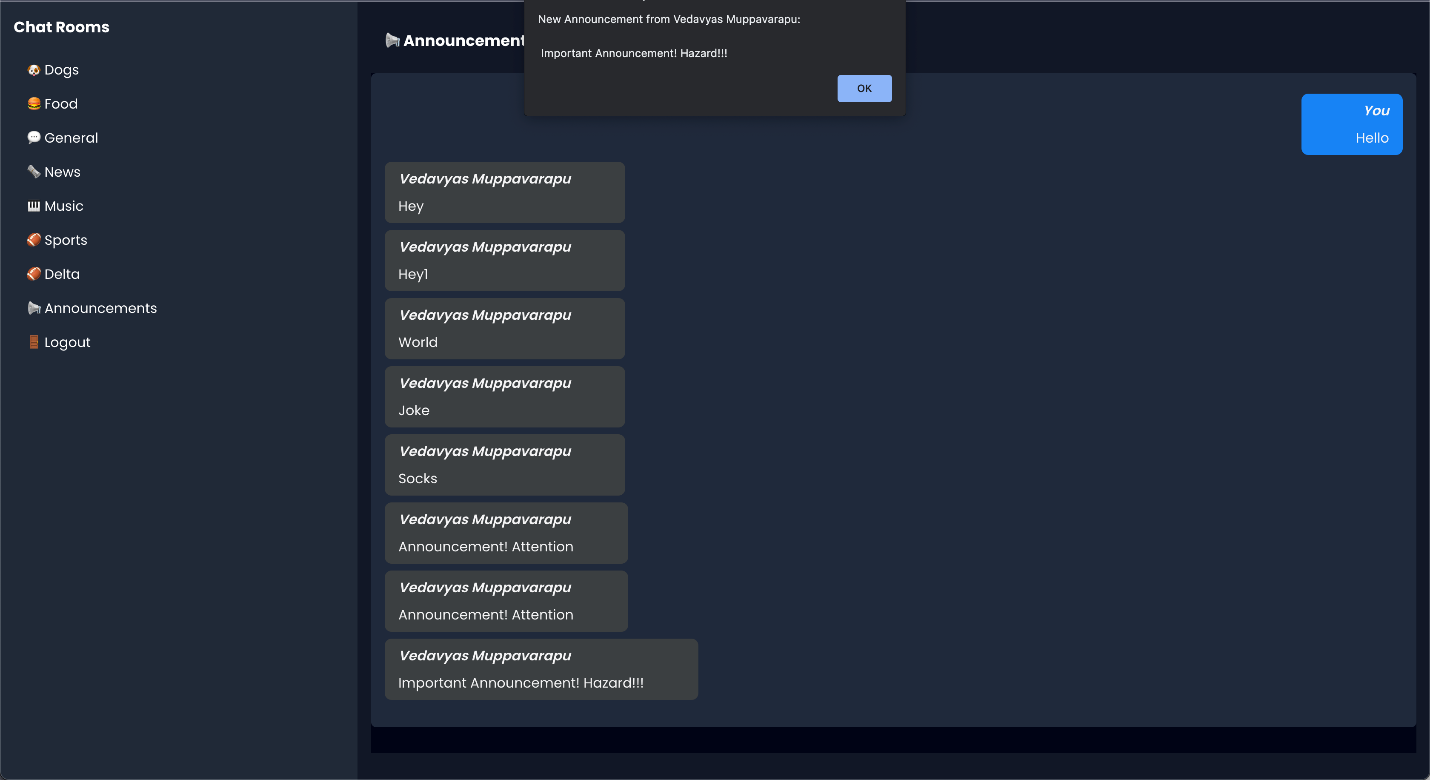
Chat rooms:

****

This will display chat rooms available with announcements that user will collaborate.

## Announcements:





## Changes made in Phase -3:

We have added firebase auth which handles google authentication and authorization. Changed UI design in frontend to look more materialistic. Added logout functionality.

## Team member contribution

**Venkata Narasimha Vedavyas Muppavarapu**

Worked on creating Login UI pages, chat room UI. Will be working with integrating firebase with react app

**Suraj Mandal**

Worked on creating chat rooms, firebase authentication, message queuing. Will be working the flask to host react app

**Phase – 4:**

a) Scenario 1: User Login

In this scenario, the application allows Users can then login using their registered username and password, and the application verifies their credentials with the database.

Scenario 2: Channels/chat rooms

In this scenario, the application retrieves data from the database and displays it in a visual format, such as charts or graphs. Users can interact with the visualizations to gain insights from the data.

Exclusion criteria: The output may fail in the following conditions:

Database connection failure: If the application is unable to establish a connection with the database, it may not be able to perform any operations that require database access, such as registration, login, data entry, retrieval, update, deletion, search, filter, or visualization.

Incorrect or missing database credentials: If the application is provided with incorrect or missing credentials for connecting to the database, it may not be able to establish a connection, and therefore may fail to perform any database-related operations.

Invalid input data: If the users enter invalid or incomplete data in the application's interface, such as incorrect format for username, password, or other fields, the application may fail to store the data in the database or retrieve the correct data from the database.

b) The main application window of the project would typically include buttons or widgets for different functionalities, such as login, chat retrieval. Depending on the user's choice of buttons or widgets, the application may create additional windows or dialog boxes for specific functionalities, such as registration window, login window, chat rooms, chat retrieval window of each channel, announcements from the authorised user. These windows would be logically connected to the main application window and interact with the database tables to perform the respective operations.

c) Exceptions or conditions that may destabilize the application or cause it to crash could include:

Database connection failure: If the application is unable to establish a connection with the database or loses connection during the operation, it may raise an exception and fail to perform the database-related operation. This can be handled by implementing error handling mechanisms, such as try-except blocks, to catch the exceptions and display appropriate error messages to the user.

Incorrect or missing database credentials: If the application is provided with incorrect or missing credentials for connecting to the database, it may raise an exception and fail to establish a connection. This can be handled by validating the credentials before establishing the connection and displaying an error message if they are incorrect or missing.

Invalid input data: If the users enter invalid or incomplete data in the application's interface, such as incorrect format for username, password, or other fields, the application may raise an exception and fail to store the data in the database or retrieve the correct data from the database. This can be handled by implementing data validation mechanisms, such as input checks and sanitization, to ensure that only valid data is stored in the database and retrieved from the database. Additionally, appropriate error messages can be displayed to the user in case of invalid input data.

d) Challenges in integrating Phase-III with Phase-IV of the project may include:

User interface integration: Integrating the user interface of Phase-III with Phase-IV may require coordination and synchronization of different components, such as buttons, widgets, and visualizations, to ensure smooth user experience. This can be achieved by carefully designing the user interface and implementing appropriate event handling mechanisms to handle user interactions

Data consistency: If the data in Phase-III is not consistent or has missing values, it may affect the accuracy and reliability of data visualization in Phase-IV. This can be addressed by implementing data cleaning or data imputation techniques to ensure that the data used for visualization is complete and accurate.

To manage and handle these challenges, thorough testing, debugging, and validation of the integrated phases can be performed. Any issues or discrepancies can be addressed through proper error handling mechanisms, data processing techniques, and user interface design to ensure seamless integration and smooth operation of the project.