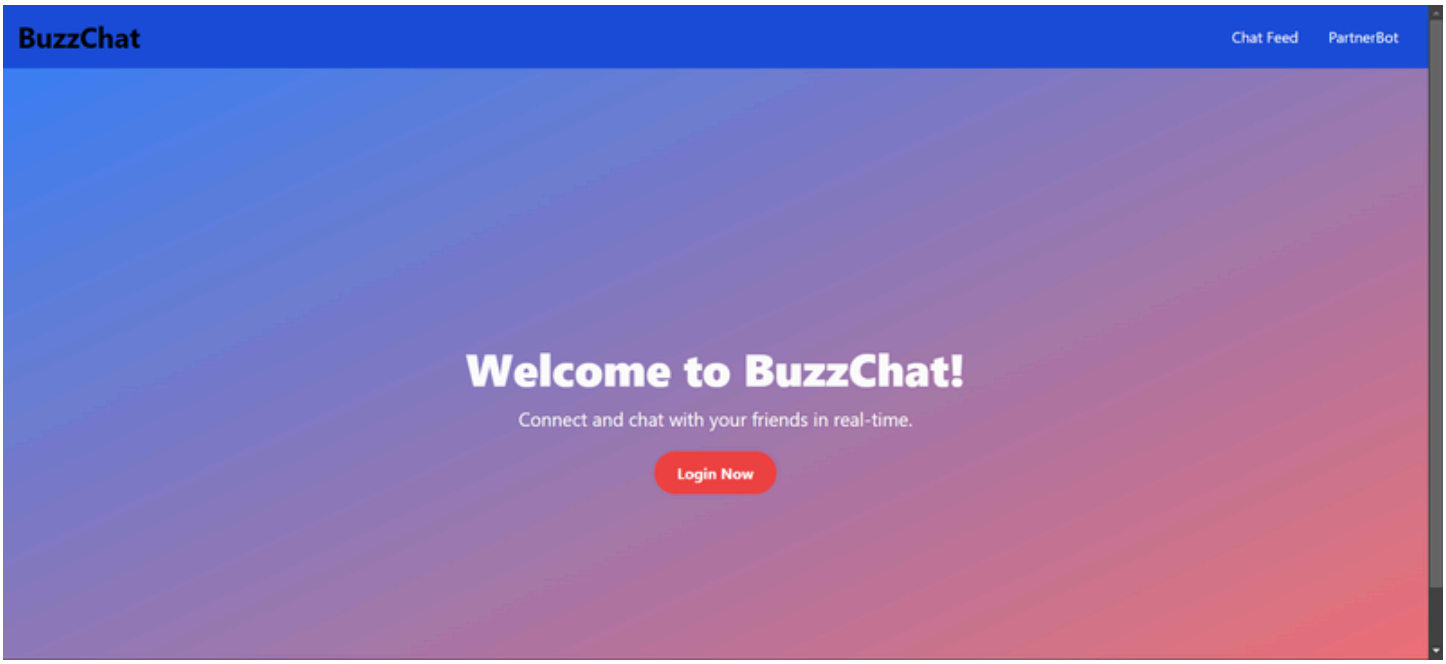
I’m Beside You

SOFTWARE ENGINEER ROLE

THARAKADATTA D HEGDE

B22CS102

IIT Jodhpur



Introducing

BUZZCHAT

Messaging Service

Project Overview

BuzzChat is a real-time messaging service designed to provide a seamless and intuitive communication experience for users. It includes essential features such as **one-to-one**

messaging, **group chat functionality**, and **smart authentication** mechanisms to ensure secure and personalized interactions.

In addition to standard messaging, BuzzChat also offers an **emotional friend chatbot** that engages with users in meaningful conversations, adding a supportive and interactive element to the app. The user-friendly interface ensures smooth navigation and allows users to easily discover and converse with new members of the platform, promoting social interaction and engagement.

BuzzChat is built with a focus on real-time communication, combining cutting-edge technology with a simple and engaging user experience. It is tailored to cater to individuals seeking both casual and purposeful conversations, providing tools that make connecting with others easy and enjoyable.

Target Audience

BuzzChat is designed for individuals who seek a smooth and efficient real-time communication platform. Its core audience includes:

- **Young adults and professionals** looking for seamless personal and group communication.
- **Social enthusiasts** who want to meet and engage with new users through one-to-one messaging or group chats.
- **Individuals in need of emotional support**, who can interact with the friendly and empathetic chatbot to express their feelings.
- **Communities and teams** requiring an easy-to-use platform for managing conversations and keeping up with social or work-related discussions in real-time.

Problem Statement

In a world where communication is key to maintaining relationships and building networks, existing platforms often lack the ability to offer real-time, secure, and engaging interactions in a single app. Users face challenges such as:

- **Inconsistent real-time communication**, where messages may be delayed or not delivered in the expected sequence.
- **Limited social discovery**, as many platforms restrict interaction to existing contacts and don't promote new connections.
- **Lack of emotional support**, where users feel isolated in digital conversations without an empathetic touch.
- **Cumbersome user interfaces**, making it hard to navigate messaging and group chat features, hindering effective communication.

Tech Stack Used

- **Frontend**:

    1) Next.js

2) JavaScript

3) TypeScript

- **Backend**:

  1) Node.js

  2) NextAuth.js

- **Database**:

  1) MongoDB

- **Third Party Integration**:

  1) Pusher

- **Chatbot**:

  1) Gemini API

Component-wise Implementation of BuzzChat (Following Atomic Design Principles)

BuzzChat has been implemented using **Atomic Design principles**, ensuring a modular and reusable component structure. This approach promotes **reusability**, **maintainability**, and **scalability**, which in turn reduces code duplication and improves overall readability.

1. **Atoms**

Atoms represent the smallest and most fundamental UI elements in the application.

- **Button**: A generic button component used throughout the app. Variations such as secondary, danger, and disabled states are handled via props.
- **InputField**: A reusable input field that supports dynamic configurations such as placeholder, type, and validation states.

2. **Molecules**

Molecules are composed of multiple atoms working together.

- **AuthForm**: Combines input fields and buttons to create the user registration and login form. This form can be reused in different authentication contexts, reducing the need for multiple form components.

3. **Organisms**

Organisms are more complex UI components that consist of molecules and atoms. These components often serve a specific section or feature of the application.

- **ChatWindow**: Comprises a list of MessageBubble molecules, an input field, and a send button for messaging. This component efficiently reuses smaller components to create the chat interface.

4. **Templates**

Templates define the layout of the application, structuring organisms, molecules, and atoms together into cohesive pages or sections.

- **ChatPage**: A template that structures the Sidebar and ChatWindow components. It defines the layout of the chat experience by positioning these components side by side.

5. **Pages**

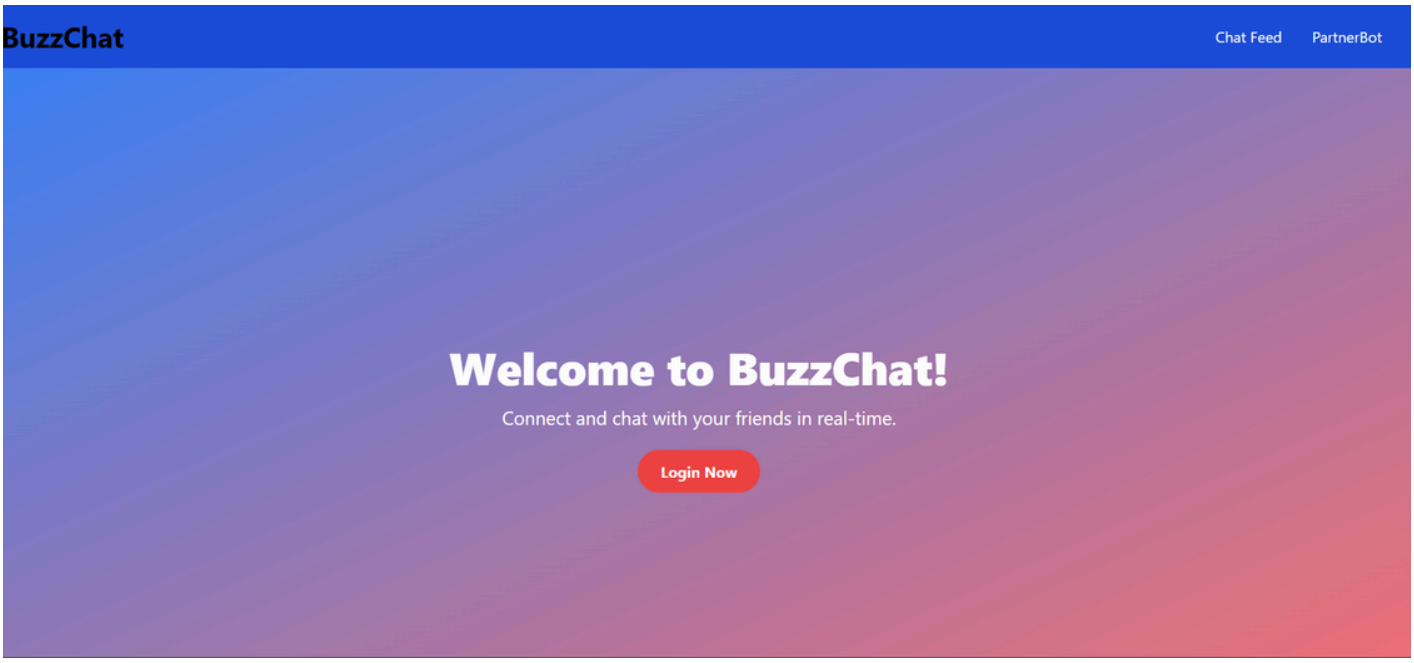
Pages are the highest level in the Atomic Design hierarchy and represent the final rendered output.

- **LoginPage**: Renders the AuthPage template with login-specific data and layout.

Frontend Design

- **Home Page**

The home page is minimalistic and simple, designed to provide clarity and avoid overwhelming the user. It acts as an introduction to the BuzzChat platform and directs users to relevant sections like authentication and chat. The page gives a brief overview of the messaging service, highlighting its features like one-to-one messaging, group chat, and the emotional chatbot. The simplicity ensures users are not confused and can easily navigate to different parts of the platform.



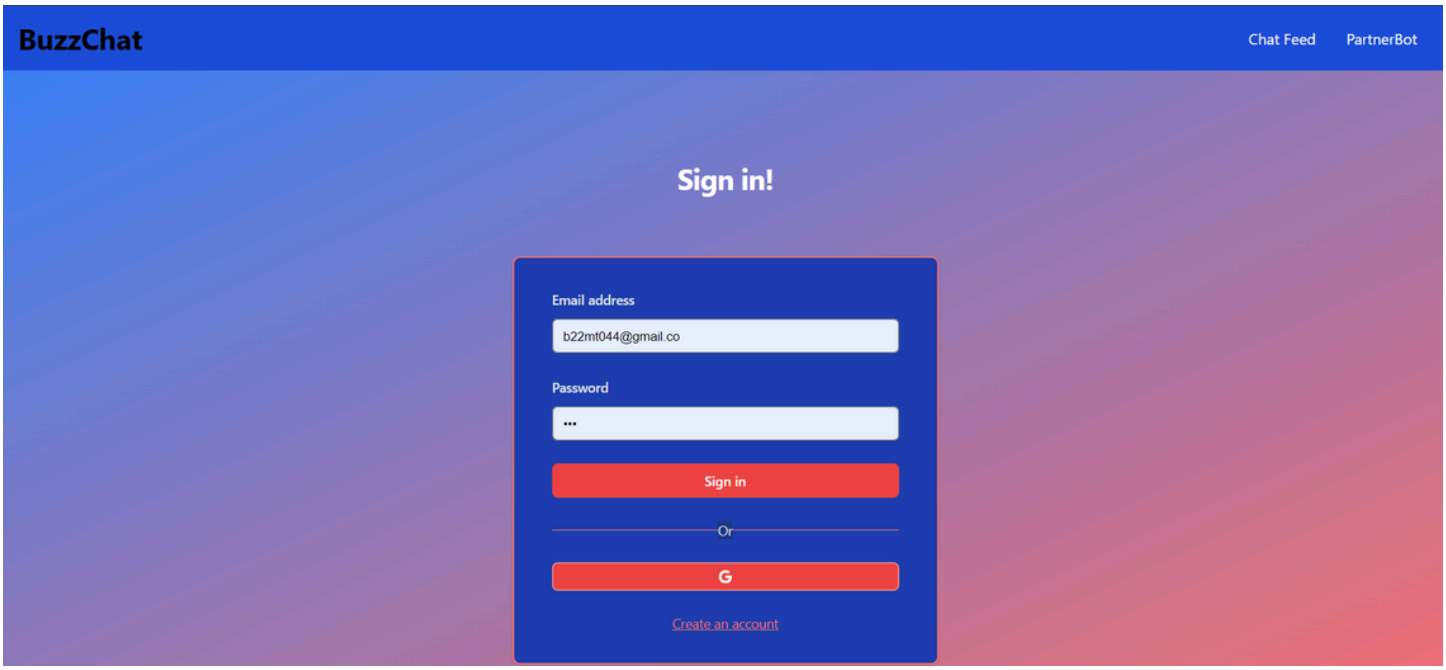
- **Authentication**

Path: "/chat"

Authentication is crucial for any user who wants to access the chat functionalities, ensuring privacy and security of conversations. BuzzChat supports two types of authentication:

1. **Manual Registration**: Users can sign up using their email and password, which is then securely stored in **MongoDB**.
2. **Google Authentication**: For a seamless experience, users can authenticate using their Google account through OAuth, allowing quick access without the need to manually register.

Once authenticated, the user can access the messaging features. The authentication flow prevents unauthorized users from accessing chats and ensures secure user sessions.

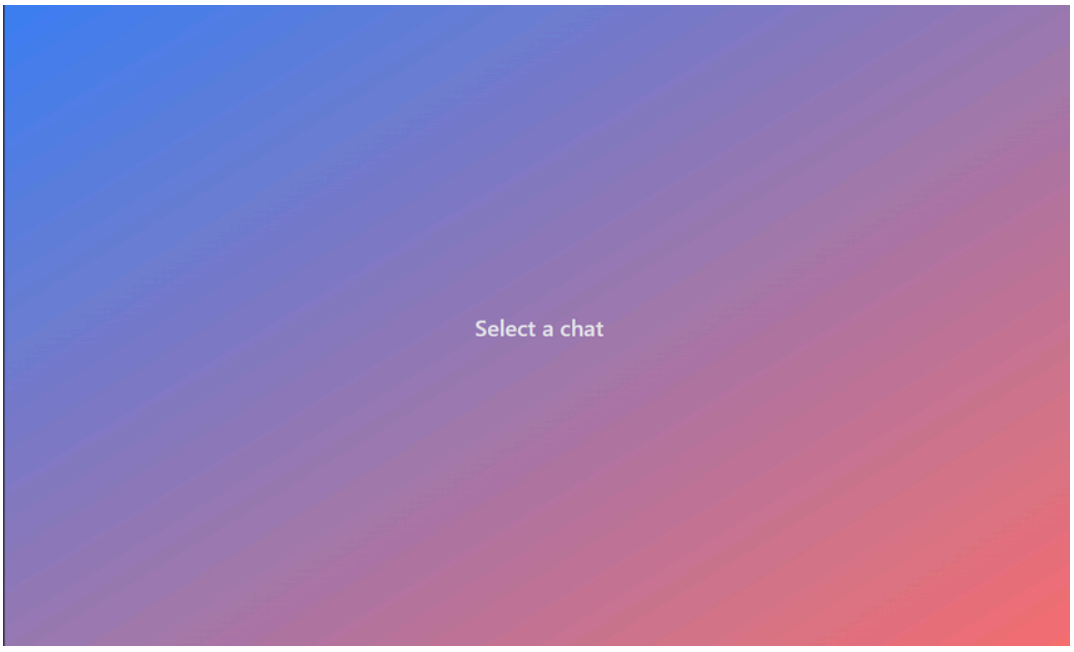


- **"Messages" Section in Chat Page**

The messages section allows users to:

- **View Previous Messages**: Users can see a list of all previous conversations, helping them maintain a history of their interactions.
- **Real-Time Updates**: New messages are delivered instantly using **Socket.io**, keeping users updated in real-time.
- **User-Friendly Chat Panel**: Users can click on a particular contact to open a conversation. The interface is designed to be intuitive, allowing users to navigate between conversations easily.

The message section is built with simplicity and ease of use in mind, making it easy for users to manage and engage with their ongoing chats.



- **"Users" Section in Chat Page**

The users section provides the ability to explore all registered users on the platform:

- **Browse Users**: Users can search and browse all other registered users.
- **Start New Conversations**: By selecting a user, a new chat panel opens, allowing the user to start a conversation directly.
- **User-Friendly Exploration**: The design is focused on making user exploration simple and efficient, so users can find new contacts without effort.

This section encourages social interaction, allowing users to connect with others they might not have chatted with before.

- **"Group Chat"**

The group chat functionality enhances the social aspect of the platform:

- **Creating Groups**: Users can create groups by selecting at least two other users from a dropdown list of available users.
- **Naming the Group**: The group creator can provide a custom group name, making it easy to identify different group chats.
- **Visible in Messages Section**: Once created, the group appears in the messages section, and any member of the group can participate in the chat.

This feature allows users to communicate efficiently in groups, making it perfect for discussions or collaborative chat sessions.

- **NavBar Navigation**

The side bar is a feature of the chat page:

- **Easy Navigation**: Users can switch between the messages, users, and group chat sections with ease.
- **User-Friendly Design**: The sidebar is intuitive and designed to reduce the learning curve for users, allowing them to navigate the platform effortlessly.

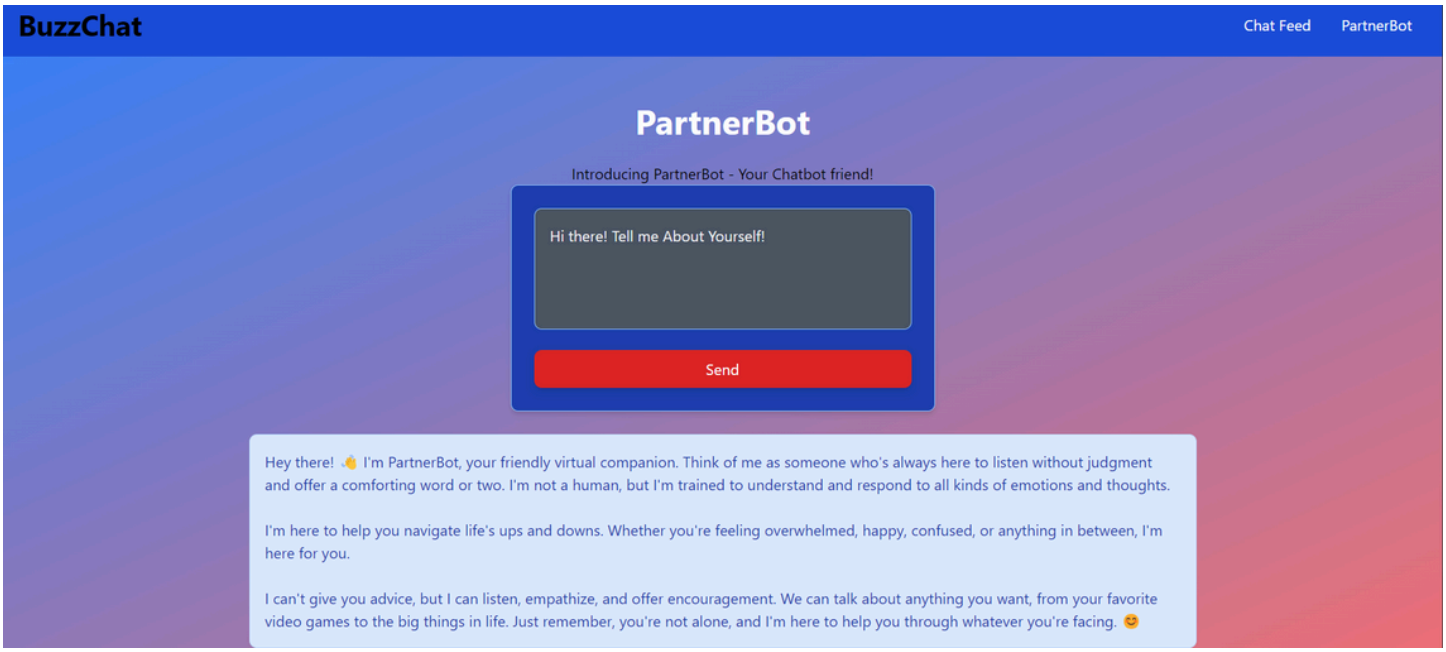
This sidebar enhances the overall user experience by providing quick access to all sections of the chat application.

- **"/chatbot" - PartnerBot**

The **PartnerBot** is an emotionally intelligent chatbot designed to provide emotional support:

- **Powered by Gemini API**: The chatbot uses advanced language processing from the Gemini API to offer users emotional counseling.
- **Motivational and Supportive**: PartnerBot is designed to uplift users during times of emotional distress, offering positive reinforcement and motivational conversations.
- **Accessible from the Chat Page**: Users can start interacting with the chatbot by navigating to the chatbot section in the side bar.

The chatbot provides a unique and empathetic user experience, making BuzzChat more than just a messaging service by offering emotional support when needed.



Backend Design

API ENDPOINTS

The backend consists of several key API endpoints that facilitate user registration, messaging, and conversation management. The endpoints are built using Next.js, integrating with a Prisma ORM for database interactions, and utilizing Pusher for real-time communication.

2. Endpoints

a. /api/register

- Purpose: To register new users and store their information in MongoDB.
- Method: POST
- Request Body:
  - email: User's email (required).
  - name: User's name (required).
  - password: User's password (required).
- Process:
  1. Validate the request body to ensure all required fields are present.
  2. Hash the password using bcrypt for secure storage.
  3. Create a new user record in the database using Prisma.
- Response: Returns the created user object or an error message if registration fails.

b. /api/messages

- Purpose: To send messages in a conversation.
- Method: POST
- Request Body:
  - message: The content of the message.

    - image: Optional image associated with the message.
    - conversationId: ID of the conversation where the message will be sent.
- Process:
    1. Validate the current user's authentication status.
    2. Create a new message in the database and associate it with the specified conversation and sender.
    3. Update the last message timestamp for the conversation.
    4. Use Pusher to trigger events for real-time updates to all users involved in the conversation.
- Response: Returns the newly created message object or an error message if the operation fails.

c. /api/conversations

- Purpose: To create or retrieve conversations.
- Method: POST
- Request Body:
    - userId: ID of the user to chat with.
    - isGroup: Boolean indicating if the conversation is a group chat.
    - members: List of user IDs for group chats.
    - name: Name of the group (if applicable).
- Process:
    1. Validate the current user's authentication.
    2. Check if a group conversation is being created and validate the data.
    3. For group chats, create a new conversation and connect all members.
    4. For one-on-one chats, check if an existing conversation already exists. If yes, return that; otherwise, create a new conversation.
- Response: Returns the created or found conversation object or an error message if the operation fails.

d. /api/conversations/{conversationid}

- Purpose: To delete a specific conversation by ID.
- Method: DELETE
- Request Parameters:
    - conversationId: The ID of the conversation to be deleted.
- Process:
    1. Validate the current user's authentication.
    2. Check if the conversation exists and if the user is part of it.
    3. If valid, delete the conversation and notify all users involved using Pusher.
- Response: Returns confirmation of deletion or an error message if the operation fails.

e. /api/chatbot

- Purpose: Interacts with the Gemini API for empathetic responses.
- Method: POST
- Request Body:
    - prompt: User's message (required).

- Process:
  1. Validate the prompt.
  2. Retrieve the Gemini API key.
  3. Define context for "PartnerBot."
  4. Send a request to the Gemini API.
  5. Return the response or an error.

Database Design Overview

1. **User**
   - **Fields**:
     - id: Unique identifier (ObjectId)
     - name: User's name
     - email: Unique email
     - emailVerified: Verification timestamp
     - image: Profile image URL
     - hashedPassword: User password hash
     - createdAt: Account creation timestamp
     - updatedAt: Last update timestamp

1. **Account**
   - **Fields**:
     - id: Unique identifier (ObjectId)
     - userId: Reference to User
     - type: Account type (e.g., OAuth)
     - provider: Service provider (e.g., Google)
     - providerAccountId: Unique account ID from provider
     - refresh_token, access_token: Tokens for authentication

1. **Conversation**
   - **Fields**:
     - id: Unique identifier (ObjectId)
     - createdAt: Timestamp for creation
     - lastMessageAt: Timestamp for the last message
     - name: Optional conversation name
     - isGroup: Boolean indicating if it’s a group conversation

1. **Message**
   - **Fields**:
     - id: Unique identifier (ObjectId)
     - body: Message content
     - image: Optional image attachment

- createdAt: Timestamp for message creation

Setting up Locally

Step 1: Clone the Repository Locally

Step 2: Setup MongoDB, Pusher and Google OAuth

Step 3: Setup .env.local File with add these variables with their Values

DATABASE_URL, NEXTAUTH_SECRET, NEXTAUTH_URL, NEXT_PUBLIC_PUSHER_APP_KEY, NEXT_PUBLIC_PUSHER_CLUSTER. PUSHER_APP_ID, PUSHER_SECRET, GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, GEMINI_API_KEY

Step 4: Run the command “yarn Install”

Step 5: Run the Command “npm run dev” to run it

Step 6: Open https://localhost:3020 to use the website.

Conclusion

BuzzChat is a versatile and user-friendly messaging platform designed to provide seamless one-on-one messaging, group chats, and emotional support through an AI-powered chatbot, PartnerBot. The project showcases modern web development practices using Next.js, JavaScript , and MongoDB for real-time communication. By incorporating OAuth authentication and an intuitive user interface, BuzzChat aims to create a secure and engaging space for users to connect and communicate.

With its extensible backend and API integration, the project is well-suited for future enhancements, including expanding chatbot functionalities or integrating additional messaging features. BuzzChat represents a practical, real-world solution to modern communication needs, blending social interaction with emotional support in an innovative way.