



# Helpful Harbor (An AI powered chatting application)

## System Design Document

Ayush Ranjan

IIT (ISM) Dhanbad

BTech. Mining Machinery Engineering (Department of Mechanical Engineering)

21JE0215

## Purpose

The purpose of this document is to provide a detailed system design for the various components that comprise the **Helpful Harbor (An AI powered chatting application)**. While the Software Architecture Document provides a high level structural view of the application and how the application interacts with external systems, this document focuses on just the system itself, and how its various software components are designed and how they interact with one another.

This document is intended to help the recruitment team determine how the system is structured at a detailed level.

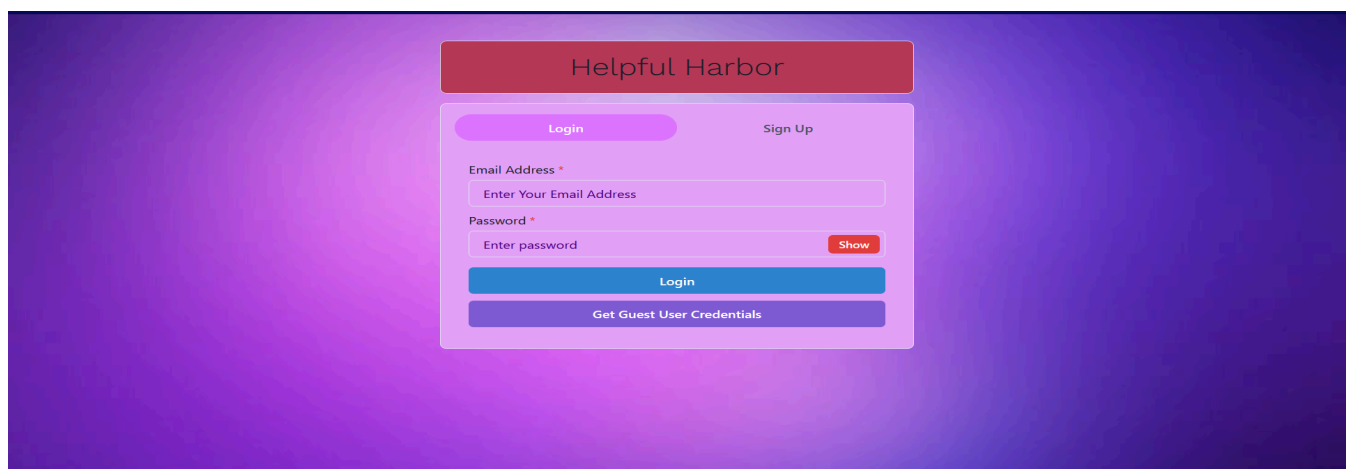
## Overview

In our everyday lives, we often want to help others or make things better for our community. But sometimes, our busy schedules get in the way, and we can't do the good deeds we wish to. This app helps with that. It connects you with NGOs or groups of people who are ready to lend a hand.

For instance, if you come across a stray dog in need of help, you can use the app to contact a local animal care NGO. They'll take care of the dog's treatment. This way, you can be a responsible citizen without giving up your daily routine.

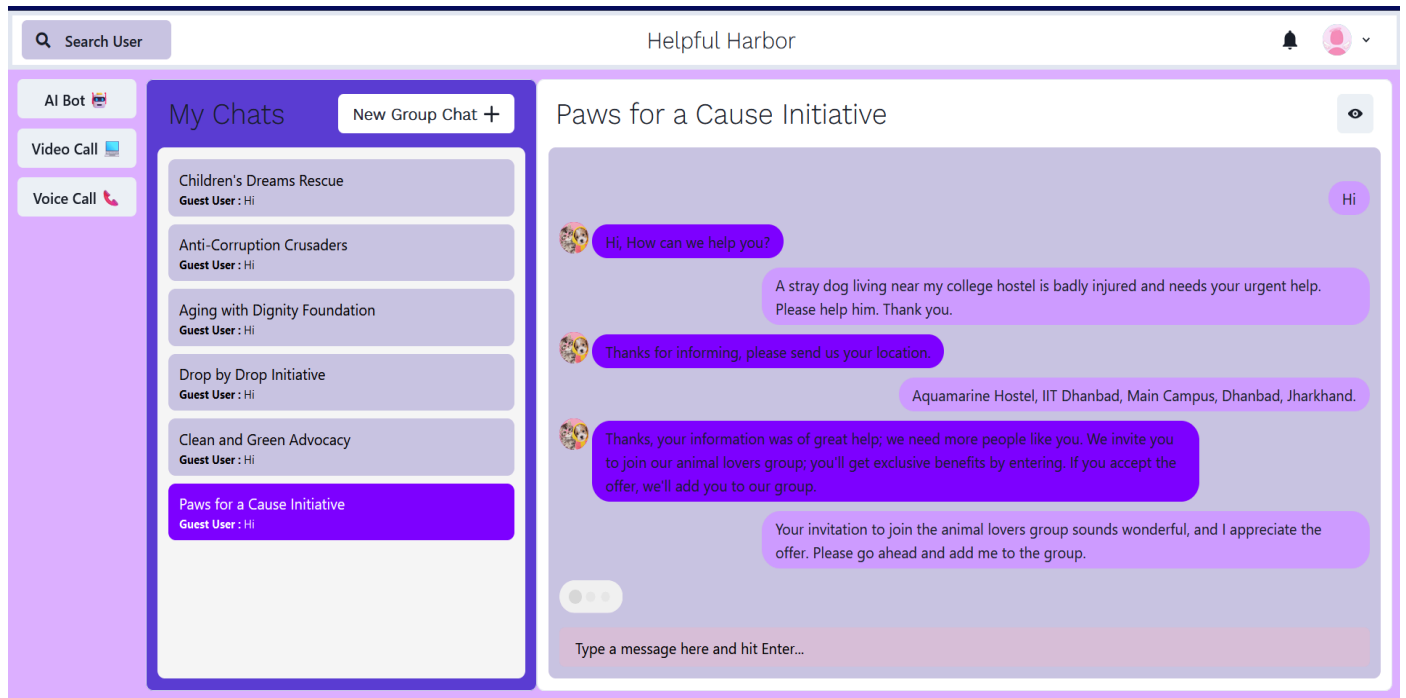
## Goals

1. Login and Signup tabs for new users or NGOs:

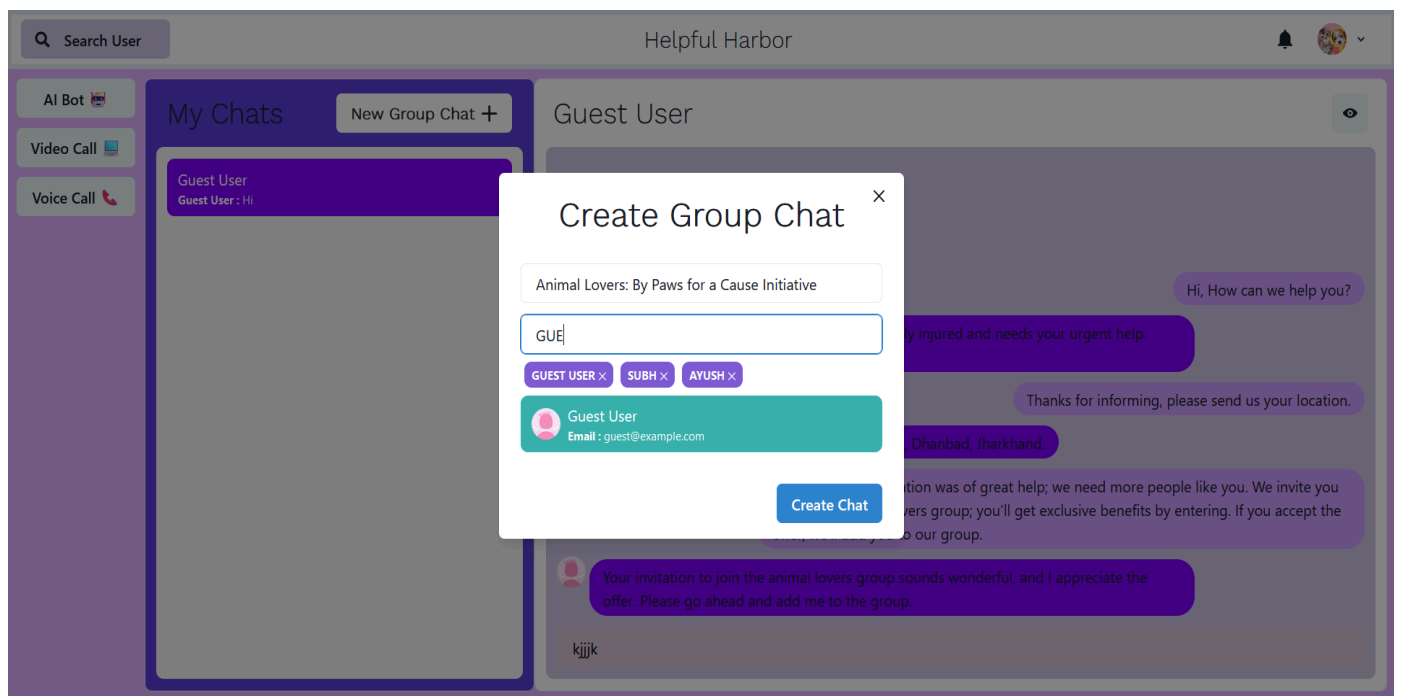


The screenshot displays the 'Helpful Harbor' login and signup interface. At the top, a dark red header bar contains the text 'Helpful Harbor'. Below this, a light purple box contains the login and signup forms. The box has two tabs: 'Login' (selected) and 'Sign Up'. The 'Login' form includes fields for 'Email Address \*' (with placeholder text 'Enter Your Email Address') and 'Password \*' (with placeholder text 'Enter password' and a 'Show' button). Below the password field is a blue 'Login' button. At the bottom of the box is a purple button labeled 'Get Guest User Credentials'.

2. The leading chat page shows all the features to the user in a clean and intuitive UI.



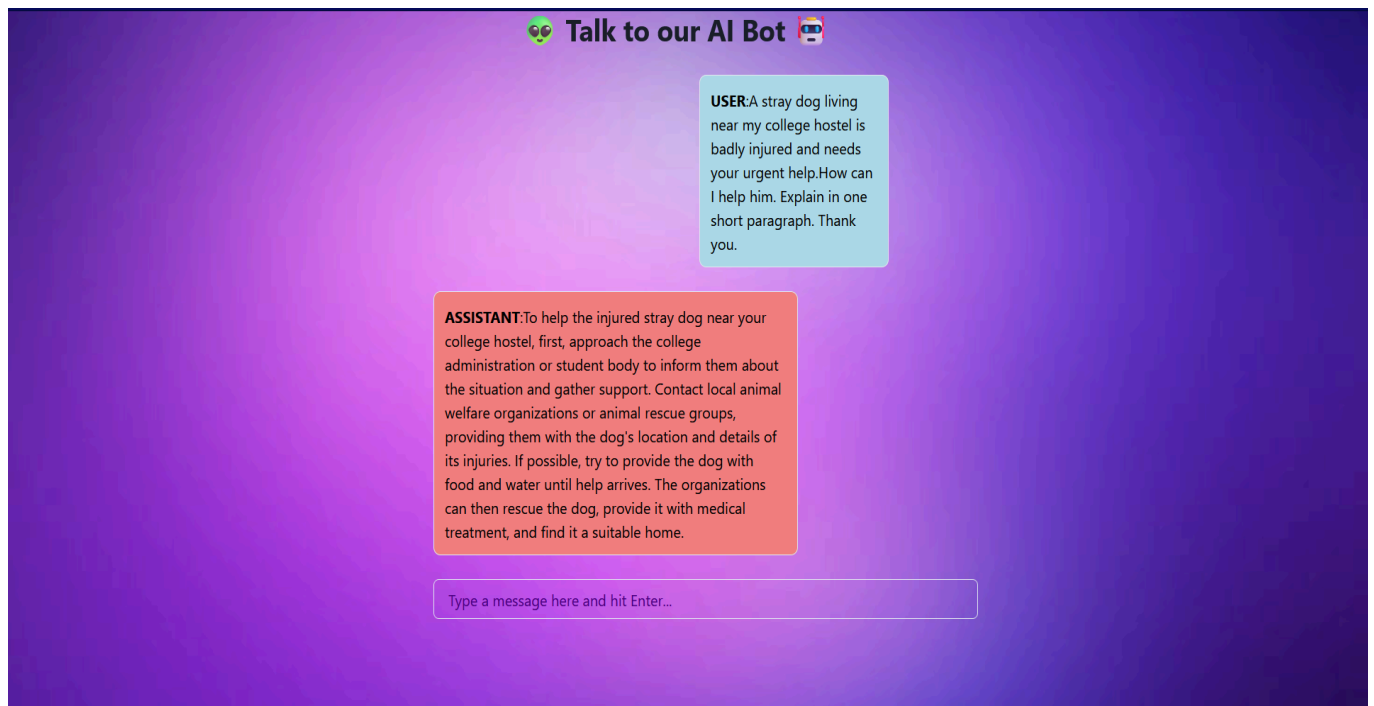
3. Group chat feature, to let the admin add new members. These groups can be used by NGO people to keep in contact with the people who are willing to help them.



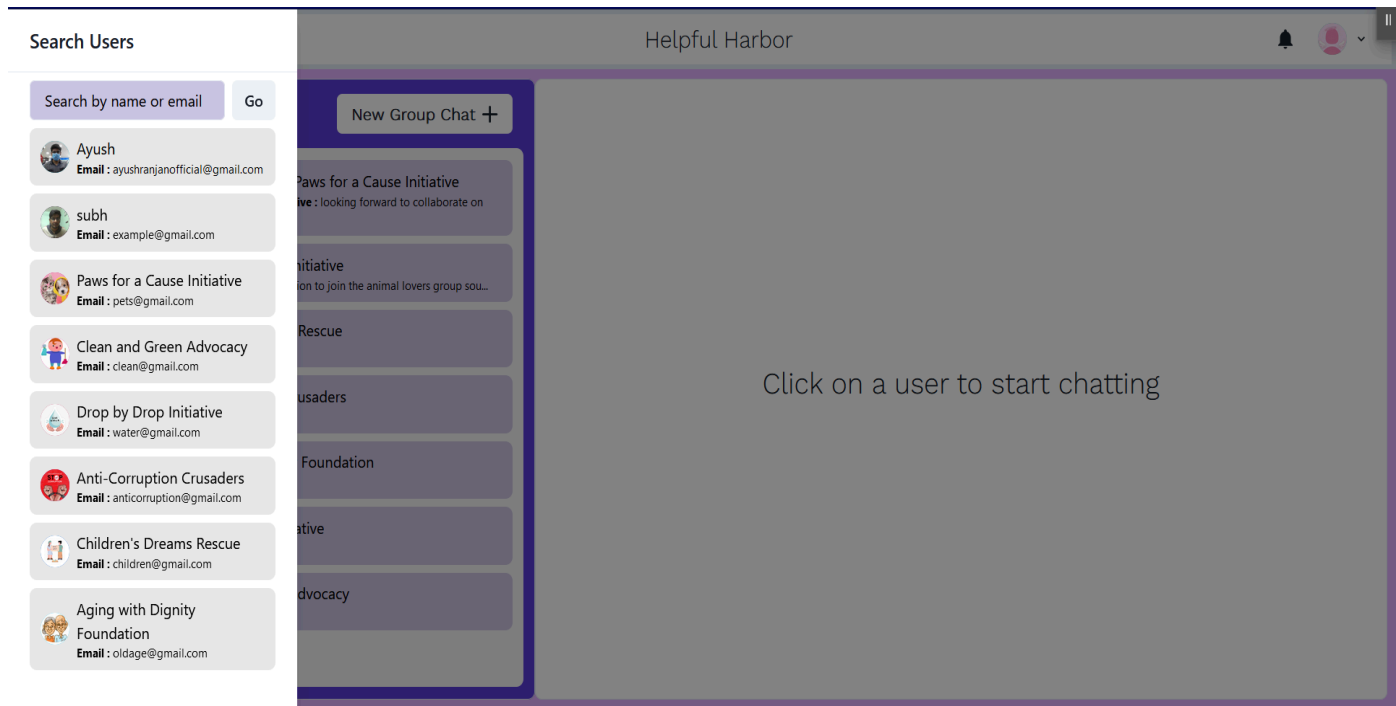
4. Notification System: To make users aware of new messages.



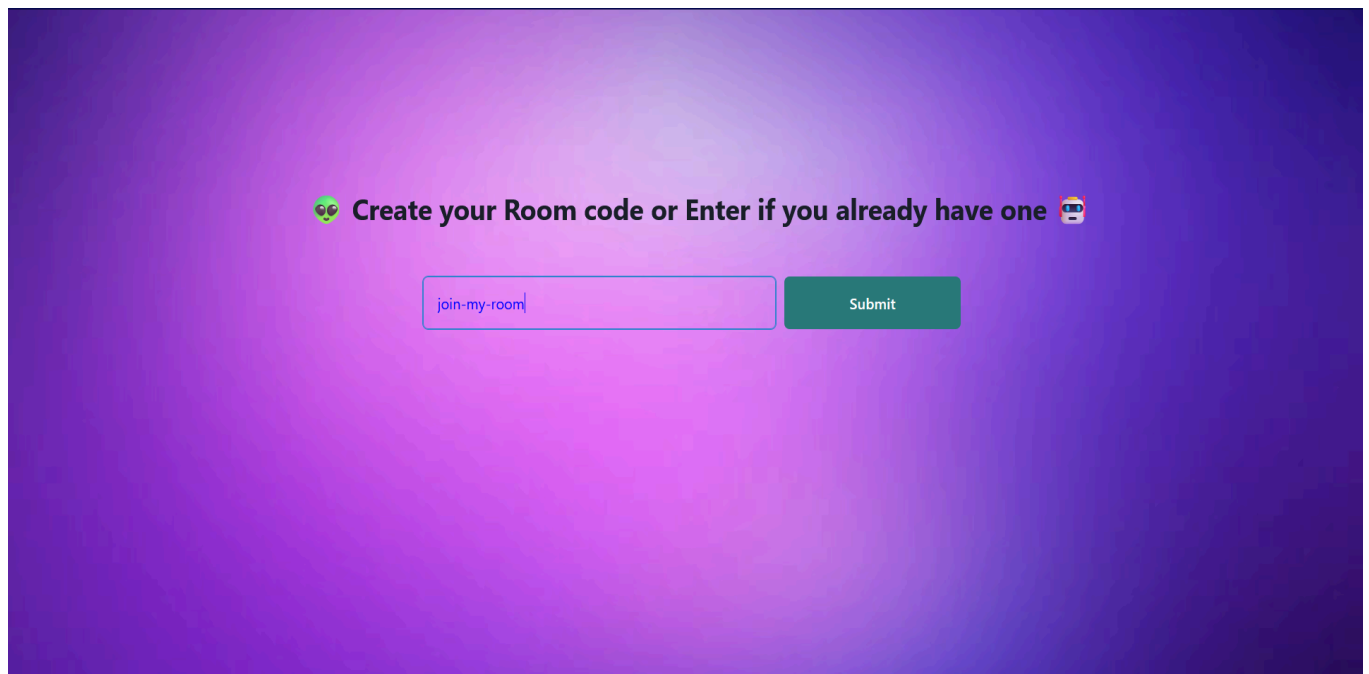
5. AI Chatbot for faster assistance; in case of emergency, AI can assist in finding solutions to the problems.

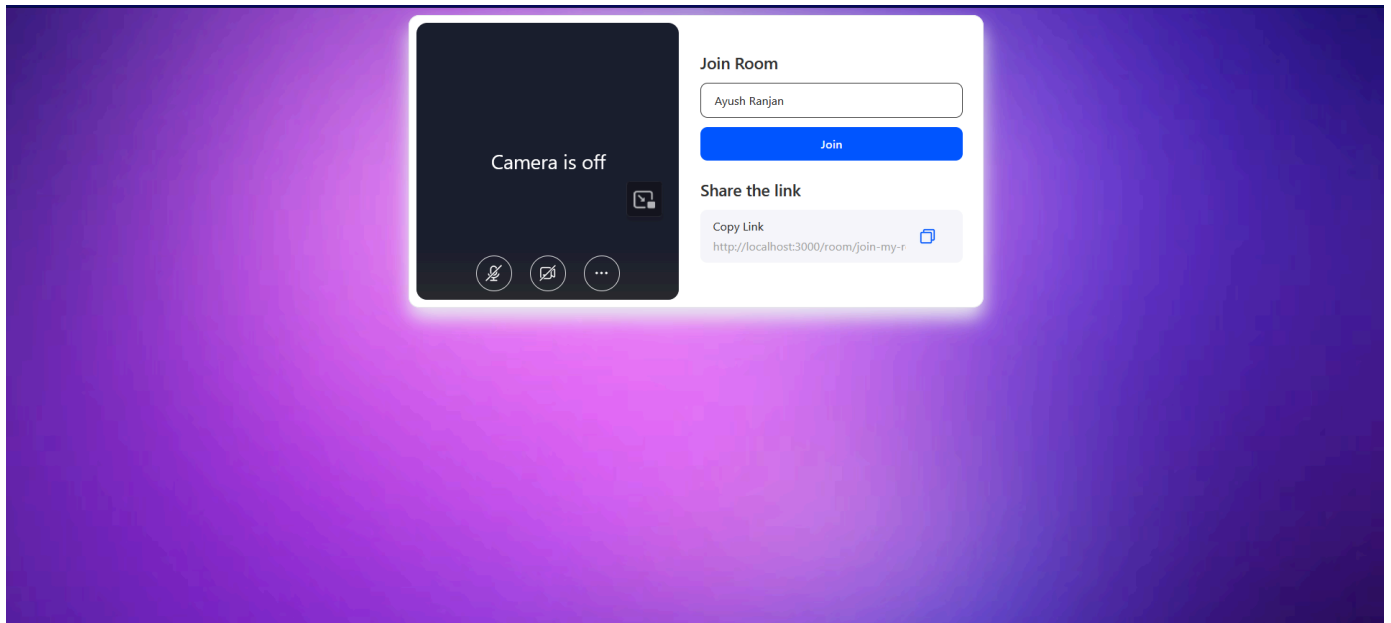


6. Search bar for searching and adding new community members to users chatbox.

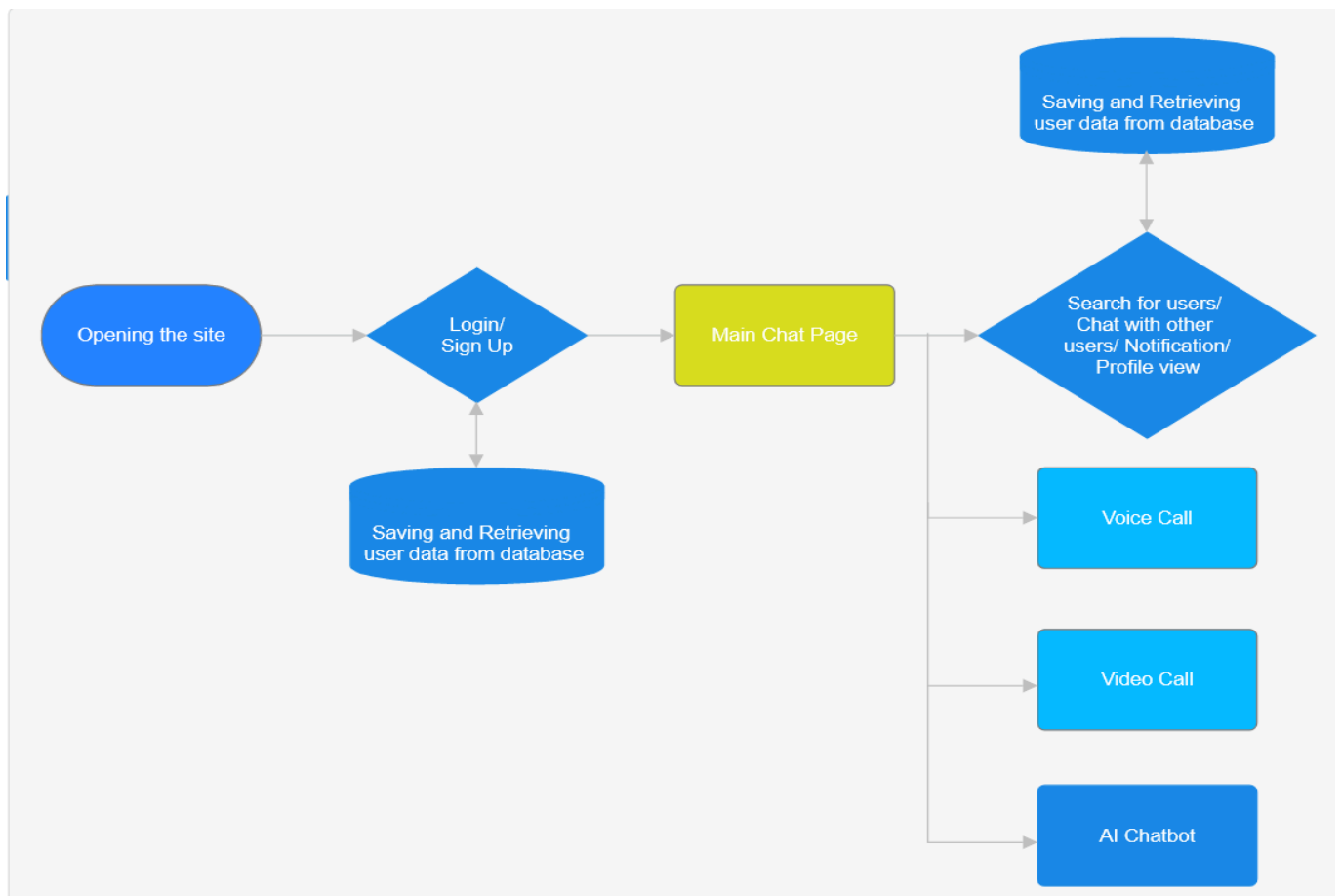


7. Video and Audio Chat pages for better exchange of information.





## System Flow Diagram:



## User Roles

The following user classes represent the four main roles that users have when interacting with the system, which is done through the use of a desktop or mobile computer:

### I. Help Seeker

User who has encountered a problem and is seeking help. He/She has three options to make:

- 1) For faster help, try to explain your problem to the AI Bot.
- 2) If you're in a group or connected with an organization that can assist you with your issue, feel free to ask for help within the group. Groups are reserved for members, so you can count on quicker assistance there.
- 3) Search for a relevant organization and text them or invite them to your voice/video call room.

### II. Help Provider

Users who are part of any organization/NGO, which is willing to help others in some way:

- 1) For a community through a group of members who are willing to help you in this cause.
- 2) Help the users, who have texted you for help.
- 3) Ask the help seekers to be a part of the Group, if they liked what the Organization is doing.

## Library/Services/SDK used:

1. **Express:**
  - For my React application, I've used Express to create a server that serves the React application, handles API requests, and interacts with a database.
2. **Nodemon:**
  - A utility that monitors changes in your server-side code and automatically restarts the server when changes are detected. It simplifies the development process by eliminating the need to manually restart the server every time you make changes to your code.
3. **Dotenv:**

- Dotenv is used to load environment variables from a `.env` file into the application's process.env. It allows you to store sensitive configuration settings like API keys or database connection strings outside of your codebase, enhancing security.
4. **Chakra UI:**
    - Chakra UI is a component library for building React applications with a focus on accessibility and developer ergonomics. It provides pre-designed UI components and styling to help streamline the development of your React application's user interface.
  5. **Chakra UI Icons:**
    - Chakra UI Icons is an icon library designed to work seamlessly with Chakra UI. It offers a collection of customizable icons to enhance the visual appeal and functionality of your UI components.
  6. **React-Router-DOM v5:**
    - React-Router-DOM is a library for handling routing in React applications. It allows you to create dynamic, single-page applications with different views, URLs, and navigation without full-page refreshes.
  7. **Axios:**
    - Axios is a promise-based HTTP client for making API requests. It simplifies sending HTTP requests from your React application to a server or external APIs and handles responses in a more intuitive way.
  8. **Mongoose:**
    - **Use:** Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It simplifies database interactions by providing a structured way to define data models, validate data, and perform operations on a MongoDB database.
  9. **Colors:**
    - The Colors library helps change the colors of terminal outputs. It improves the readability and aesthetics of terminal logs and outputs during development.
  10. **Express-Async-Handler:**
    - Express-Async-Handler is a middleware for handling asynchronous operations in Express routes. It simplifies error handling and makes it easier to write asynchronous code within your Express route handlers.
  11. **JSON Web Token (JWT):**
    - JWT is a standard for creating tokens that can be used for user authentication and authorization. It helps secure your React application by verifying the identity of users and authorizing access to specific resources or routes.
  12. **Bcryptjs:**



- Bcryptjs is a library for hashing passwords before storing them in a database. It enhances security by encrypting user passwords, making it more challenging for unauthorized users to access sensitive information.

**13. Cloudinary:**

- Cloudinary provides an API for managing and serving images and other media files. It simplifies the process of uploading, storing, and serving user-generated content such as profile pictures in your React application.

**14. Socket.IO:**

- Socket.IO is a library for adding real-time, bidirectional communication between the server and clients in a web application. It enables features like real-time chat, notifications, and updates in your React application by maintaining a persistent connection between clients and the server.

**15. OpenAI:** Provides API services for AI integration.

**16. Zegocloud:** Provides services for Voice and Video calling features.

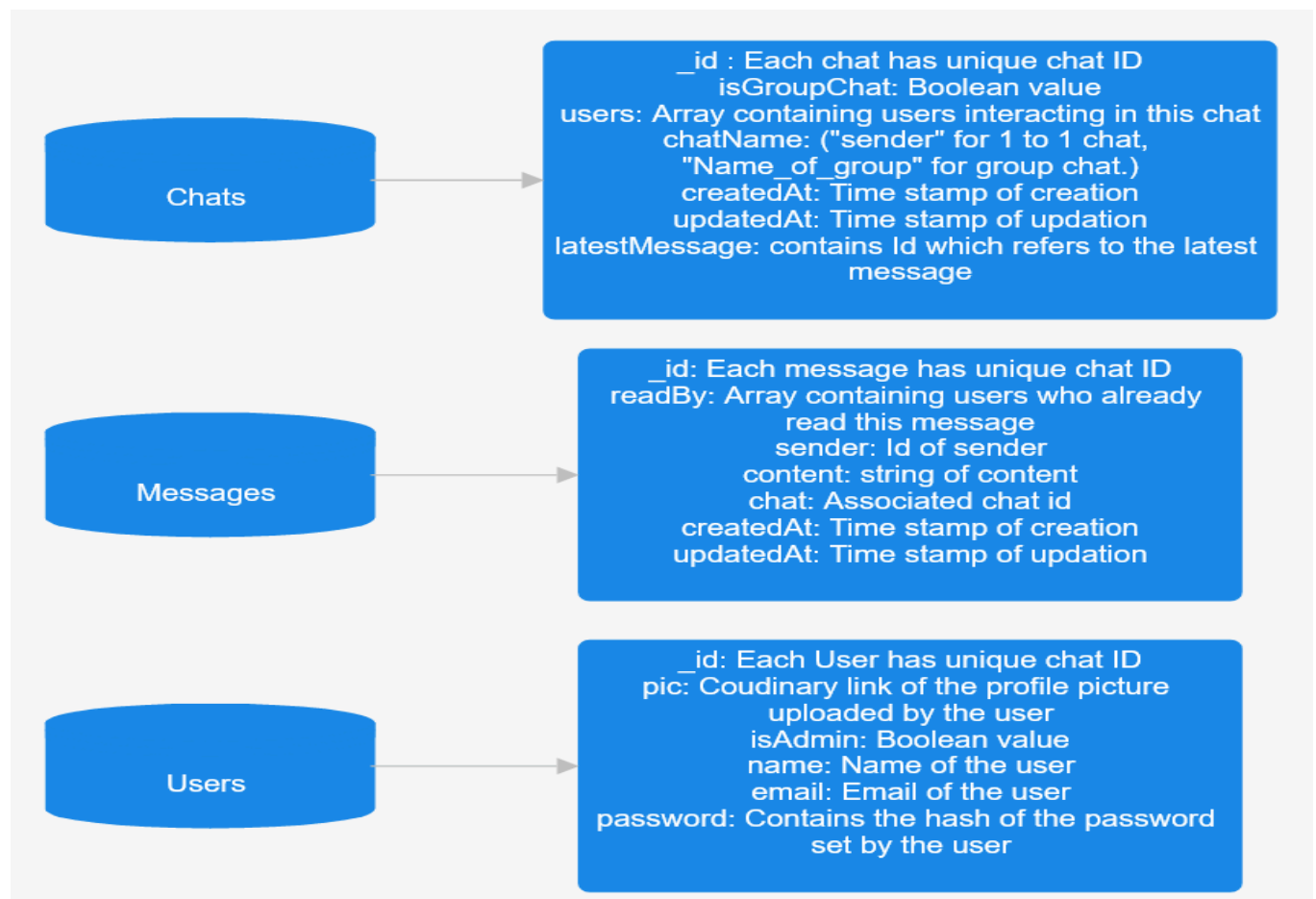
## Database:

Here I have used MongoDB, a NoSQL database, which is a more suitable choice for building chat applications when compared to traditional SQL databases. Here's a detailed comparison explaining why MongoDB is a good fit for this use case:

Property	MongoDB(NoSQL)	SQL
1) Rapidly growing datasets and high write load of the chat application.	MongoDB is designed to scale horizontally with ease. It supports auto-sharding, making it a better choice	Traditional SQL databases can be challenging to scale horizontally when the application experiences increased chat activity. This often requires complex sharding and partitioning.
2) varying attributes of messages like sender, timestamp, and content.	MongoDB employs a flexible, schema-less, document-based data model.	SQL databases use a structured, tabular data model with predefined schemas. This can be rigid and less flexible.
3) Instant message delivery and updates.	MongoDB excels at real-time data storage and retrieval	SQL databases are not optimized for real-time data updates and retrieval.

		Queries can become slow as the dataset grows.
4) Simplification in working with chat data.	MongoDB stores data as JSON-like documents, which closely resemble the data structures used in modern programming languages.	SQL databases typically store data in tables, which can be less intuitive when working with semi-structured data like chat messages.
5) Development Complexity	MongoDB's flexible schema makes it easier to adapt to changing requirements without major disruptions to the application.	may require extensive schema changes as your chat application evolves, leading to more development effort.

## Data Description:



## API Services:

### 1) Interacting with OpenAI

```
app.post("/text", async (request, response) => For  
posting the prompt by user, to get the response  
from openAI.
```

### 2) Interacting with chats

```
router.route("/").post(protect, accessChat); => For  
accessing chats  
  
router.route("/").get(protect, fetchChats); => For  
fetching chats  
  
router.route("/group").post(protect,  
createGroupChat); => For creating group chats  
  
router.route("/rename").put(protect, renameGroup);  
=> For Renaming group chats  
  
router.route("/groupremove").put(protect,  
removeFromGroup); => For removing from group  
  
router.route("/groupadd").put(protect, addToGroup);  
=> For adding to group
```

### 3) Interacting with Messages:

```
router.route("/:chatId").get(protect, allMessages);  
=> Get a message using chatID  
  
router.route("/").post(protect, sendMessage);  
=> post a message
```

### 4) Interacting with users:

```
router.route("/").get(protect, allUsers); =>  
Get user details  
  
router.route("/").post(registerUser); => post  
user details  
  
router.post("/login", authUser); => Authorising  
user while login process.
```

**Data for .env file (For recruitment team for testing will delete later for security reasons) :**

#### 1) Inside the .env file in root directory:

```
PORT = 5000
```

```
MONGO_URI =  
mongodb+srv://ayushranjanofficial27:ayush.1  
970@cluster0.jhiysef.mongodb.net/?retryWrit  
es=true&w=majority  
  
JWT_SECRET = ayush  
  
API_KEY =  
sk-USOXSIZsKG8jTXGyM1YOT3B1bkFJp3NzS1sdaP34  
pr5Jrd6Z  
  
NODE_ENV=production
```

## 2) Data for 2nd .env file inside frontend folder:

```
REACT_APP_APP_ID=1098023691  
  
REACT_APP_SERVER_SECRET=2cc5e5948c4b2955e32  
8d4f758625087
```