**CEREBRIO**

**(Chat Application)**

**A PROJECT REPORT**

**for**

**Project (KCA451)**

**Session (2024-25)**

**Submitted by**

**Prashant Srivastava**

**( 2300290140123 )**

**Satvik Srivastava**

**( 2300290140161 )**

**Ruchi Patel**

**( 2300290140149 )**

**Submitted in partial fulfilment of the**

**Requirements for the Degree of**

**MASTER OF COMPUTER APPLICATION**

**Under the Supervision of**

**Ms. Neha Tyagi**

**( Teaching Assistant )**

****

**Submitted to**

**Department Of Computer Applications**

**KIET Group of Institutions, Ghaziabad**

**Uttar Pradesh-201206**

**( June 2025)**

**CERTIFICATE**

Certified that **Prashant Srivastava ( 2300290140123 ), Satvik Srivastava ( 2300290140161), Ruchi Patel ( 2300290140149 )** has/ have carried out the project work having “**CEREBRIO**” (**Project-KCA451**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU**)** (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

**Ms. Neha Tyagi Dr. Akash Rajak**

**Teaching Assistant Dean**

**Department of Computer Applications Department of Computer Applications**

**KIET Group of Institutions, Ghaziabad KIET Group of Institutions, Ghaziabad**

**CEREBRIO**

Prashant Srivastava

Satvik Srivastava

Ruchi Patel

**ABSTRACT**

In today's digital age, communication plays a crucial role in connectin gindividuals across the globe. With the increasing demand for real-time messaging applications, building a single message chat application has becomea popular endeavor. This abstract introduces a chat application developed using the MERN (MongoDB, Express.js, React.js, and Node.js) technology stack , highlighting its key features and functionality.The proposed chat application leverages the MERN technology stack to providea robust and scalable solution. MongoDB, a NoSQL database, ensures efficientdata storage and retrieval, enabling seamless storage of user profiles andmessage history. Express.js, a web application framework for Node.js , facilitates the creation of a server-side backend that handles authentication , routing, and interaction with the database. Node.js acts as the run time environment, executing JavaScript code on the server-side, enabling high-performance communication. React.js, a JavaScript library, serves as thefrontend framework, providing an interactive and responsive user interface.The chat application encompasses essential features such as user registrationand authentication, login & logout, and ability to send and receive messages in real time.

**ACKNOWLEDGEMENT**

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Ms. Neha Tyagi** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Akash Rajak**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

**Prashant Srivastava**

**Satvik Srivastava**

**Ruchi Patel**

**TABLE OF CONTENTS**

**Content** **Page Number**

Certificate ii

Abstract iii

Acknowledgements iv

Table of Contents v-vi

List of Figures vii

1. **Introduction 1-4**

1.1 Introduction 1

* + 1. Background 1
    2. Project Overview 1
    3. Objective 2
    4. Key Features 3
    5. Scope of the project 4

1. **Problem Identification & Feasibility Study 5-8**
   1. Problem identification 5
   2. Feasibility Study 6
      1. Technical Feasibility 6
      2. Operational Feasibility 7
      3. Economic Feasibility 8
2. **Requirement analysis 9-12**
   1. Functional Requirements 9
      1. User Module 9
      2. Administrator Module 10
   2. Non-Functional Requirements 10
      1. Perforamance 11
      2. Security 11
      3. Scalability 11
      4. Usability 11-12
3. **Project Planning And Scheduling 13-15**
   1. Project Evaluation and Review Technique 13-15
4. **Hardware and Software Specification 16-17**
   1. Hardware Specification 16
   2. Software Specification 17
5. **ER-Diagram 18-21**
   1. Entity-Relationship Model 18
   2. Use Case Diagram 19
   3. Data Flow Diagram 20
      1. Context Level Diagram 20
      2. First Level Data Flow Diagram 21
6. **Design 22-26**
   1. Login 22-23
   2. Register 24
   3. Chat interface 25
   4. Chatbot interface 26
7. **Coding 27-53**
   1. Frontend Code 27-40
   2. Backend Code 41-53
8. **Testing 54-57**

9.1 Introduction 54

9.2 Type of Testing 54

9.2.1 Black box Testing 54

9.2.2 White box Testing 55

9.2.3 Unit Testing 56

9.2.4 Integration Testing 57

**Conclusion 58**

**References 59**

**LIST OF FIGURES**

**Figure No. Name of Figure Page No.**

4.1 Project Evaluation and Review Technique 15

6.1 Entity Relationship Diagram 18

6.2 Use Case Diagram 19

6.3 Context Level Data Flow Diagram 20

6.4 First Level Data Flow Diagram 21

7.1 Login Page 22

7.2 Login Page 23

7.3 Register Page 24

7.4 Chat Interface 25

7.5 Chatbot Interface 26

**CHAPTER 1**

**INTRODUCTION**

* 1. **Introduction**
     1. **Background**

Cerebrio is developed to address the growing need for efficient, real-time communication and AI-powered assistance in various domains. With the increasing demand for reliable and intuitive platforms, traditional messaging applications often lack the capabilities to provide intelligent responses tailored to user needs. Inspired by the United Nations’ Sustainable Development Goal (SDG) for Innovation and Infrastructure (SDG 9), Cerebrio combines the power of modern technologies like **React**, **Node.js**, and **AI models** to deliver an advanced messaging solution. This project aims to revolutionize user interaction by integrating chatbot intelligence with real-time communication features.

* + 1. **Project Overview**
* Cerebrio is a real-time messaging application integrated with an AI-poweredchatbot that adapts to user queries and assists with tasks.
* The platform includes core modules like user registration, authentication, and a chat interface.
* Users can interact with both human contacts and the AI chatbot, making the application suitable for personal, professional, and educational purposes.
* The application employs React.js for a dynamic and responsive frontend, while Node.js and MongoDB ensure secure and efficient data management on the backend.
* Cerebrio offers features such as instant notifications, a seamless messaging experience, and AI-driven solutions for user-specific tasks.
* The application aims to provide seamless connectivity for individuals and groups, incorporating advanced features for enhanced user experience
  + 1. **Objective**
* The primary objective of Cerebrio is to provide a robust and user-centric platform that enhances communication and problem-solving with AI assistance. Specific goals include:
* Facilitating seamless real-time messaging between users.
* Incorporating an AI chatbot capable of understanding user needs and generating relevant, contextual responses.
* Ensuring data security and privacy, adhering to modern standards for encryption and user authentication.
* Promoting innovation and accessibility, in alignment with SDG 9, by building a scalable and inclusive application.
* Cerebrio aims to improve productivity and simplify tasks through its dual capabilities of communication and AI support.
* To design a user-friendly interface that supports seamless communication across web and mobile platforms.
* To integrate the chatbot into various sectors such as customer service, education, healthcare, or enterprise support systems.
* To implement machine learning models that enable the chatbot to learn from interactions and improve over time.
* To build an AI-powered chatbot capable of understanding and processing natural language inputs from users.
* To ensure data privacy, security, and compliance with relevant regulations.
* To provide instant, automated responses for common queries, reducing the need for human intervention.
  + 1. **Key Features**

Cerebrio stands out with its advanced features tailored to user needs:

* **Real-Time Messaging:** Instant communication with delivery status updates and notifications.
* **AI-Powered Chatbot:** A smart assistant capable of generating responses, solving user queries, and performing tasks.
* **User Authentication:** Secure login and registration with role-based access control.
* **Chat Management:** Organized conversations with options to pin, delete, or search chats.
* **Responsive Design:** Fully optimized for desktops, tablets, and mobile devices.
* **Scalable Infrastructure:** Supports high user traffic while maintaining performance.
* **Customization :** Options to personalize the chatbot’s behavior or design themes to suit user preferences.
* **Multi-Platform Support :** Compatible with web , mobile , and desktop platforms, enabling users to access the chatbot anytime, anywhere.
* **User-Friendly Interface :** Designed with an intuitive and interactive UI to ensure ease of use and smooth navigation for users of all levels .
* **Analytics dashboard :** Offers detailed insights and reports on users interactions, chatbot performance, and frequently asked questions to support continuous improvement.
* **Security:** End-to-end encryption for messages and calls.

These features make Cerebrio an all-in-one solution for communication and intelligent task management.

* + 1. **Scope of the project**

The scope of Cerebrio defines its functionality, audience, and development boundaries:

**Inclusions:**

* Real-time messaging capabilities for users.
* AI chatbot integration for task handling and query resolution.
* User and admin modules to manage roles, data, and system functionality.
* Backend features like secure APIs, database management, and robust server communication.

**Exclusions:**

* Currently, Cerebrio does not include integrations with third-party applications like payment gateways or advanced machine learning features such as predictive analytics.
* Offline messaging functionality is outside the current version's scope but could be added in future enhancements.

**Target Audience:**

Cerebrio caters to businesses, educational institutions, and individuals seeking a blend of communication and AI-driven solutions.

**Future Possibilities:**

The project lays the groundwork for further development, such as integrating voice-to-text capabilities, multilingual support, and real-time analytics to enhance user experience and broaden its usability.

**CHAPTER 2**

**PROBLEM IDENTIFICATION & FEASIBILITY STUDY**

**2.1 Problem Identification**

In the modern era of digital communication, users increasingly rely on messaging platforms for both personal and professional purposes. However, most platforms lack features that combine **real-time communication** with **intelligent AI assistance**. This creates the following gaps:

* **Lack of Intelligent Assistance:** Traditional messaging applications do not offer real-time AI-driven solutions, leaving users to switch between multiple platforms to get answers or perform tasks.
* **Time-Consuming Interactions:** Users often spend excessive time searching for information or manually solving problems that could be addressed by an AI chatbot.
* **Limited Customization:** Existing platforms provide minimal personalization options, limiting user satisfaction and adaptability to specific needs.
* **Operational Challenges in Businesses:** Organizations struggle with efficient customer query handling and communication tools that lack automation capabilities.
* **High operational Costs :** Maintaining a large team of human support agents increases long-term costs.

**Real-world Scenario:**

Imagine a customer reaching out to a support team via chat for a product query. Traditional messaging apps only allow communication with a human operator, leading to delays during non-operational hours. An AI-powered solution like Cerebrio can instantly provide relevant responses and escalate complex queries to human agents.

Cerebrio addresses these issues by integrating a real-time messaging platform with an AI chatbot to enhance communication and productivity.

**2.2 Feasibility Study**

A Feasibility Study is an assessment conducted to evaluate the practicality and viability of a proposed project or system. It analyzes various factors to determine whether the project is technically, economically, legally, and operationally feasible and if it aligns with the organization's goals.

A feasibility study helps determine whether the Cerebrio AI-based chat application project is viable across various aspects. The key feasibility areas include:

**2.2.1 Technical Feasibility**

Cerebrio leverages modern, reliable technologies and tools to ensure smooth functionality:

* React.js for the frontend: Offers a responsive and dynamic user interface.
* Node.js and Express.js for the backend: Ensures fast and scalable server-side operations.
* MongoDB for the database: Provides a flexible and efficient data storage solution.
* AI-powered chatbot: Uses machine learning models or APIs like OpenAI for generating intelligent responses.
* Skill Availability: The required development skills in JavaScript, AI integration, and database management are readily accessible.
* The project uses proven technologies such as Natural Language Processing (NLP), Machine Learning (ML), and cloud-based platforms.
* Skilled developers and AI experts are accessible to implement the solution.

The project is technically feasible as the tools and frameworks are mature, widely supported, and capable of meeting the project’s requirements.

**2.2.2 Operational Feasibility**

Cerebrio aligns well with operational needs:

* **Ease of Use:** The application is designed with an intuitive interface, ensuring a smooth learning curve for users.
* **AI Chatbot Integration:** Enhances efficiency by reducing the workload on human operators and providing 24/7 availability.
* **Scalability:** Can handle a growing number of users, making it suitable for individuals, businesses, and institutions.
* **Cross-Domain Applications:** Supports use cases across industries such as customer support, education, and project collaboration.
* **Supports Multi-Language Communication:** Can be extended to support multiple languages to cater to a wider audience.
* **Feedback and Continuous Improvement:** The system collects user feedback and learns from interactions to continuously improve performance.
* **Efficient Handling of High Volume Queries:** Capable of managing multiple user queries simultaneously without performance degradation.
* **Improved Customer Experience:** Instant responses and intelligent conversation handling enhance overall customer engagement and loyalty.
* **Easy Integration with Existing Systems:** The chat application can be seamlessly integrated into websites, mobile apps, and customer service platforms.
* **User-Friendly Interface:** Designed with a simple and intuitive UI, making it easy for both users and administrators to interact with the system.
* **Adaptability to Different Domains:** Can be customized for various industries such as healthcare, education, finance, and e-commerce.

**2.2.3 Economic Feasibility**

The development and maintenance of Cerebrio are cost-effective due to the use of open-source technologies:

* **Low Development Costs:** Tools like React.js, Node.js, and MongoDB reduce software expenses.
* **Efficient Resource Usage:** The application is hosted on scalable platforms like cloud services, which optimize costs for growing user bases.
* **Revenue Potential:** Cerebrio can generate revenue through subscription models, premium AI features, and business-oriented customizations.
* **Supports Business Growth:** Automating customer support enables businesses to grow without proportionally increasing support staff.
* **Faster Issue Resolution:** Reduces downtime and support ticket handling costs due to real-time problem-solving capabilities.
* **Use of Cost-Effective Technologies**: Cloud platforms, open-source libraries (e.g., TensorFlow, Rasa), and NLP APIs reduce software licensing and hosting costs.
* **Scalability Saves Future Costs:** The application is easily scalable to handle increasing users without proportionate increase in costs.
* **Minimal Maintenance Cost:** Once deployed, the application requires minimal upkeep, with periodic updates and model training .
* **Moderate Initial Investment:** The development cost is reasonable and manageable, especially with the use of open-source tools and frameworks.

By balancing development costs with potential earnings, Cerebrio proves to be a financially viable project.

**CHAPTER 3**

**REQUIREMENT ANALYSIS**

Requirement analysis is a crucial phase in the development of Cerebrio, a real-time messaging platform integrated with an AI-powered chatbot. The purpose of this phase is to identify and define the essential features and functionalities required for the successful implementation of the system. This section will outline both functional and non-functional requirements, ensuring that Cerebrio can deliver on its promise of efficient communication, secure operations, and continuous improvement based on real-time data.

**3.1 Functional Requirements**

**3.1.1 User Module (Implemented)**

The User Module is designed to provide end-users with an interactive, intuitive, and seamless communication experience. It includes:

* **Real Time Messaging :** Enables users to send and receive messages without noticeable delays, ensuring smooth and seamless conversations.
* **Chat History management :**

Users can engage in one-on-one or group chats with immediate message delivery. This is powered by WebSocket technology, ensuring no delays in communication.

* **File Sharing:**

Users can upload and download files securely, including images, documents, and other formats. Files are stored on a secure server and can be accessed or shared as needed.

* + **AI Chatbot Support :**

The AI chatbot offers users instant assistance for common queries, automates tasks like reminders or scheduling, and learns from user interactions to improve its responses over time.

* + **Customizable User Profile:**

Users can customize their profiles by uploading a profile picture, changing their status, and editing personal details, enhancing their interaction within the platform.

**3.1.2 Administrator Module (Implemented)**

The Administrator Module empowers platform administrators to manage users, monitor activity, and ensure the platform operates smoothly. It includes:

* **User Management :**

Admins can activate/deactivate user accounts, manage roles , and set user-specific preferences.

* **Content Moderation:**

AI-based filtering ensures that inappropriate content is flagged, allowing admins to delete messages or block users based on platform policies.

* **System Analytics Dashboard :**

A real-time dashboard provides insights into system performance, user activity, and server health, allowing admins to make informed decisions about platform maintenance and improvements.

* **Data Backup and Restore:**

The system backs up all data regularly and enables recovery in case of system failure, ensuring no data is lost during unforeseen events.

* **Security Monitoring:**

Admins can monitor security events, such as suspicious logins or activity, and take immediate action to prevent potential threats.

**3.2 Non-Functional Requirements**

**3.2.1 Performance:**

Performance is crucial for the success of Cerebrio, ensuring that the platform operates efficiently and handles large volumes of users and messages. Key performance requirements include:

* **Low Latency:**

Messages and responses (including those from the AI chatbot) should be delivered in real-time, with minimal delay.

* **Scalability:**

The platform must be able to handle increasing numbers of concurrent users without degrading the performance of chat functionalities or server responses.

* **High Availability :**

The system should maintain high uptime, ensuring that users can access and interact with the platform without interruptions.

**3.2.2 Security**

Security is a top priority for ensuring the integrity and confidentiality of user data and platform interactions. Key security requirements include:

* **Data Encryption:**

All messages and user data must be encrypted both in transit and at rest to prevent unauthorized access.

* **User Authentication and Authorization:**

Secure authentication methods, such as multi-factor authentication, must be implemented to ensure that only authorized users access the platform. Role-based access control will also be used to define the permissions of each user type

* **Content Moderation and Filtering:**

AI-powered content moderation tools must ensure that inappropriate messages, images, or files are detected and filtered before they reach the platform's users.

**3.2.3 Scalability**

As Cerebrio grows, it must be scalable to accommodate more users and new features. The platform needs to support the following scalability requirements:

* **Horizontal scalling:**

The platform should be able to scale horizontally by adding more servers to handle increased load without performance degradation.

* **Database Scalability:**

The database must support the rapid increase in user data and chat history, with mechanisms in place for efficient querying and storage.

* **Cloud-Based Infrastructure:**

Using cloud services (e.g., AWS, Google Cloud) will ensure that the platform can scale up or down depending on the demand, providing flexibility in resource management.

**3.2.4 Usability**

Usability ensures that the Cerebrio platform is user-friendly, intuitive, and efficient for all users, regardless of their technical expertise. Usability requirements include:

* **Intuitive User Interface :**

The platform must have a clean, easy-to-navigate interface, with features easily accessible and understandable by both regular users and administrators.

* **Responsive Design :**

The platform should function seamlessly across different devices (desktop, tablet, mobile) and adapt to varying screen sizes without compromising the user experience.

* **Accessibility :**

The platform should be accessible to users with disabilities, including support for screen readers and keyboard navigation for visually impaired users.

* **Feedback Machanism :**

Users should have the ability to provide feedback on their experience, helping administrators identify areas for improvement.

**CHAPTER 4**

**PROJECT PLANNING AND SCHEDULING**

**4.1 Project Evaluation and Review Technique**

The **Project Evaluation and Review Technique (PERT)** chart is a project management tool used to plan, schedule, and track the tasks and milestones involved in a project. It visually represents the sequence of activities, the time required for each activity, and their interdependencies. In the case of **Cerebrio**, the PERT chart will help organize the development and deployment phases of the real-time messaging platform and its AI-powered chatbot.

Below is an outline of the PERT chart for the Cerebrio project, showing the key milestones and tasks involved in its development process.

**Steps and Milestones:**

1. **Project Initiation:**

* Define project scope and objectives.
* Identify the stakeholders (users, admins, developers, etc.).
* Set up initial project resources and team.

1. **System Design:**

* Finalize the architecture of the platform (frontend, backend, database).
* Design user interface (UI) and experience (UX) for the platform.
* Select technology stack (e.g., Node.js, React, MongoDB, etc.).
* Design chatbot workflow and features.

1. **Frontend Development:**

* Develop user interface for real-time messaging.
* Implement chat history and search features.
* Design user profile management and file sharing features.

1. **Backend Development:**

* Implement real-time messaging via WebSocket.
* Set up user management and authentication.
* Integrate the AI chatbot and train it on initial datasets.
* Develop the database schema and integrate data storage.

1. **Admin Module Development:**

* Implement user management system.
* Create content moderation tools and filters.
* Set up system analytics and reporting dashboard.
* Implement data backup and restore features.

1. **Testing Phase:**

Conduct unit testing for individual components.

* + Perform integration testing to ensure smooth communication between frontend, backend, and chatbot.
  + Perform load testing to ensure scalability and performance under high user traffic.

1. **Deployment:**
   * Set up cloud hosting (e.g., AWS, Google Cloud).
   * Deploy the platform to production servers.
   * Ensure proper configuration of security features and data encryption.
2. **Post-Deployment Monitoring and Maintenance:**
   * Monitor server health, user activity, and chatbot performance.
   * Collect feedback from users to identify areas for improvement.
   * Release updates for bug fixes and new features.

A PERT chart for **Cerebrio** would display these milestones as nodes (tasks) connected by arrows representing the dependencies between tasks. Each task would be assigned an estimated duration, and critical paths would be identified to highlight the essential sequence of tasks that must be completed on time to avoid project delays.

For example, the development of the User Module (including messaging, file sharing, and AI chatbot integration) must be completed before testing and deployment can begin. Similarly, the Admin Module development is dependent on the completion of the Frontend Development and Backend Development phases.

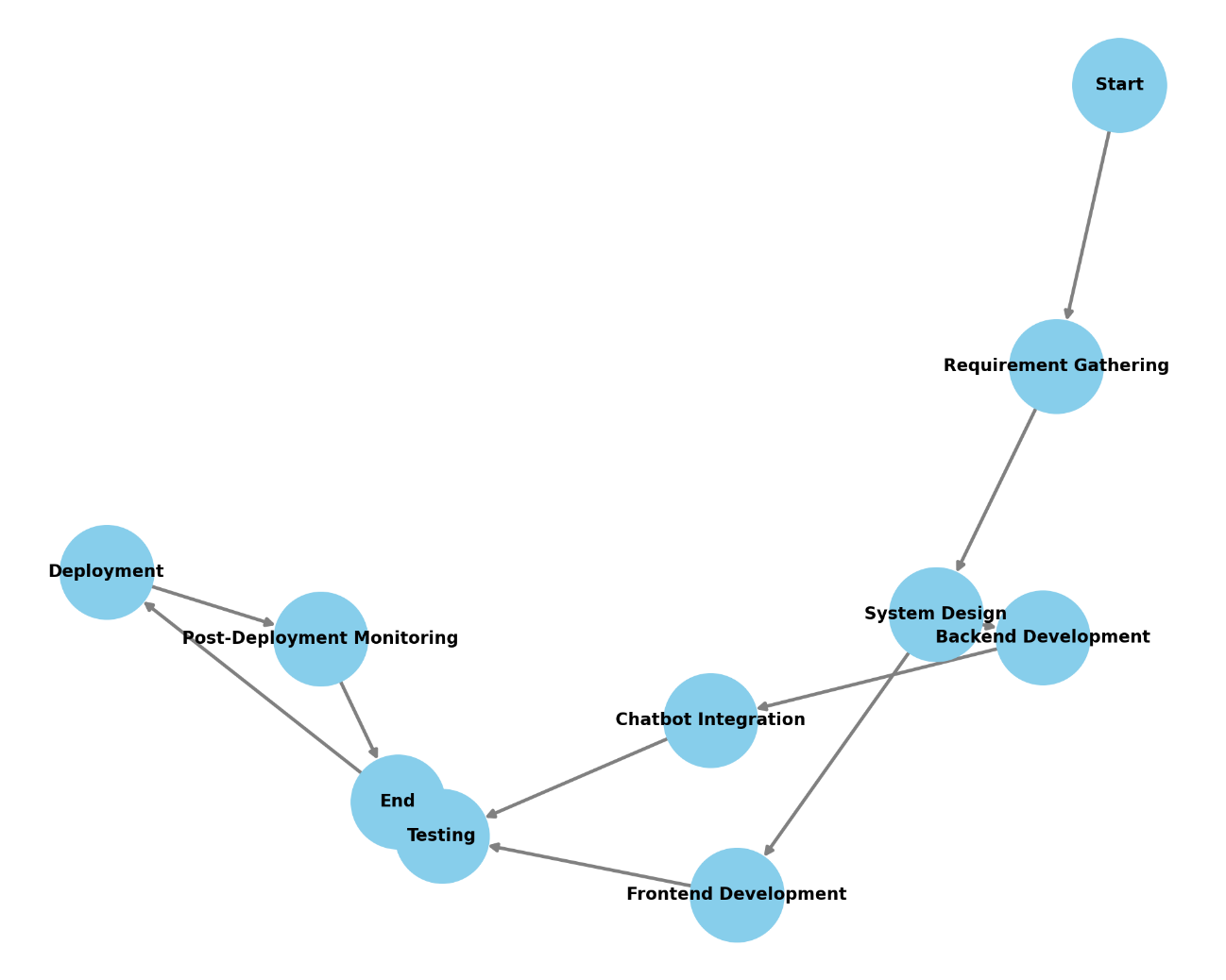


Fig 4.1 Project Evaluation and Review Technique

Project Evaluation and Review Technique (PERT) chart illustrating the critical path and timeline for key development phases of the Cerebrio project, ensuring efficient task scheduling and timely delivery.

**CHAPTER 5**

**HARDWARE AND SOFTWARE SPECIFICATION**

**5.1 Hardware Specification**

The hardware specifications for the Cerebrio project are tailored to ensure that the platform operates efficiently and handles the necessary real-time communication and AI processes. Below are the required hardware specifications:

**1. Server Requirements:**

* **Processor:** Intel Xeon E5 or AMD Ryzen 5 or better
* **RAM:** 16GB or more
* **Storage:** SSD with a minimum of 1TB storage
* **Network Interface:** 1Gbps Ethernet or higher
* **Backup:** Redundant storage and cloud backup options for high availability

**2. Client Requirements:**

* **Processor:** Intel i5 or higher for Windows/Linux/macOS systems
* **RAM:** 8GB minimum
* **Storage:** 100GB free disk space
* **Display:** Minimum 1920x1080 resolution
* **Internet:** Stable high-speed internet (minimum 10Mbps)
* **Operating System:** Windows 10/11, macOS, or Linux

**3. Other Requirements:**

* **Database Servers:** High-performance database servers (e.g., MongoDB) with adequate CPU, RAM, and SSD storage for fast data retrieval and backup.
* **AI Model Training Resources:** GPUs (e.g., Nvidia RTX 3090) for training complex AI models efficiently.

**5.2 Software Specification**

The software specification outlines the technologies and tools used to develop and maintain Cerebrio, ensuring a robust and scalable application. The key software specifications are:

**1. Frontend:**

* + **React.js:** A JavaScript library for building user interfaces, ensuring fast rendering and a seamless experience.
  + **Redux:** For state management, especially for handling the global state of chat data and user preferences.
  + **HTML5, CSS3, JavaScript (ES6+):** To build responsive and interactive web components.
  + **Bootstrap:** For fast and responsive UI development, ensuring consistency across devices.

**2. Backend:**

* + **Node.js:** A JavaScript runtime used for handling asynchronous events and managing real-time messaging and chat functionalities.
  + **Express.js:** A web framework for building RESTful APIs and ensuring smooth communication between the frontend and backend.
  + **Socket.IO:** For real-time, bidirectional communication between users (e.g., live chat).
  + **MongoDB:** NoSQL database for storing user data, messages, and chat logs.
  + **Mongoose:** ODM (Object Data Modeling) library for MongoDB, providing a straightforward schema-based solution to model the app's data.

**3. AI Integration:**

* **TensorFlow or PyTorch:** For building and training machine learning models for the AI-powered chatbot.
* **NLP (Natural Language Processing) Libraries:** Libraries such as spaCy or NLTK to handle the understanding and generation of human language for the chatbot.
* **Dialogflow:** Google’s natural language understanding platform used for developing chatbots and conversational interfaces.

**4. DevOps and Deployment:**

* **Git:** For version control and collaborative development.
* **Docker:** For containerizing the application and deploying it consistently across different environments.
* **AWS/GCP:** Cloud platforms for hosting the application and managing the infrastructure.
* **NGINX:** Web server for load balancing and managing API requests.

**5. Testing:**

* **Jest:** JavaScript testing framework for unit testing frontend and backend components.

**CHAPTER 6**

**ER-DIAGRAM**

**6.1 Entity Relationship Model**

An Entity-Relationship (ER) Diagram is a graphical representation used in database design to illustrate the structure of a system and the relationships between entities within it.

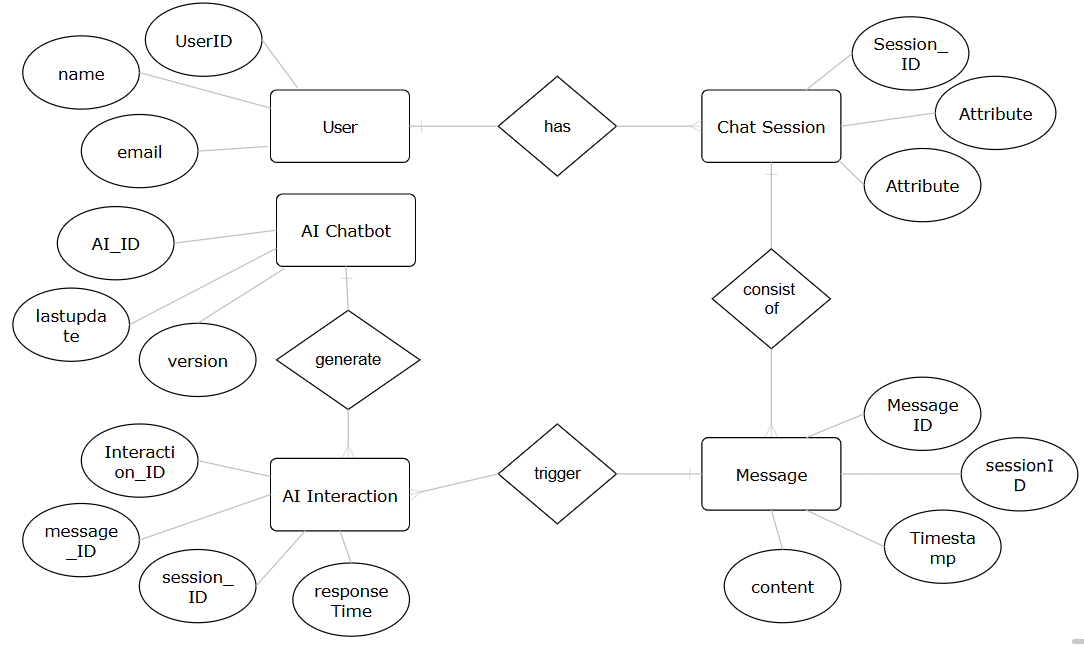


Fig 6.1 Entity Relationship Diagram

**6.2 Use Case Diagram**

A Use Case Diagram is a type of visual representation in the Unified Modeling Language (UML) that depicts the functional requirements of a system from the user's perspective. It helps identify the system's interactions with users and the specific tasks or processes that the system supports.

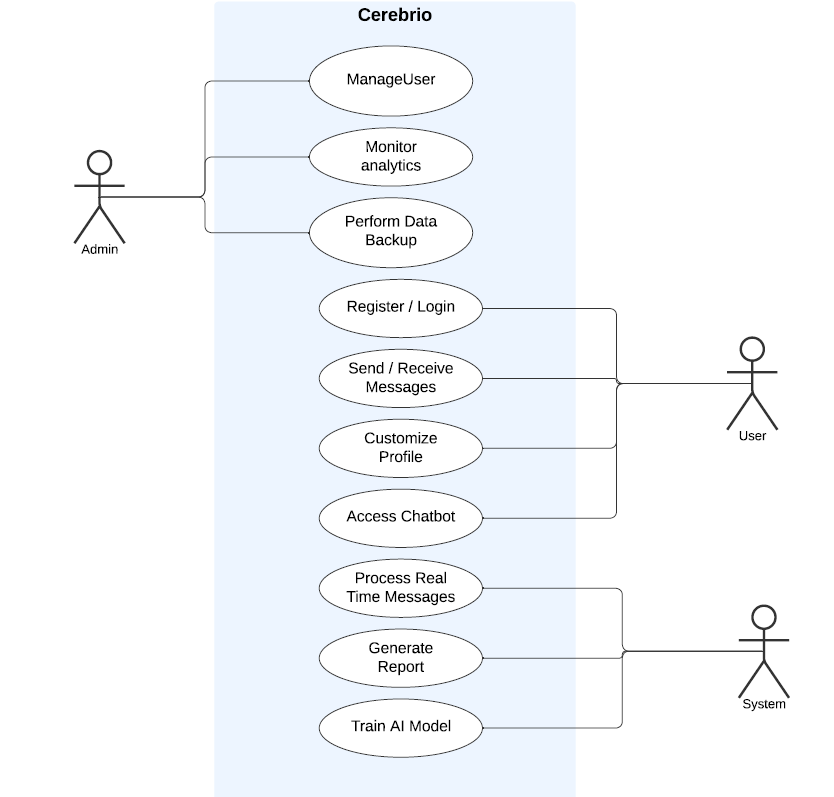


Fig 6.2 Use Case Diagram

**6.3 Data Flow Diagram**

**6.3.1 Context Level Diagram**

A Context Level Diagram (CLD), also known as a Level 0 Data Flow Diagram (DFD), is a high-level overview of a system. It represents the system as a single process and shows the flow of data between the system and external entities.

**Components of a Context Level Diagram**:

* **Process**: The central system represented as a single circle or bubble.
* **External Entities**: Represented as rectangles, these are sources or destinations of data interacting with the system (e.g., users, other systems).
* **Data Flows**: Represented as arrows, these indicate the flow of data between the system and external entities.

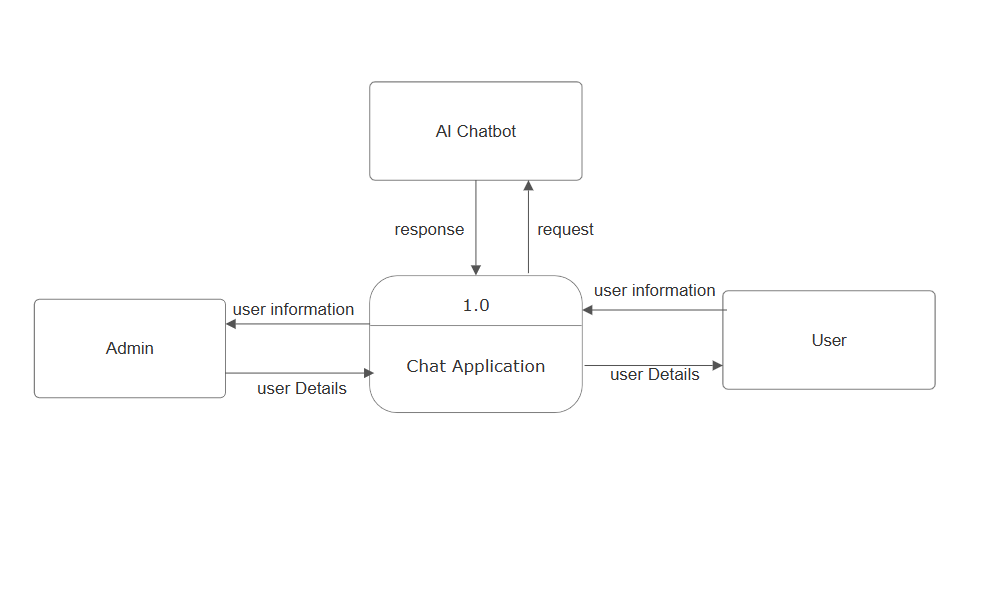


Fig 6.3 Context Level Data Flow Diagram

**6.3.2 First Level Data Flow Diagram**

A FirstLevel Data Flow Diagram (DFD**)**, also known as Level 1 DFD, breaks down the high-level process (from the Context Level Diagram) into sub-processes to provide a more detailed view of the system. It shows how the main system functions interact and exchange data internally while continuing to interact with external entities.

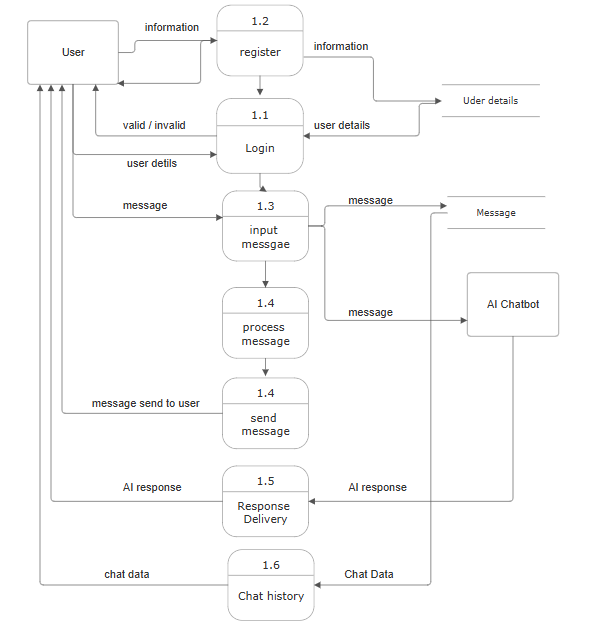


Fig 6.4 First Level Data Flow Diagram

**CHAPTER 7**

**DESIGN**

* 1. **Login Page**

The login page of the CEREBRIO application serves as a secure entry point for users to access its AI-powered messaging and chatbot features. The design is user-friendly, intuitive, and ensures a seamless authentication process.

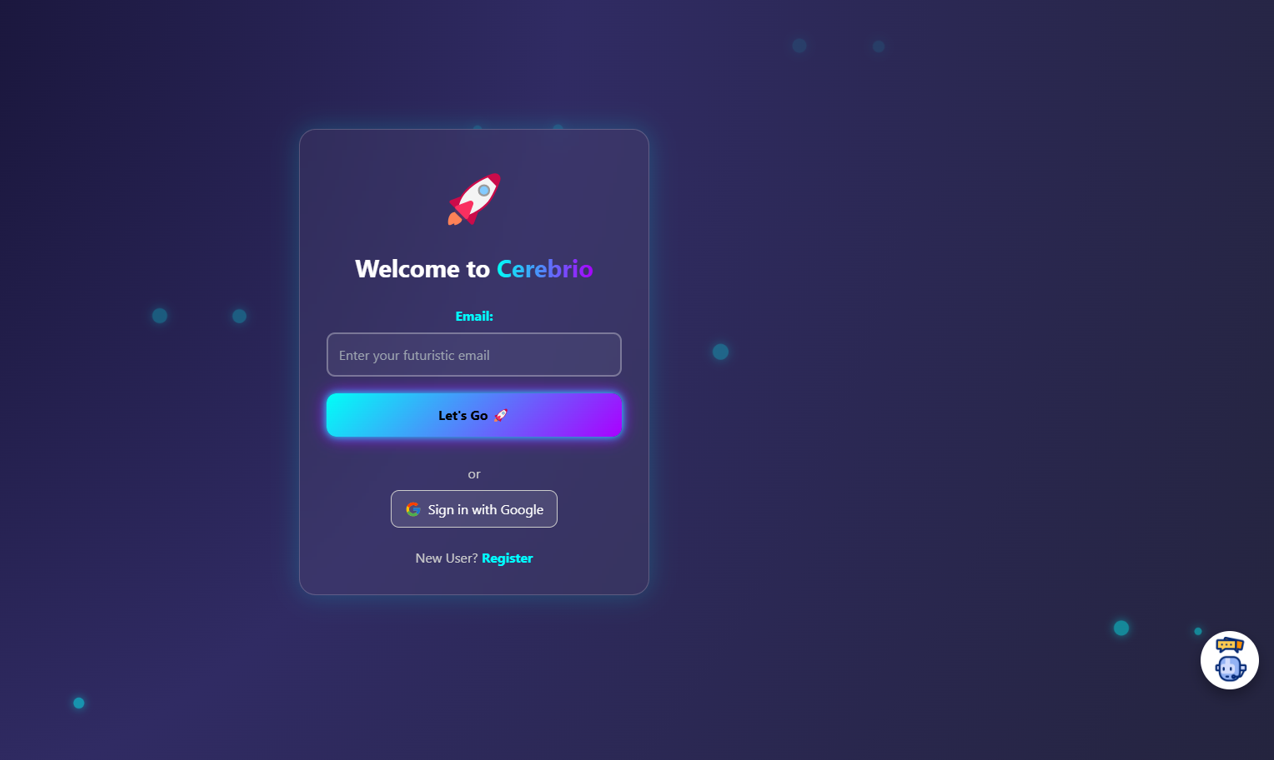
****

Fig 7.1 Login page

User Can Login Through Email and password

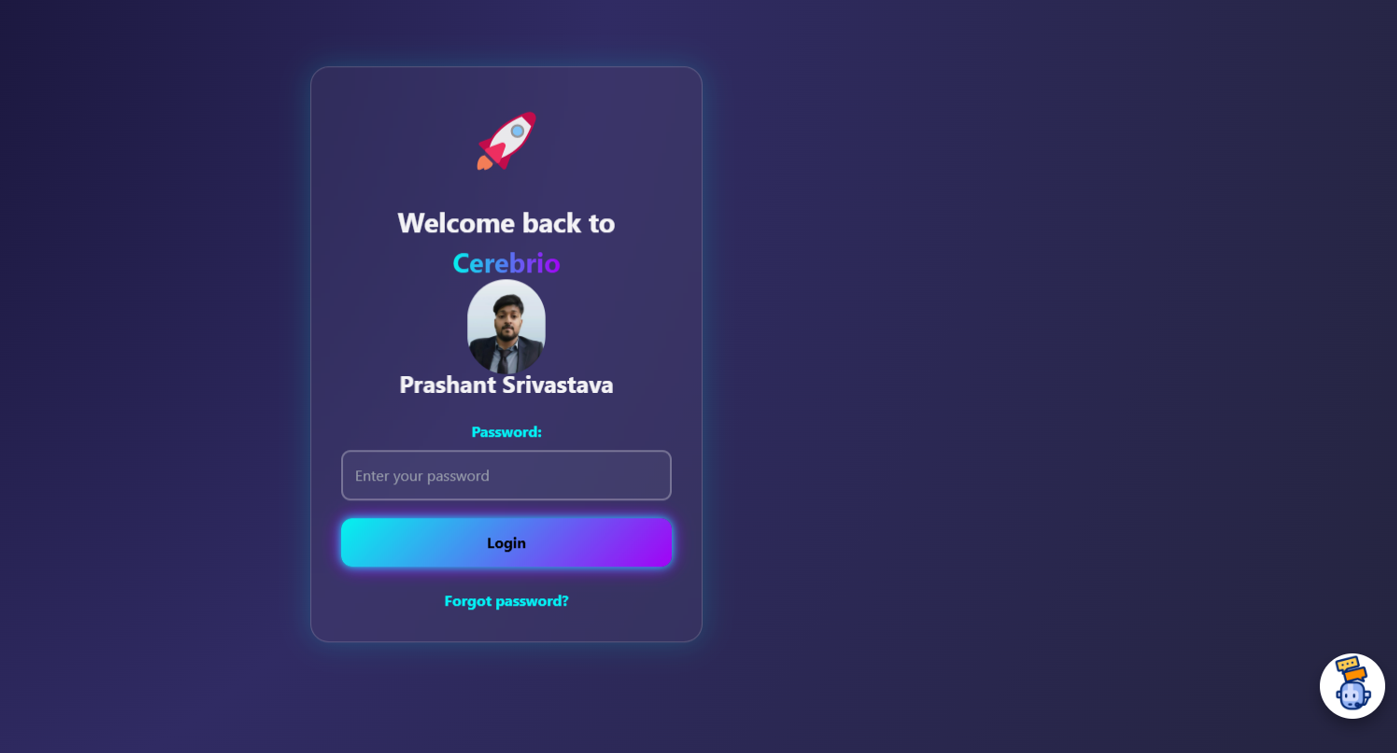


Fig 7.2 Login page

* 1. **Register Page**

The registration page of the CEREBRIO application is designed to enable new users to create an account seamlessly and securely. It ensures a user-friendly experience while maintaining data privacy and security standards. User Can register through Name , email, password and profile picture.

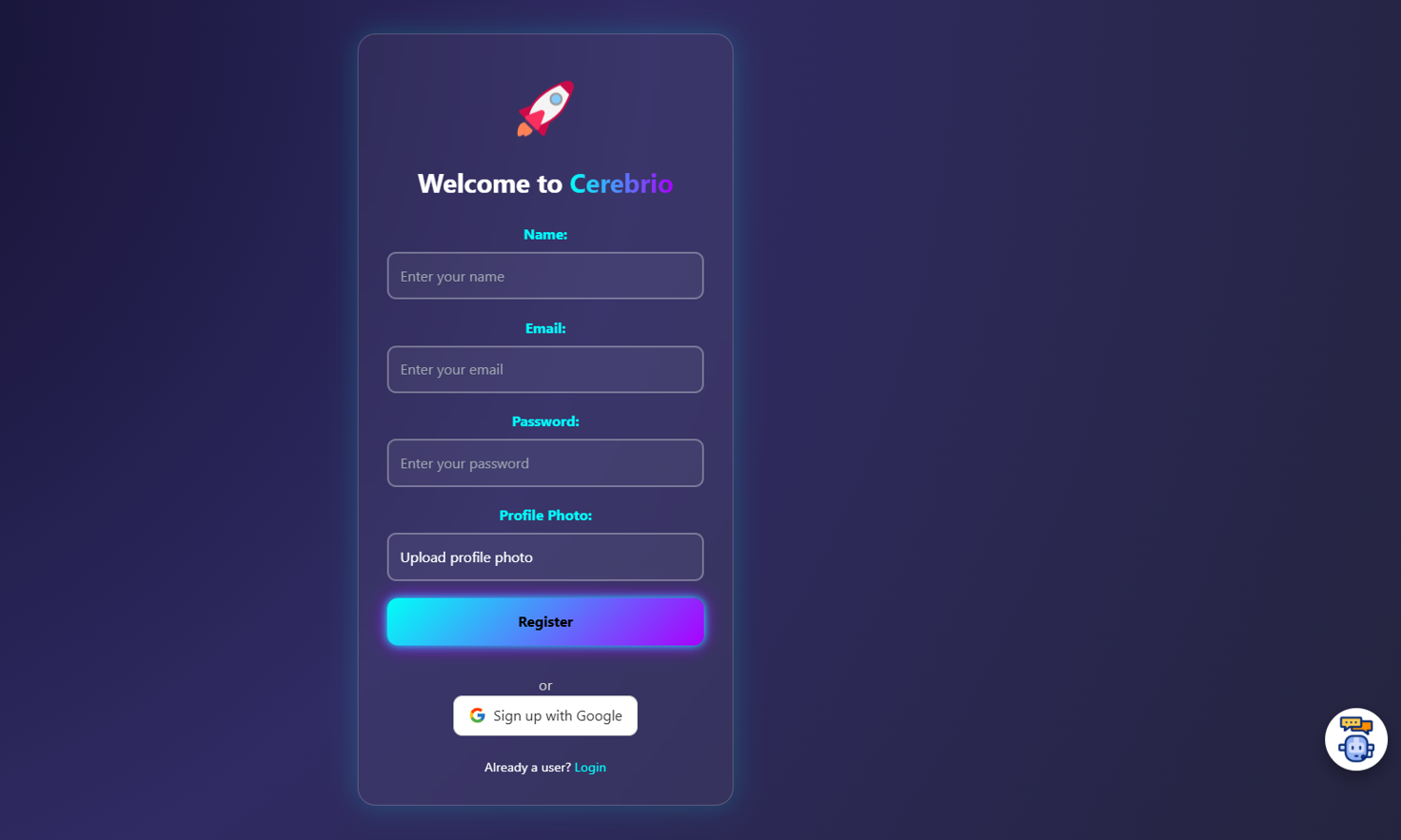


Fig 7.3 Register Page

* 1. **Chat interface**

The chat interface of the CEREBRIO application is designed to provide a seamless and intelligent communication experience. Powered by AI, it combines a user-friendly layout with advanced functionalities to ensure efficient and engaging interactions.



Fig 7.4 Chat Interface

**Chatbot interface**

The chatbot interface of the CEREBRIO application offers an intuitive and AI-driven environment for seamless interactions. Designed for efficiency and user engagement, it leverages advanced artificial intelligence to provide meaningful and personalized responses.



Fig 7.5 Chatbot Interface

**CHAPTER 8**

**CODING**

The development of **Cerebrio** was guided by modern software engineering practices, focusing on modularity, scalability, and maintainability. The platform utilizes a MERN stack (MongoDB, Express.js, React.js, Node.js) to handle both the frontend and backend, ensuring a smooth and responsive user experience. Additionally, WebSocket technology was integrated for real-time communication, and AI-based chatbotfunctionality was developed using machine learning libraries..

**8.1 Frontend Code**

The frontend of **Cerebrio** is built using **React.js**, allowing for a dynamic and responsive user interface. The user interface includes key features such as real-time messaging, user profile management, chatbot interaction, and file uploads.

**1.Register Page Code**

import React, { useState } from 'react';

import { IoClose } from "react-icons/io5";

import { Link, useNavigate } from 'react-router-dom';

import axios from 'axios';

import toast from 'react-hot-toast';

import './RegisterPage.css'; // Make sure to import the CSS

const RegisterPage = () => {

  const [data, setData] = useState({

    name: "",

    email: "",

    password: "",

profile\_pic: ""

  });

  const [uploadPhoto, setUploadPhoto] = useState(null);

  const navigate = useNavigate();

  const handleOnChange = (e) => {

    const { name, value } = e.target;

    setData((prev) => ({

      ...prev,

      [name]: value

    }));

  };

  const handleUploadPhoto = async (e) => {

    const file = e.target.files[0];

    if (file) {

      setUploadPhoto(file); // Display file name in UI

    }

  };

  const handleClearUploadPhoto = (e) => {

    e.stopPropagation();

    e.preventDefault();

    setUploadPhoto(null);

    setData((prev) => ({

      ...prev,

      profile\_pic: ""

    }));

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    const formData = new FormData();

    formData.append("name", data.name);

    formData.append("email", data.email);

    formData.append("password", data.password);

    if (uploadPhoto) {

      formData.append("profile\_pic", uploadPhoto);

    }

    const URL = `${process.env.REACT\_APP\_BACKEND\_URL}/api/register`;

    try {

      const response = await axios.post(URL, formData, {

        headers: {

          "Content-Type": "multipart/form-data"

        }

      });

      if (response.data.success) {

        toast.success(response.data.message);

        setData({

          name: "",

          email: "",

          password: "",

          profile\_pic: ""

        });

        setUploadPhoto(null);

        navigate('/email');

      } else {

        toast.error(response.data.message);

      }

    } catch (error) {

      toast.error(error.response?.data?.message || "Error registering user");

    }

  };

  return (

    <div className='register-container'>

      <div className='bg-card w-full max-w-md rounded overflow-hidden p-4 mx-auto'>

        <h3 className='text-center text-2xl mb-4'>Welcome to Chat App!</h3>

        <form className='grid gap-4 mt-5' onSubmit={handleSubmit}>

          <div className='flex flex-col gap-1'>

            <label htmlFor='name'>Name :</label>

            <input

              type='text'

              id='name'

              name='name'

              placeholder='Enter your name'

              className='input-field'

              value={data.name}

              onChange={handleOnChange}

              required/>

          </div>

          <div className='flex flex-col gap-1'>

            <label htmlFor='email'>Email :</label>

            <input

              type='email'

              id='email'

              name='email'

              placeholder='Enter your email'

              className='input-field'

              value={data.email}

              onChange={handleOnChange}

              required/>

          </div>

          <div className='flex flex-col gap-1'>

            <label htmlFor='password'>Password :</label>

            <input

              type='password'

              id='password'

              name='password'

              placeholder='Enter your password'

              className='input-field'

              value={data.password}

              onChange={handleOnChange}

              require/>

</div>

          <div className='flex flex-col gap-1'>

            <label htmlFor='profile\_pic'>Photo:</label>

            <div

              onClick={() => document.getElementById('profile\_pic').click()}

              className='h-14 bg-slate-200 flex justify-center items-center border rounded hover:border-primary cursor-pointer profile-upload'  >

              <p className='text-sm max-w-[300px] text-ellipsis line-clamp-1'>

                {uploadPhoto?.name || "Upload profile photo"}

              </p>

              {uploadPhoto && (

                <button className='text-lg ml-2 hover:text-red-600' onClick={handleClearUploadPhoto}>

                  <IoClose />

                </button>

              )}

            </div>

            <input

              type='file'

              id='profile\_pic'

              name='profile\_pic'

              className='hidden'

              onChange={handleUploadPhoto} />

          </div>

          <button className='btn-submit'>Register</button>

        </form>

        <p className='my-3 text-center'>

          Already have an account? <Link to={"/email"} className='link-login'>Login</Link>

        </p>

      </div>

    </div>

  );

};export default RegisterPage;

**8.2 Backend Code**

const express = require('express');

const { Server } = require('socket.io');

const http = require('http');

const getUserDetailsFromToken = require('../helpers/getUserDetailsFromToken');

const UserModel = require('../models/UserModel'); // User model

const { ConversationModel, MessageModel } = require('../models/ConversationModel');

require('dotenv').config();

// Create an Express application

const app = express();

app.use(express.json()); // Middleware to parse JSON requests

// Set up a Socket.IO server with CORS settings

const server = http.createServer(app);

const io = new Server(server, {

cors: {

origin: "http://localhost:3000",

credentials: true,

},

});

// Online user set to track connected users

const onlineUser = new Set();

io.on('connection', async (socket) => {

console.log("Connected User", socket.id);

const token = socket.handshake.auth.token;

console.log("token inside sockwet index.js"+token);

const user = await getUserDetailsFromToken(token);

if (!user || !user.\_id) {

console.log('Invalid user or missing user ID.');

socket.disconnect();

return;

}

socket.join(user.\_id.toString());

onlineUser.add(user.\_id.toString());

io.emit('onlineUser', Array.from(onlineUser));

socket.on('loadMessages', async (conversationId) => {

try {

const messages = await MessageModel.find({ conversationId })

.populate('msgByUserId', 'username') // Populate user details if needed

.exec();

socket.emit('messages', messages);

} catch (error) {

console.error('Error loading messages:', error);

socket.emit('error', { message: 'Could not load messages' });

}

});

socket.on('new message', async (data) => {

try {

let conversation = await ConversationModel.findOne({

"$or": [

{ sender: data.sender, receiver: data.receiver },

{ sender: data.receiver, receiver: data.sender },

]

});

if (!conversation) {

conversation = await ConversationModel.create({

sender: data.sender,

receiver: data.receiver,

});

}

const message = new MessageModel({

text: data.text,

imageUrl: data.imageUrl,

videoUrl: data.videoUrl,

msgByUserId: data.sender,

conversationId: conversation.\_id,

});

const savedMessage = await message.save();

// Push the message ID into the conversation

conversation.messages.push(savedMessage.\_id);

await conversation.save(); // Save the updated conversation

// Emit the new message to both users

io.to(data.sender).emit('message', savedMessage);

io.to(data.receiver).emit('message', savedMessage);

} catch (error) {

console.error('Error handling new message:', error);

socket.emit('error', { message: 'Could not send message' });

}

});

socket.on('typing', (receiverId) => {

socket.to(receiverId).emit('typing', user.\_id);

});

// Event to handle user disconnection

socket.on('disconnect', () => {

onlineUser.delete(user.\_id.toString());

console.log('User disconnected:', user.\_id.toString());

io.emit('onlineUser', Array.from(onlineUser));

});

});

// Export the app and server (if needed for testing or further development)

module.exports = {

app,

server,

};

**Search User**

import React, { useEffect, useState } from 'react';

import { IoSearchOutline, IoClose } from "react-icons/io5";

import Loading from './Loading';

import UserSearchCard from './UserSearchCard';

import toast from 'react-hot-toast';

import axios from 'axios';

const SearchUser = ({ onClose }) => {

    const [searchUser, setSearchUser] = useState([]);

    const [loading, setLoading] = useState(false);

    const [search, setSearch] = useState("");

    const handleSearchUser = async () => {

        const URL = `${process.env.REACT\_APP\_BACKEND\_URL}/api/search-user`;

        try {

            setLoading(true);

            const response = await axios.post(URL, {

                search: search // Send the search term to the backend (email or name)

            });

            setLoading(false);

            setSearchUser(response.data.data); // Set the users returned by backend

        } catch (error) {

            setLoading(false);

            toast.error(error?.response?.data?.message || 'An error occurred');

        }

    };

    useEffect(() => {

        if (search.trim()) { // Only trigger search if the search term is not empty

            handleSearchUser();

        } else {

            setSearchUser([]); // Clear search results when the input is empty

        }

    }, [search]); // Trigger search when `search` value changes

    // Inline Styles for Animations

    const backgroundFadeAnimation = {

        animation: "backgroundFade 5s ease-in-out infinite alternate"

    };

    const buttonHoverAnimation = {

        backgroundSize: "200% auto",

        transition: "all 0.5s ease",

    };

    const inputFocusAnimation = {

        animation: "inputFocus 0.3s ease-in-out"

    };

    const avatarPulseAnimation = {

        animation: "avatarPulse 2s infinite ease-in-out"

    };

    const inputFocusKeyframes = `

        @keyframes inputFocus {

            0% {

                border-color: #ccc;

            }

            100% {

                border-color: #4f46e5; /\* Blue border \*/

                box-shadow: 0 0 5px rgba(79, 70, 229, 0.5); /\* Blue glow \*/

            }

        }

    `;

    const backgroundFadeKeyframes = `

        @keyframes backgroundFade {

            0% {

                background-color: rgba(0, 0, 0, 0.4); /\* Dark background \*/

            }

            100% {

                background-color: rgba(0, 0, 0, 0.6); /\* Lighter background \*/

            }

        }

    `;

    const avatarPulseKeyframes = `

        @keyframes avatarPulse {

            0% {

                transform: scale(1);

                opacity: 0.8;

            }

            50% {

                transform: scale(1.1);

                opacity: 1;

            }

            100% {

                transform: scale(1);

                opacity: 0.8;

            }

        }

    `;

    return (

        <div

            className="fixed top-0 bottom-0 left-0 right-0 bg-slate-700 bg-opacity-40 p-2 z-10"

            style={backgroundFadeAnimation}

        >

            <div className="w-full max-w-lg mx-auto mt-10">

                {/\* Input search by email or name \*/}

                <div className="bg-white rounded h-14 overflow-hidden flex shadow-lg hover:shadow-2xl transition-shadow duration-300">

                    <input

                        type="text"

                        placeholder="Search user by email or name"

                        className="w-full outline-none py-1 h-full px-4 transition-all duration-300 ease-in-out transform hover:scale-105 focus:ring-2 focus:ring-blue-500 focus:outline-none"

                        onChange={(e) => setSearch(e.target.value)}

                        value={search}

                        style={inputFocusAnimation}

                    />

                    <div

                        className="h-14 w-14 flex justify-center items-center bg-gradient-to-r from-purple-500 to-indigo-600 hover:scale-110 rounded-r-md transition-all"

                        style={buttonHoverAnimation}

                    >

                        <IoSearchOutline size={25} className="text-white" />

                    </div>

                </div>

                {/\* Display search results \*/}

                <div className="bg-white mt-2 w-full p-4 rounded">

                    {/\* No user found \*/}

                    {searchUser.length === 0 && !loading && search && (

                        <p className="text-center text-slate-500">No user found!</p>

                    )}

                    {/\* Loading indicator \*/}

                    {loading && (

                        <div className="flex justify-center"><Loading /></div>

                    )}

                    {/\* Display users found \*/}

                    {searchUser.length !== 0 && !loading && (

                        searchUser.map((user) => (

                            <UserSearchCard key={user.\_id} user={user} onClose={onClose} />

                        ))

                    )}

                </div>

            </div>

            {/\* Close button \*/}

            <div

                className="absolute top-0 right-0 text-2xl p-2 lg:text-4xl hover:text-white hover:scale-125 transition-all duration-300"

                onClick={onClose}

                style={avatarPulseAnimation}

            >

                <button>

                    <IoClose />

                </button>

            </div>

            {/\* Inline Keyframes \*/}

            <style>{`

                ${inputFocusKeyframes}

                ${backgroundFadeKeyframes}

                ${avatarPulseKeyframes}

            `}</style>

        </div>

    );

};

export default SearchUser;

**Message Page**

import React, { useEffect, useRef, useState } from "react";

import { useSelector } from "react-redux";

import { Link, useParams } from "react-router-dom";

import Avatar from "./Avatar";

import { HiDotsVertical } from "react-icons/hi";

import { FaAngleLeft } from "react-icons/fa";

import uploadFile from "../helpers/uploadFile";

import { IoPaperPlaneOutline, IoClose } from "react-icons/io5";

import Loading from "./Loading";

import backgroundImage from "../assets/wallapaper.jpeg";

import moment from "moment";

const MessagePage = () => {

  const params = useParams();

  const socketConnection = useSelector((state) => state?.user?.socketConnection);

  const user = useSelector((state) => state?.user);

  const [dataUser, setDataUser] = useState({

    name: "",

    email: "",

    profile\_pic: "",

    online: true,

    \_id: "",

  });

  const [openImageVideoUpload, setOpenImageVideoUpload] = useState(false);

  const [message, setMessage] = useState({ text: "", imageUrl: "", videoUrl: "" });

  const [loading, setLoading] = useState(false);

  const [allMessage, setAllMessage] = useState([]);

  const currentMessage = useRef(null);

  useEffect(() => {

    if (currentMessage.current) {

      currentMessage.current.scrollIntoView({ behavior: "smooth", block: "end" });

    }

  }, [allMessage]);

  useEffect(() => {

    if (socketConnection) {

      socketConnection.emit("message-page", params.userId);

      socketConnection.emit("seen", params.userId);

      const handleMessageUser = (data) => {

        setDataUser(data);

      };

      const handleMessage = (data) => {

        setAllMessage((prev) => [...prev, data]);

      };

      socketConnection.on("message-user", handleMessageUser);

      socketConnection.on("message", handleMessage);

      return () => {

        socketConnection.off("message-user", handleMessageUser);

        socketConnection.off("message", handleMessage);

        setAllMessage([]);

      };

    }

  }, [socketConnection, params.userId]);

  const handleUploadImageVideoOpen = () => {

    setOpenImageVideoUpload((prev) => !prev);

  };

  const handleUploadImage = async (e) => {

    const file = e.target.files[0];

    setLoading(true);

    const uploadPhoto = await uploadFile(file);

    setLoading(false);

    setOpenImageVideoUpload(false);

    setMessage((prev) => ({ ...prev, imageUrl: uploadPhoto.url }));

  };

  const handleClearUploadImage = () => {

    setMessage((prev) => ({ ...prev, imageUrl: "" }));

  };

  const handleUploadVideo = async (e) => {

    const file = e.target.files[0];

    setLoading(true);

    const uploadVideo = await uploadFile(file);

    setLoading(false);

    setOpenImageVideoUpload(false);

    setMessage((prev) => ({ ...prev, videoUrl: uploadVideo.url }));

  };

  const handleClearUploadVideo = () => {

    setMessage((prev) => ({ ...prev, videoUrl: "" }));

  };

  const handleOnChange = (e) => {

    setMessage((prev) => ({ ...prev, [e.target.name]: e.target.value }));

  };

  const handleSendMessage = (e) => {

    e.preventDefault();

    if (message.text || message.imageUrl || message.videoUrl) {

      if (socketConnection) {

        socketConnection.emit("new message", {

          sender: user?.\_id,

          receiver: params.userId,

          text: message.text,

          imageUrl: message.imageUrl,

          videoUrl: message.videoUrl,

          msgByUserId: user?.\_id,

        });

        setMessage({ text: "", imageUrl: "", videoUrl: "" });

      }

    }

  };

  const renderMessageText = (text) => {

    return text.split("\n").map((line, index) => (

      <React.Fragment key={index}>

        {line}

        <br />

      </React.Fragment>

    ));

  };

  return (

    <div style={{ backgroundImage: `url(${backgroundImage})` }} className="bg-no-repeat bg-cover">

      <header className="sticky top-0 h-16 bg-white flex justify-between items-center px-4 shadow-md">

        <div className="flex items-center gap-4">

          <Link to="/" className="lg:hidden">

            <FaAngleLeft size={25} />

          </Link>

          {/\* Display the profile picture \*/}

          <Avatar

            width={50}

            height={50}

            imageUrl={dataUser?.profile\_pic}

            name={dataUser?.name}

            userId={dataUser?.\_id}

          />

          {/\* User's name and online/offline status \*/}

          <div>

            <h3 className="font-semibold text-lg my-0 text-ellipsis line-clamp-1">

              {dataUser?.name}

            </h3>

            <p className="-my-2 text-sm flex items-center">

              <span

                className={`w-2.5 h-2.5 mr-2 rounded-full ${

                  dataUser.online ? "bg-green-500" : "bg-red-500"

                }`}

              ></span>

              {dataUser.online ? (

                <span className="text-primary">online</span>

              ) : (

                <span className="text-slate-400">offline</span>

              )}

            </p>

          </div>

        </div>

        {/\* Options menu \*/}

        <div>

          <button className="cursor-pointer hover:text-primary">

            <HiDotsVertical />

          </button>

        </div>

      </header>

      <section className="h-[calc(100vh-128px)] overflow-x-hidden overflow-y-scroll scrollbar relative bg-slate-200 bg-opacity-50">

        <div className="flex flex-col gap-2 py-2 mx-2" ref={currentMessage}>

          {allMessage.map((msg, index) => (

            <div

              key={index}

              className={`flex ${

                user.\_id === msg?.msgByUserId ? "justify-end" : "justify-start"

              } items-center gap-2`}

            >

              {user.\_id !== msg?.msgByUserId && (

                <Avatar

                  width={40}

                  height={40}

                  imageUrl={dataUser?.profile\_pic}

                  name={dataUser?.name}

                  userId={dataUser?.\_id}

                />

              )}

              <div

                className={`p-2 py-1 rounded w-fit max-w-[280px] md:max-w-sm lg:max-w-md ${

                  user.\_id === msg?.msgByUserId ? "bg-teal-100" : "bg-white"

                }`}

              >

                <div className="w-full relative">

                  {msg?.imageUrl && (

                    <img

                      src={msg?.imageUrl}

                      className="w-full h-full object-scale-down"

                      alt="message"

                    />  )}

                  {msg?.videoUrl && (

                    <video

                      src={msg.videoUrl}

                      className="w-full h-full object-scale-down"

                      control/>  )}

                </div>

                <p className="px-2" style={{ whiteSpace: "pre-wrap" }}>

                  {renderMessageText(msg.text)}

                </p>

                <p className="text-xs ml-auto w-fit">

                  {moment(msg.createdAt).format("hh:mm")}

                </p>

              </div>

              {user.\_id === msg?.msgByUserId && (

                <Avatar

                  width={40}

                  height={40}

                  imageUrl={user?.profile\_pic}

                  name={user?.name}

                  userId={user?.\_id}

                />

              )}

            </div>

          ))}

        </div>

      </section>

      <footer className="h-16 bg-white w-full flex justify-between items-center gap-4 p-3 border-t border-slate-300">

        <textarea

          name="text"

          rows="2"

          value={message.text}

          onChange={handleOnChange}

          placeholder="Write a message..."

          className="w-full resize-y border-2 border-slate-300 p-2 rounded-md outline-none focus:border-primary"

        />

        <button onClick={handleSendMessage} className="bg-primary p-2 rounded-full text-white hover:bg-primary-dark">

          <IoPaperPlaneOutline size={25} />

        </button>

      </footer>

    </div>

  );

};

export default MessagePage;

**Slide Bar :**

import React, { useEffect, useState } from 'react'

import { IoChatbubbleEllipses } from "react-icons/io5";

import { FaUserPlus } from "react-icons/fa";

import { NavLink, useNavigate } from 'react-router-dom';

import { BiLogOut } from "react-icons/bi";

import Avatar from './Avatar'

import { useDispatch, useSelector } from 'react-redux';

import EditUserDetails from './EditUserDetails';

import Divider from './Divider';

import { FiArrowUpLeft } from "react-icons/fi";

import SearchUser from './SearchUser';

import { FaImage } from "react-icons/fa6";

import { FaVideo } from "react-icons/fa6";

import { logout } from '../redux/userSlice';

const Sidebar = () => {

    const user = useSelector(state => state?.user)

    const [editUserOpen,setEditUserOpen] = useState(false)

    const [allUser,setAllUser] = useState([])

    const [openSearchUser,setOpenSearchUser] = useState(false)

    const socketConnection = useSelector(state => state?.user?.socketConnection)

    const dispatch = useDispatch()

    const navigate = useNavigate()

    useEffect(()=>{

        if(socketConnection){

            socketConnection.emit('sidebar',user.\_id)

            socketConnection.on('conversation',(data)=>{

                console.log('conversation',data)

                const conversationUserData = data.map((conversationUser,index)=>{

                    if(conversationUser?.sender?.\_id === conversationUser?.receiver?.\_id){

                        return{

                            ...conversationUser,

                            userDetails : conversationUser?.sender

                        }

                    }

                    else if(conversationUser?.receiver?.\_id !== user?.\_id){

                        return{

                            ...conversationUser,

                            userDetails : conversationUser.receiver

                        }

                    }else{

                        return{

                            ...conversationUser,

                            userDetails : conversationUser.sender

                        }

                    }

                })

                setAllUser(conversationUserData)

            })

        }

    },[socketConnection,user])

    const handleLogout = ()=>{

        dispatch(logout())

        navigate("/email")

        localStorage.clear()

    }

  return (

    <div

      className='w-full h-full grid grid-cols-[48px,1fr] text-slate-600 flex flex-col'

      style={{

        background: 'linear-gradient(to bottom, #6a11cb, #2575fc)' // Gradient background

      }}

    >

        <div className='bg-slate-100 w-12 h-full rounded-tr-lg rounded-br-lg py-5 text-slate-600 flex flex-col justify-between'>

            <div>

                <NavLink className={({isActive})=>`w-12 h-12 flex justify-center items-center cursor-pointer hover:bg-slate-200 rounded ${isActive && "bg-slate-200"}`} title='chat'>

                    <IoChatbubbleEllipses size={20} />

                </NavLink>

                <div title='add friend' onClick={()=>setOpenSearchUser(true)} className='w-12 h-12 flex justify-center items-center cursor-pointer hover:bg-slate-200 rounded' >

                    <FaUserPlus size={20}/>

                </div>

            </div>

            <div className='flex flex-col items-center'>

                <button className='mx-auto' title={user?.name} onClick={()=>setEditUserOpen(true)}>

                    <Avatar

                        width={40}

                        height={40}

                        name={user?.name}

                        imageUrl={user?.profile\_pic}

                        userId={user?.\_id}

                    />

                </button>

                <button title='logout' className='w-12 h-12 flex justify-center items-center cursor-pointer hover:bg-slate-200 rounded' onClick={handleLogout}>

                    <span className='-ml-2'>

                        <BiLogOut size={20}/>

                    </span>

                </button>

            </div>

        </div>

        <div className='w-full'>

            <div className='h-16 flex items-center'>

                <h2 className='text-xl font-bold p-4 text-slate-800'>Message</h2>

            </div>

            <div className='bg-slate-200 p-[0.5px]'></div>

            <div className='h-[calc(100vh-65px)] overflow-x-hidden overflow-y-auto scrollbar'>

                {

                    allUser.length === 0 && (

                        <div className='mt-12'>

                            <div className='flex justify-center items-center my-4 text-slate-500'>

                                <FiArrowUpLeft size={50} />

                            </div>

                            <p className='text-lg text-center text-slate-400'>Explore users to start a conversation with.</p>

                        </div>

                    )

                }

                {

                    allUser.map((conv,index)=>{

                        return(

                            <NavLink to={"/"+conv?.userDetails?.\_id} key={conv?.\_id} className='flex items-center gap-2 py-3 px-2 border border-transparent hover:border-primary rounded hover:bg-slate-100 cursor-pointer'>

                                <div>

                                    <Avatar

                                        imageUrl={conv?.userDetails?.profile\_pic}

                                        name={conv?.userDetails?.name}

                                        width={40}

                                        height={40}

                                    />

                                </div>

                                <div>

                                    <h3 className='text-ellipsis line-clamp-1 font-semibold text-base'>{conv?.userDetails?.name}</h3>

                                    <div className='text-slate-500 text-xs flex items-center gap-1'>

                                        <div className='flex items-center gap-1'>

                                            {

                                                conv?.lastMsg?.imageUrl && (

                                                    <div className='flex items-center gap-1'>

                                                        <span><FaImage/></span>

                                                        {!conv?.lastMsg?.text && <span>Image</span>}

                                                    </div>

                                                )

                                            }

                                            {

                                                conv?.lastMsg?.videoUrl && (

                                                    <div className='flex items-center gap-1'>

                                                        <span><FaVideo/></span>

                                                        {!conv?.lastMsg?.text && <span>Video</span>}

                                                    </div>

                                                )

                                            }

                                        </div>

                                        <p className='text-ellipsis line-clamp-1'>{conv?.lastMsg?.text}</p>

                                    </div>

                                </div>

                                {

                                    Boolean(conv?.unseenMsg) && (

                                        <p className='text-xs w-6 h-6 flex justify-center items-center ml-auto p-1 bg-primary text-white font-semibold rounded-full'>{conv?.unseenMsg}</p>

                                    )

                                }

                            </NavLink>

                        )

                    })

                }

            </div>

        </div>

        {/\*\*edit user details\*/ }

        {

            editUserOpen && (

                <EditUserDetails onClose={()=>setEditUserOpen(false)} user={user}/>

            )

        {

            openSearchUser && (

                <SearchUser onClose={()=>setOpenSearchUser(false)}/>

            )

        }

    </div>

  }

export default Sidebar;

**CHAPTER 9**

**TESTING**

**9.1 Introduction**

Testing is a critical phase in the software development lifecycle to ensure the application functions as expected, meets user requirements, and is secure, scalable, and reliable. For Cerebrio, testing is carried out using various strategies, including Black Box Testing and White Box Testing, along with other methodologies, to validate its robustness and efficiency.

**9.2 Types of Testing**

**9.2.1 Black Box Testing**

Black Box Testing evaluates the application's functionality without any knowledge of its internal implementation. It focuses solely on input-output verification to ensure the system meets user expectations.

* **User Authentication:** Verified login/logout with valid and invalid credentials.

**Test Case:** Verify login functionality with valid and invalid credentials.

**Test ID TC-01-Login**

Objective Verify login finctionality.

Precondition User must have an active account.

|  |
| --- |
| 1. Navigate to the login page.  2. Enter valid username and password.  3. Click the "Login" button. |
| User is redirected to the dashboard.   * **Chatbot Responses:** Tested contextual responses to various user queries. * **Real-Time Messaging:** Checked instant message delivery across users and |

**9.2.2 White Box Testing**

White Box Testing involves analyzing the internal structure, logic, and code of the application to ensure that the underlying implementation is error-free and optimized.

* **Algorithm Validation:** Verified AI chatbot's response generation logic.

**Code Validation:** AI chatbot's response generation logic.

// controllers/chatbotController.test.js

const generateResponse = require('../services/chatbotService');

test('should generate a valid response for a given query', () => {

  const query = 'Hello';

  const result = generateResponse(query);

  expect(result).toContain('Hello');

});

* **File Upload Validation**

const uploadFile = require('../services/fileService');

test('should upload a valid file successfully', () => {

  const file = { name: 'document.pdf', size: 1024, type: 'application/pdf' };

  const result = uploadFile(file);

  expect(result).toEqual({ success: true, message: 'File uploaded successfully.' });

});

test('should fail for unsupported file type', () => {

  const file = { name: 'malicious.exe', size: 2048, type: 'application/exe' };

  const result = uploadFile(file);

  expect(result).toEqual({ success: false, message: 'Unsupported file type.' });

});

* **Database Query Optimization:** Ensured all database queries are efficient and return accurate results.
* **Code Coverage:** Checked critical backend functions like WebSocket connections for real-time messaging.

**Sample White Box Test Case:**

const uploadFile = require('../services/fileService');

test('should upload a valid file successfully', () => {

const file = { name: 'document.pdf', size: 1024, type: 'application/pdf' };

const result = uploadFile(file);

expect(result).toEqual({ success: true, message: 'File uploaded successfully.' });});

test('should fail for unsupported file type', () => {

const file = { name: 'malicious.exe', size: 2048, type: 'application/exe' };

const result = uploadFile(file);

expect(result).toEqual({ success: false, message: 'Unsupported file type.' });});

**9.2.3 Unit Testing**

* **Objective:** Validate individual functions, components, or modules.
* **Tools Used:** Jest, Mocha, Chai.
* **Example: User Registration**

// controllers/userController.test.js

const { registerUser } = require('../controllers/userController');

const User = require('../models/userModel');

jest.mock('../models/userModel'); // Mock User model

test('should register a new user successfully', async () => {

   const req = { body: { name: 'John Doe', email: 'john@example.com',

password: 'password123' } };

   const res = { status: jest.fn().mockReturnThis(), json: jest.fn() };

   User.findOne = jest.fn().mockResolvedValue(null); // Mock no existing user

   User.prototype.save = jest.fn().mockResolvedValue(req.body);

   await registerUser(req, res);

   expect(res.status).toHaveBeenCalledWith(201);

  expect(res.json).toHaveBeenCalledWith({ message: 'User registered

successfully' });

});

**9.2.4 Integration Testing**

* **Objective:** Ensure modules interact as intended.
* **Tools Used:** Postman, Cypress.

**Example: Chatbot API**

// test/chatbotController.test.js

const request = require('supertest');

const app = require('../server'); // Express app

describe('POST /api/chatbot/chat', () => {

  it('should return a valid chatbot response', async () => {

    const response = await request(app)

      .post('/api/chatbot/chat')

      .send({ query: 'Hello' });

    expect(response.status).toBe(200);

    expect(response.body.response).toContain('Hello');

  });

**CONCLUSION**

The successful design and development of our project on Web Based Chat Application using React mark a significant milestone in our journey.

CEREBRIO, the web-based chat application with AI-integrated chatbot functionality, revolutionizes digital communication by delivering intelligent, real-time, and personalized interactions. Powered by advanced AI technologies, it offers features like automated responses, contextual understanding, and user behavior insights, making it an efficient tool for customer service, collaboration, and personal assistance.

With robust security measures and compliance with data privacy standards, CEREBRIO ensures trust and reliability. Its innovative AI-driven capabilities position it as a scalable and user-friendly solution for modern communication needs, fostering engagement and productivity across diverse applications.

By prioritizing data security and user trust, CEREBRIO delivers a reliable, scalable, and efficient solution, redefining digital messaging for modern needs.

Through this endeavor, we have gained invaluable experience in website development, honing our skills and understanding of the intricacies of creating engaging online platforms. As we move forward, we anticipate further growth and learning opportunities, leveraging our newfound expertise to continue innovating and delivering exceptional experiences for our users

**REFERENCES**

**[1] United Nations, "Goal 9: Industry, Innovation and Infrastructure," *United Nations Sustainable Development Goals*. [Online]. Available:** [**https://sdgs.un.org/goals/goal9**](https://sdgs.un.org/goals/goal9)

**[2] J. Eisenstein, *Introduction to Natural Language Processing*. Cambridge, MA: MIT Press, 2019.**

**[3] Node.js Foundation, "Node.js Documentation." [Online]. Available:** [**https://nodejs.org**](https://nodejs.org)

**[4] Express.js, "Express.js Documentation." [Online]. Available:** [**https://expressjs.com**](https://expressjs.com)

**[5] React, "React.js Documentation." [Online]. Available:** [**https://reactjs.org**](https://reactjs.org)

**[6] MongoDB Inc., "MongoDB Documentation." [Online]. Available:** [**https://www.mongodb.com**](https://www.mongodb.com)

**[7] OpenAI, "OpenAI API Documentation." [Online]. Available:** [**https://openai.com**](https://openai.com)

**[8] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ: Pearson, 2020.**

**[9] Cloudinary, "Cloudinary Documentation." [Online]. Available:** [**https://cloudinary.com**](https://cloudinary.com)