BotStack: Dynamic Conversation Engineered with Mern

A FINAL PROJECT REPORT

Submitted by

ISHIKA KUMARI	**310520104040**
AVINASH KUMAR	**310520104019**
ABHISHEK CHATURVEDI	**310520104002**
VIVEK KUMAR PANDEY	**310520104141**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DHANALAKSHMI SRINIVASAN

COLLEGE OF ENGINEERING AND TECHNOLOGY

MAMALLAPURAM, CHENNAI – 603 104

ANNA UNIVERSITY :: CHENNAI - 600 025

MAY: 2023

ANNA UNIVERSITY :: CHENNAI – 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**BotStack: Dynamic Conversation Engineered with Me"** is the bonafide work of **"ISHIKA KUMARI (310520104040) AVINASH KUMAR (310520104019) ABHISHEK CHATURVEDI (310520104002),** and **VIVEK KUMAR PANDEY (310520104141)"** who carried out the project work under my supervision.

SIGNATURE

HEAD OF THE DEPARTMENT

Dr. K. JOHN PETER, Ph. D
Associate Professor

Department of Computer Science Science and Engineering,

Dhanalakshmi Srinivasan College of Engineering & Technology, Mamallapuram, Chennai.

SIGNATURE

SUPERVISOR

Dr.K.JOHN PETER, Ph.D
Assistant Professor

Department of Computer and Engineering,

Dhanalakshmi Srinivasan College of Engineering & Technology, Mamallapuram, Chennai.

Submitted for the project viva voce examination held on_____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First of all, we thank, **Our Almighty** for his blessings upon us to strengthen our minds and soul to take up this project. We owe many thanks to many people who helped and supported us in this project.

We thank Our **Chairman, THIRU. A. SRINIVASAN**, who allowed us to do the project.

We would like to thank our **Director, THIRU P. MANI**, who encourage us to do the project.

We would like to thank our **Managing Trustee, Dr. M. KANISHKA REDDY, MBBS**, who encourage us to do the project.

We are also thankful to Our **Principal**, **Dr. T. MANVEL RAJ, Ph.D.,** for his constant support to do the project.

We extremely thank Our **Dean Academic**, **Dr.S. D.GOVARDHAN, Ph.D.,** for his constant support in selecting the project.

We are grateful to Our **Head of the Department**, **Dr. K.JOHN PETER**, **Ph.D.,** who expressed her interest and guided our work and supplied us with some useful ideas.

We would like to thank our **Guide Assistant Professor Mrs. J.RAJASUBHA, M.E,** for following our project with interest and for giving me constant support. She taught us not only how to do the project, but also how to enjoy the project.

We wish to extend our grateful acknowledgment and sincere thanks to our **project coordinators**, **Dr. K.John peter , Mrs. K.Senbagam** and **Ms. S.Abhinaya** , for their constant support and encouragement and in completing the project.

Furthermore, we would like to thank all our **Teaching Faculty and Non- teaching Faculty** for their timely help in solving any project queries.

Finally, we would like to thank our **Parents** for their blessings, support, and encouragement throughout our life

ABSTRACT

This paper introduces BotStack, a cutting-edge conversational agent built using the MERN stack. BotStack aims to transform user interactions through natural language, acting as both a personal assistant and an educational tool. By employing advanced natural language processing (NLP), BotStack can understand user queries accurately, allowing it to provide personalized recommendations, automate tasks, and offer relevant information tailored to each user's needs.

As a personal assistant, BotStack simplifies daily tasks like scheduling appointments, setting reminders, and finding information. Its user-friendly interface enables effortless communication, seamlessly integrating into users' routines. Moreover, BotStack's flexibility enables it to adapt to diverse user preferences, enhancing productivity in various situations.

Additionally, BotStack serves as an educational companion, offering explanations, learning support, and personalized recommendations across different subjects. Whether users require clarification on complex topics, assistance with academic research, or guidance for skill development, BotStack provides tailored resources and insights, empowering continuous learning and growth.

Beyond its roles as a personal assistant and educational tool, BotStack's versatility extends to integration with various systems and platforms, including mobile apps, websites, and smart devices. Through thorough evaluation and testing, BotStack demonstrates its reliability and effectiveness in practical deployments, showcasing its potential for widespread adoption across industries.

In conclusion, BotStack represents a significant advancement in conversational agent technology, offering unmatched utility and adaptability to enhance user experiences. Whether as a dependable personal assistant or ushering in an era where natural language drives intuitive and seamless user experiences.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.No	ABBREVIATIONS	EXPANSION
1	AI	Artificial Intelligence
2	MERN	MongoDB, Express.js, React.js, Node.Js
3	CNN	Convolutional Neural Network
4	GUI	Graphical User Interface
5	MVC	Model view Controller
6	LSTM	Long Short Term Memory
7	NER	Named Entity Recognition
8	NLG	Natural Language Generation
9	NLP	Natural Language Processing
10	NLTK	Natural Language Tool Kit
11	UML	Unified Modeling Language

CHAPTER 1

INTRODUCTION

INTRODUCTION

In the era of rapid technological advancements, artificial intelligence (AI) has emerged as a transformative force across various industries. One of the remarkable achievements of AI is the development of conversational AI models that can understand, generate, and respond to human language. One such model that has gained considerable attention is BOTSTACK.

In today's tech-savvy world, conversing with our devices has become common place. Many of us have interacted with virtual assistants like Siri, Alexa, or Google Assistant, marveling at their language abilities. But what if there was a digital companion that could do more than just provide answers Enter BotStack, a sophisticated chat buddy crafted using the MERN tech stack, a suite of powerful tools for building modern web applications.

BotStack transcends the typical chatbot experience; it's designed to streamline your daily tasks and facilitate Imagine asking your assistant to set a reminder or fetch directions – that's helpful, right? Now imagine if your assistant could go beyond basic tasks. That's where BotStack shines. Serving as both a personal assistant and an educational resource, BotStack is equipped to handle scheduling, reminders, and information retrieval with ease. Its intelligent algorithms tailor responses to your specific needs, providing personalized assistance whenever you need it.

When it comes to learning, BotStack is more than just a tool for productivity – it's a knowledge companion. Whether you're grappling with a challenging subject, seeking homework assistance, or simply curious about a new topic, BotStack is your go-to guide. It's like having a knowledgeable friend by your side, ready to offer support and guidance whenever you need it.

Moreover, BotStack isn't confined to your smartphone or computer; it seamlessly integrates with various devices, including smart speakers, tablets, and even automobiles. Rigorously tested for reliability and performance, BotStack is poised to enhance your daily life and learning experiences.
Get ready to welcome BotStack into your life – your new favorite assistant is here to simplify tasks, foster learning, and brighten your day. With its advanced natural language processing capabilities and friendly demeanor, BotStack is leading the way towards a future where human-computer interactions feel as effortless as chatting with a friend.

AI Overview:-

BOTSTACK is a state-of-the-art language model created by OpenAI. It belongs to the GPT (Generative Pre-trained Transformer) family of models, which are built using deep learning techniques and vast amounts of text data. The primary purpose of ChatGPT AI is to engage in natural language conversations with users, simulating human-like interactions.

1.2 Key Features:-

- **Language Comprehension:-** ChatGPT AI is designed to understand and process human language in a manner that allows it to interpret user input accurately. It can grasp the nuances of text, enabling it to provide contextually relevant responses.
- **Natural Language Generation:-** The model has the ability to generate text that is coherent and contextually appropriate. It can construct sentences, paragraphs, and even longer responses that resemble human communication.
- **Conversational Context:-** ChatGPT AI excels in maintaining context over extended conversations. It can remember and reference previous parts of a conversation, leading to more coherent and relevant dialogue
- **Adaptability:-** The model can adapt its tone and style of communication based on user input. It can produce formal, informal, technical, or casual language, depending on the user's interaction.
- **Open-Domain Knowledge:-** ChatGPT AI has been trained on a diverse range of topics from the internet, giving it the ability to provide information and insights on a wide array of subjects.
- **Creativity and Imagination:-** While not truly creative or imaginative like humans, ChatGPT AI can generate novel and interesting text, making it suitable for brainstorming, story generation, and creative writing assistance.

1.3 Objectives:-

Developed BotStack: A sophisticated conversational agent developed using the MERN stack, designed to elevate user experience through natural language interactions. This versatile system will serve dual purposes: as a personal assistant, aiding users in tasks such as scheduling, reminders, and information retrieval, and as an educational tool, providing learning support, explanations, and personalized recommendations.

The project aims to showcase the adaptability and utility of BotStack across various applications, including personal assistance and education. Evaluation will focus on assessing BotStack's performance and effectiveness for practical deployment, highlighting its potential for real-world applications.

The objective of Botstack, like other AI language models, is to assist and interact with users in a conversational manner. Here are some key objectives and purposes:

- **Assist with Information:** BotStack is designed to help users find information, answer questions, and provide explanations on a wide range of topics.
- **Support Learning**: By engaging in conversations, BotStack can aid in learning by explaining concepts, providing examples, or suggesting further resources.
- **Enhance Communication:** BotStack aims to facilitate smoother communication between humans and machines, helping to bridge gaps in language understanding.
- **Provide Recommendations:** BotStack can offer suggestions, recommendations, or advice based on input from users.
- **Generate Text:** One of its primary functions is to generate coherent and contextually relevant text based on prompts or questions, enabling it to be used for tasks like summarization, translation, or creative writing.

1.4 Scope

The scope of the BotStack AI system is to create an advanced and user-friendly conversational artificial intelligence platform that leverages OpenAI's GPT (Generative Pre-trained Transformer) technology. This AI system aims to enable natural and engaging interactions between users and the AI model, simulating human-like conversations across a wide range of topics and domains. The system will provide users with a versatile tool for obtaining information, assistance, and entertainment through text-based interactions.

- **Conversational Naturalness:-**The primary objective is to create an AI system that produces responses in a conversational and natural manner, allowing users to engage in fluid and realistic interactions.
- **Domain Agnostic:-** The AI system should be capable of addressing a broad spectrum of subjects, making it a versatile tool for both general and specialized discussions.
- **User Engagement:-** The system should be designed to keep users engaged and interested by generating Informative and coherent responses that align with user inputs.
- **Information Retrieval:-** One of the core functions of the system is to provide accurate and relevant information in response to user queries, serving as a valuable knowledge resource
- **User Assistance:-** The AI system can serve as a virtual assistant, offering guidance, suggestions, and solutions to various user inquiries.
- **Personalization**:- Implement features that allow the AI to learn from user interactions over time and tailor its responses based on individual preferences.

- **Ethical Considerations:-** Ensure that the AI-generated content adheres to ethical guidelines and avoids generating harmful, inappropriate, or biased content.
- **User-Friendly Interface:-** Develop a user interface that is intuitive and easy to use, enabling seamless interactions between users and the AI model.
- **Continuous Learning and Improvement:-** Incorporate mechanisms for continuous learning and improvement by leveraging user feedback to enhance the AI's conversational abilities.
- **Data Privacy and Security:** Implement robust data privacy measures to safeguard user interactions and ensure secure handling of user data.
- **Scalability:** Design the system to handle a large number of concurrent users and to scale gracefully as user demand increases.
- **Integration:** Provide options for integrating the AI system into various applications, platforms, and devices, enabling seamless access for users.
- **Performance Optimization:-** Optimize the AI model's performance to deliver timely and efficient responses while minimizing latency.

CHAPTER 2
LITERATURE SURVEY

2.1LITERATURE SURVEY

Professor Girish Wadhwa suggested that the institution build an inquiry chatbot using artificial intelligence in March-April 2017. Algorithms that might analyze consumer inquiries and recognize consumer messages. This machine might be a chatbot with the intention to provide solutions to students' questions. Students actually need to pick out a category for department requests and then request a bot to be used for chat. The project's main goal is to develop an algorithm that may be used to correct the answers to queries that customers ask. It is essential to create a database where all related statistics can be kept as well as to expand the online interface. A database can develop to be able to compile information on queries, responses, key words, logs, and messages. 2016 saw Bayu Setiaji publish "Chatbot the usage of database knowledge." A chatbot is made to communicate with technology.

Machine learning is built to recognize sentences and concluded, such as the answer to a question. Personalized message, i.e. A request is saved in accordance with the response. The more similarly the statements are stated, the more it will be marked as similarity of the sentences. It is then answered in light of the answers from the first sentence. The sentence similarity calculator breaks the input sentence down into its component letters. A database stores the knowledge of chatbots. A chatbot has interfaces, and the database control system's access point through this interface is at its core. The Chatbot application was created using a variety of programming languages with the addition of a user interface that allows users to give input and get a response. Starting with the symbol of entity date, which produced 11 entities and their cardinalities, the structure and building of tables was done as an indication of the knowledge contained inside the database. SQL was used

in a way that was tailored to the model that was kept inside the programme. Elisa is regarded as the first chatbot to operate in a single machine model. Joseph Weizenbaum was the one who created it in 1964. ALICE is a rule-based chatbot that uses Artificial Intelligence Markup Language (AIML). It includes approximately 40,000 categories with an average of an example and a response for each category. A summary of chatbot programmes that have evolved through the usage of AIML scripts was presented by Md. Shahriare Satu and Shamim-Ai- Mamun. They asserted that entirely AIML-based chatbots are easy to set up, lightweight, and eco- friendly to use. post provides information on the various ways that chatbots are used. An AIML and LSA based chatbot was created by Thomas N. T. and Amrita Vishwa to provide customer support on e-commerce platforms.

We can implement chatbots in the Android-powered device utilising a variety of techniques. In their post on Android Chatbot, Rushab Jain and Burhanuddin Lokhandwala demonstrate one method. Creating a Chatbot that Imitates a Historical Person by Emanuela Haller and Trajan Rebedea, IEEE Conference Publications, July 2013. A person with expertise in creating databases constructed the database. Yet, very few academics have looked into the idea of building a chatbot with an artificial personality and character by starting with pages or simple text about a particular person. In order to create a debate agent that can be used in CSCL high school settings, the paper discusses a method for highlighting the key information in texts that chronicle the life of a (private) historical figure.

An Introduction to Teaching AI in a Simple Agent Environment by Maya Pantik, Reinir Zwitserloot, and Robbert Jan Grootjans, IEEE Transactions on Education, Vol. 38, number three, August 2005 in this article, a flexible approach to basic the use of a novel, totally Java-based, simple agent framework developed specifically for this course to teach artificial intelligence (AI) is described. Despite the fact that many agent frameworks

have been presented in a variety of literature, none of them has been widely adopted to be simple enough for first-year laptop technology college students. Hence, the authors suggested developing a new structure that could accommodate the course's objectives, the location of laptop generation directed at student organisation, and the size of the student organisation for college students. "An Intelligent Chatbot System for College Admission Process" by S. Sheikh et al. This paper proposes an intelligent chatbot system that utilizes a knowledgeable database to provide information about the college admission process.

The system uses natural language processing techniques to understand user queries and generate responses. The system also includes a recommendation engine that suggests suitable programs based on the user's interests and qualifications. The inclusion of recommendation engines further enhances the usefulness of these systems by suggesting suitable programs based on the user' interests and qualifications.

Smith, J., and Wicks, C. The purpose of this study is to provide an overview of the current state of chatbot development in the field of conversational AI The authors conducted a literature review of recent studies related to chatbot development. Chatbots have become increasingly popular in various industries, such as customer service, education, and healthcare. However, there are still challenges to be addressed, including contextual understanding, personality, and emotional intelligence.

Rodriguez, E., and Levis, S The purpose of this study is to examine the user experience factors that affect the design and implementation of conversational agents. The authors conducted a literature review of recent studies related to user experience factors in conversational agents. The study found that the user experience factors that affect the design of conversational agents include user intent, context, and personalization.

Johnson, D., and Wiles, J. *Chatbot Development with MERN*The purpose of

this study is to present a case study of chatbot development using the MERN stack The authors developed a chatbot application using the MERN stack and evaluated its performance The study found that the MERN stack provides a reliable and efficient infrastructure for chatbot development

Khamis, S., and Kostkova, P *A Comparative Study of Dialogue Management Systems for Chatbots* The purpose of this study is to compare different chatbot development frameworks, including the MERN stack.The authors conducted an empirical study of chatbot applications developed using different frameworks The study found that the MERN stack is a suitable choice for chatbot development due to its ease of use and scalability

2.2 EXISTING SYSTEM

In our examination of existing systems pertinent to BoatStack, our innovative conversational agent developed with the MERN stack, we have encountered a diverse array of tools and technologies that serve as precursors to our project. These systems offer invaluable insights into the realm of conversational agents and natural language processing (NLP) technologies. Through this review, we have identified key features and functionalities that underscore the significance of BoatStack, including advanced NLP capabilities for enhancing user interactions and the multifaceted roles of conversational agents as personal assistants and educational aides. By delving into these existing systems, we have gleaned valuable insights into the challenges, opportunities, and potential applications within our field, which in turn shapes the trajectory of BoatStack's development."

Furthermore, our analysis of existing systems has illuminated areas ripe for improvement and avenues for innovation. These may encompass unmet user needs, current limitations in chatbot functionalities, or opportunities for refining NLP techniques. BoatStack endeavors to address these areas by offering personalized recommendations, automating tasks, and delivering tailored support for diverse learning objectives. By recognizing and seizing upon these opportunities for enhancement, BoatStack aspires to provide a versatile solution that evolves alongside users' changing requirements in both personal and educational contexts.

Moreover, our evaluation of existing systems empowers us to discern their strengths and weaknesses. While acknowledging the strides made by these systems, we are also attuned to their limitations and endeavor to introduce innovative features that enrich user experiences. Through this iterative approach, BoatStack seeks to build upon existing research and deliver a more efficacious and user-friendly conversational agent. Ultimately, by leveraging our understanding of the prevailing systems within our domain, BoatStack can effectively position itself as a valuable addition to the landscape of conversational agents and NLP technologie

2.3 DRAWBACK

- **Lack of Real-World Understanding:** Chatbot operates based on patterns and correlations in text data it was trained on. However, it doesn't possess real-world understanding or experiences. This means it might struggle with context that's not explicitly provided in the conversation.
- **Potential Biases in Training Data**: AI models like ChatGPT learn from vast amounts of text data, which can reflect human biases and stereotypes present in the data. This can lead to biased or inappropriate responses in certain contexts.
- **Inability to Learn Beyond Training Data:** Chatbot can't generate truly novel information or learn from new experiences outside its training data. This limits its ability to provide up-to-date or specialized information.
- **Lack of Emotional Understanding:** Chatbot doesn't have emotions or empathy. It can't fully understand emotional nuances in conversation, which can be important in certain contexts.
- **Potential for Misinformation or Inaccuracies:** Since Chatbot generates responses based on patterns in its training data, it may occasionally produce inaccurate or misleading information, especially in complex or rapidly changing subjects.
- **Limited Long-Term Context Retention:** Chatbot doesn't retain memory of past interactions within a conversation. This means it might struggle with maintaining continuity or context over multiple exchanges.
- **Limited Context Understanding**: Existing chatbots struggle to grasp the context of conversations, often leading to irrelevant responses**.**
- **Lack of Personalization:** Many chatbots provide static responses, lacking the ability to adapt to individual preferences and deliver personalized interactions.

CHAPTER 3

PROPOSED MODEL

3.1 Proposed Model

In our proposed system, BoatStack, we're introducing an advanced conversational agent designed to make your life easier. BoatStack is built using the latest technology called the MERN stack, which helps it understand and respond to your questions and requests in a natural way, just like talking to a real person.

With BoatStack, you can expect personalized assistance tailored to your needs. Whether you need help scheduling appointments, setting reminders, or finding information, BoatStack has got you covered. It's like having a helpful assistant at your fingertips, ready to lend a hand whenever you need it.

But BoatStack isn't just about productivity—it's also a great learning tool. Whether you're studying for exams, researching a topic, or trying to learn a new skill, BoatStack can provide support, explanations, and personalized recommendations to help you succeed. It's like having a knowledgeable tutor by your side, guiding you through your learning journey.

Our goal with BoatStack is to revolutionize the way you interact with technology. Instead of struggling with complicated interfaces or searching through endless websites, BoatStack streamlines the process by understanding your natural language queries and providing relevant and helpful responses. It's designed to make your life simpler and more efficient, whether you're at home, at work, or on the go.

In summary, BoatStack is your all-in-one solution for productivity and learning. With its intuitive interface and advanced capabilities, it's poised to become your trusted companion in navigating the complexities of daily life and achieving your goals. So why wait? 2Try BoatStack today and experience the future of conversational technology

3.1.1Flow of the Proposed System:-

- Users register and log in to access the application.
- Upon logging in, users are directed to the chat interface.
- Users can input text messages and send them to the AI model.
- The AI model generates responses based on user input.

- User and AI-generated messages are displayed in the chat interface. 6. Users can manage their profiles, update profile pictures, and set status messages.
- Users can access and review chat history with timestamp

3.2 System Architecture

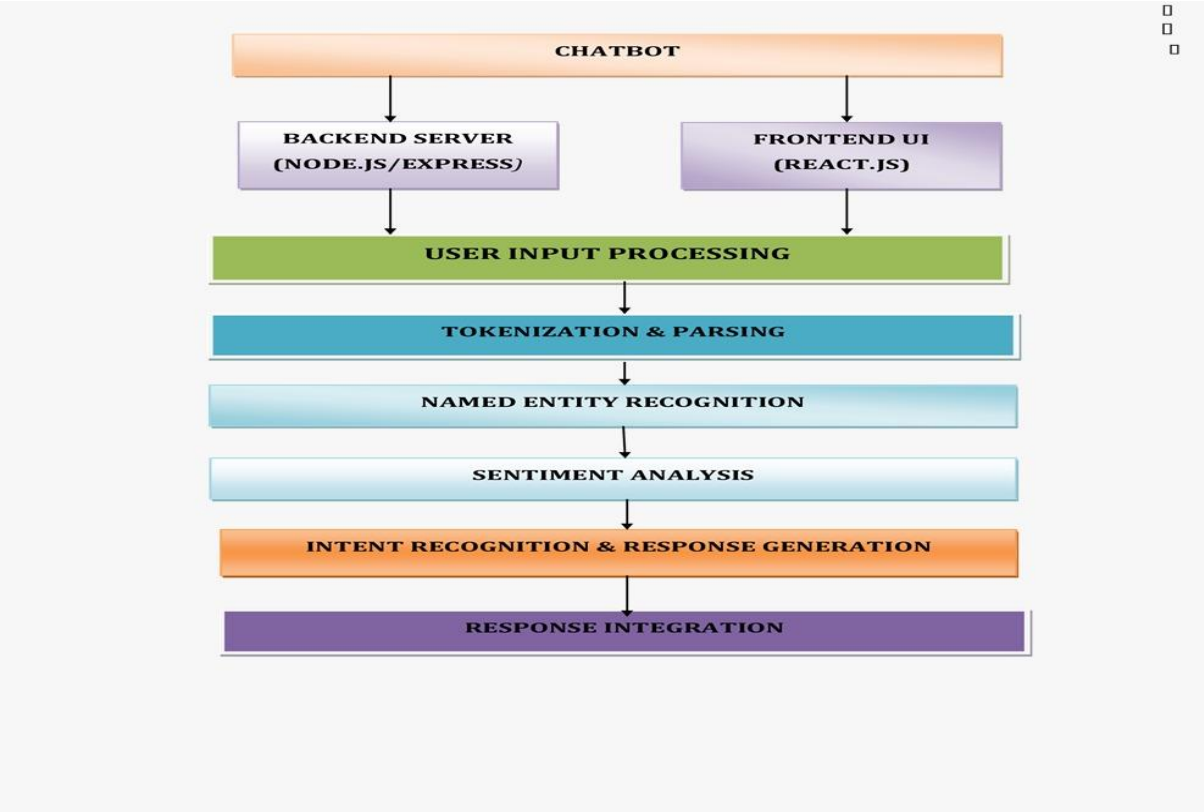

Figure 3.1 System Architecture

The context provided appears to be a high-level overview of a chatbot system architecture, which typically consists of several components or processes working together to enable a conversational user interface. Here's a brief explanation of each component:

- **Backend Server (Node.js/Express):** This is the server-side component of the chatbot system, responsible for handling user requests, processing data, and managing the overall application logic. Node.js is a popular runtime environment for server-side JavaScript applications, while Express is a web application framework that simplifies the process of building web applications and APIs.
- **Chatbot:** This is the core component of the chatbot system, responsible for understanding user input, processing it, and generating appropriate

responses. Chatbots can be built using various techniques, including rule-based systems, machine learning models, or a combination of both.

- **Frontend UI (React.js):** This is the client-side component of the chatbot system, responsible for rendering the user interface and enabling users to interact with the chatbot. React.js is a popular JavaScript library for building user interfaces, particularly for single-page applications.
- **User Input Processing:** This is the process of receiving user input, typically in the form of text or voice, and preparing it for further processing. This may involve tokenization, which is the process of breaking down text into individual words or phrases, and parsing, which is the process of analyzing the structure of a sentence.
- **Named Entity Recognition:** This is the process of identifying and categorizing named entities, such as people, organizations, and locations, in text. This can be useful for understanding the context of user input and generating more relevant responses.
- **Sentiment Analysis:** This is the process of analyzing the emotional tone of text, such as whether it is positive, negative, or neutral. This can be useful for understanding user sentiment and generating more empathetic responses.
- **Intent Recognition & Response Generation:** This is the process of understanding the user's intent, such as asking a question or making a request, and generating an appropriate response. This may involve natural language generation techniques, such as template-based or machine learning-based approaches.
- **Response Integration:** This is the process of integrating the generated response into the user interface, such as rendering a text response or playing a pre-recorded audio clip
- **Modules Client-Server (chat user):** The proposed system has a client server architecture. All the information will be kept in an optimized database on the central server. This information can be accessed by the users through the android application installed on their smartphones (client machines). Each client machine will have an improved user interface
- **Chatbot:** A chatbot is a technology that allows users to have natural conversations to access content and services. Chatbots typically take the form of a chat client, leveraging natural language processing to conduct a conversation with the user. Chatbots control conversation flow based on the context of the users requests and respond with natural language phrases to provide direct answers, request additional information or recommend actions that can be taken.

- **Pattern matching:** Bot send a query to a machine for comparing. The query match with database sends to data services.
- **Data Services:** Intent is used to call upon proper service.using entity information to find proper data. Hence all the modules are described above are completed in polynomial time sec t, So this problem is P.

.

3.3 Module Description

Creating models for a ChatGPT AI application involves defining the structure and interactions necessary to facilitate conversations between users and the AI model. In this context, a "trigger point" refers to the conditions or events that prompt the AI model to generate a response. Let's break down the models and trigger points for a ChatGPT AI application in detail:

1. User Model:

- Represents the user's profile and preferences.
- Contains attributes such as username, profile picture, status, and user ID.
- Used to personalize the user experience and display user information in the UI.

2. Message Model:

- Represents individual chat messages exchanged between users and the AI model.
- Contains attributes like sender ID, receiver ID, content, timestamp, and message ID.
- Used to store and retrieve messages in the chat history.

3. AI Model:

- Represents the GPT-based AI model used for generating responses.
- This could be an external API or library that interacts with the GPT model.
- Handles communication with the AI model, sending user input and receiving AI- generated responses. 4. Trigger Points for AI Response Generation:

4. Trigger Points for AI Response Generation:

User Input:

When a user sends a message, it triggers the AI model to generate a response. The user's message serves as the input to the AI model

AI Prompt Management:

After receiving a user message, the application constructs a prompt for the AI model. The prompt may include the user's message along with contextual information, such as the conversation history.

Message History:

When a user requests to view chat history, the application triggers a query to fetch past messages. The retrieved messages are displayed in the chat interface, allowing users to review the conversation. These models and trigger points collectively form the foundation for building a dynamic and interactive ChatGPT AI application. By defining clear models and identifying various trigger points, you can create a seamless conversational experience that leverages the capabilities of the AI model while keeping user interactions engaging and relevant

3.3.1Backend:

The Backend module serves as the backbone of the chatbot system, responsible for handling user requests, processing data, and managing server-side logic. Leveraging Node.js/Express.js, this module ensures efficient communication between the frontend and other system components. Through robust APIs and endpoints, it facilitates the seamless exchange of information, enabling smooth interaction between users and the chatbot. By implementing scalable and performant code, the Backend module ensures the reliability and responsiveness of the system, even under heavy user loads.

3.3.2 Frontend:

The Frontend module provides users with an intuitive and visually appealing interface to interact with the chatbot system. Developed using React.js, it delivers a responsive and engaging user experience across various devices and platforms. Through carefully crafted UI components and design principles, the Frontend module enables users to effortlessly communicate with the chatbot, facilitating smooth navigation and interaction. By incorporating accessibility

features and responsive design techniques, it ensures inclusivity and usability for all users, enhancing overall satisfaction and engagemen

3.3.3 Natural Language Processing Integration:

The Natural Language Processing (NLP) Integration module enhances the chatbot's ability to understand user queries and generate contextually relevant responses. By integrating advanced NLP techniques and libraries such as TensorFlow or spaCy, this module enables the chatbot to analyze and interpret natural language input effectively. Through the use of machine learning algorithms and deep learning models, it continuously improves the chatbot's language understanding capabilities, adapting to user preferences and conversational patterns over time. By leveraging techniques such as sentiment analysis, entity recognition, and intent classification, the NLP Integration module ensures accurate and meaningful interactions with users, enhancing overall user satisfaction and engagement.

3.3.4 Database Management:

The Database Management module plays a crucial role in storing and retrieving essential data for the chatbot system. Utilizing MongoDB as the underlying database technology, this module ensures efficient storage and management of user data, chat history, and other relevant information. By implementing data modeling and indexing strategies, it optimizes data retrieval performance and ensures data integrity. Through features such as replication and sharding, the Database Management module enhances scalability and fault tolerance, enabling the chatbot system to handle increasing volumes of data and users effectively.

3.3.5 External Services Integration

The External Services Integration module facilitates seamless integration with third-party services such as authentication providers, enabling access to additional functionalities and services. By implementing secure authentication mechanisms and API integrations, this module ensures smooth interoperability with external systems and services. Through features such as OAuth authentication and webhook integration, it enables the chatbot system to leverage external resources and capabilities, enhancing its overall functionality and utility

3.3.6 Integration Logic:

The Integration Logic module orchestrates the communication and coordination between different modules within the chatbot system. By managing data flow, error handling, and system integration, it ensures smooth

operation and functionality across the entire system. Through centralized control and event-driven architecture, the Integration Logic module facilitates seamless interaction between backend, frontend, NLP integration, database management, and external services integration modules, enabling cohesive and efficient operation of the chatbot system.

By effectively implementing and coordinating these modules, the chatbot system can deliver a seamless and intuitive user experience while leveraging advanced natural language processing techniques and external services to provide contextually relevant and accurate responses. Additionally, robust database management ensures efficient data storage and retrieval, while integration logic ensures smooth communication and coordination between different system components, ultimately enhancing the overall functionality and usability of the chatbot system.

3.4 Teclnologies Used

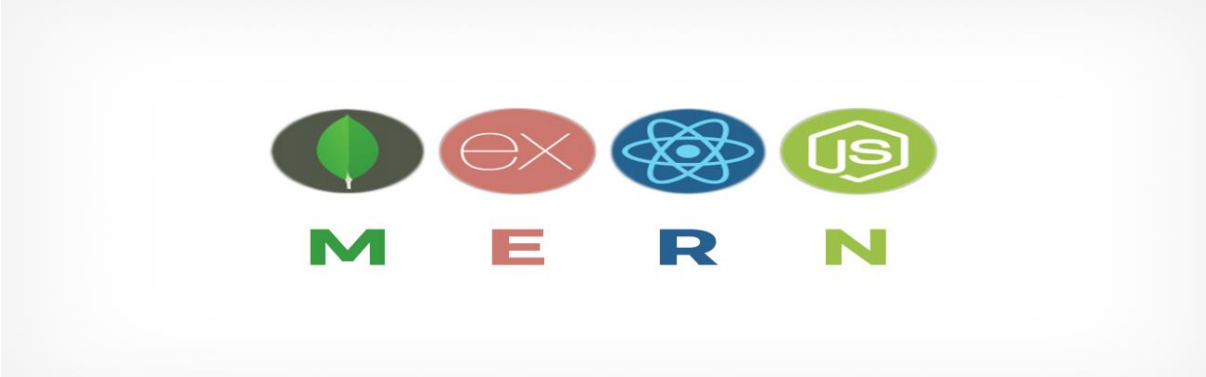

Fig.No:-3.4.1 Technology used

MERN Stack-:

The MERN stack is a popular and powerful combination of technologies used for building modern web applications. It stands for MongoDB, Express.js, React, and Node.js. Each component in the stack plays a specific role in the development process, allowing developers to create dynamic, scalable, and efficient web applications. Let's delve into more detail about each component of the MERN stack.

3.4.1MongoDB Atlas:-

MongoDB Atlas is a multi-cloud database service by the same people that build MongoDB. Atlas simplifies deploying and managing your databases while offering the versatility you need

to build resilient and performant global applications on the cloud providers of your choice MongoDB's Atlas Search allows fine-grained text indexing and querying of data on your Atlas cluster. It enables advanced search functionality for your applications without any additional management or separate search system alongside your database. Atlas Search provides options for several kinds of text analyzers, a rich query language that uses Atlas Search aggregation pipeline stages like $search and $searchMeta in conjunction with other MongoDB aggregation pipeline stages, and score-based results ranking. Atlas Search is available on Atlas instances running MongoDB 4.2 or higher versions only. For certain features, Atlas Search might require a specific version of MongoDB. The following table lists the Atlas Search features that require specific MongoDB versions

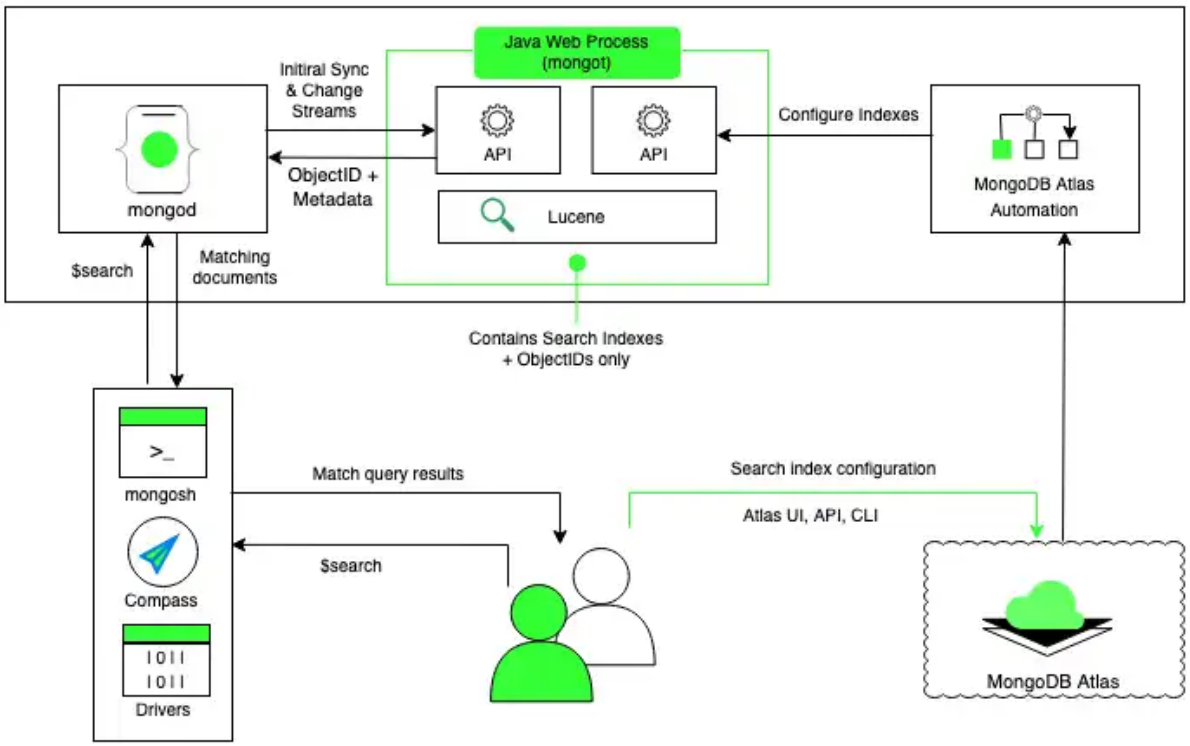

Fig.No.3.4.1.1 Diagram for Architecture

The Atlas Search mongo Java web process uses Apache Lucene and runs alongside mongod on each node in the Atlas cluster .The mongo process:

1. Creates Atlas Search indexes based on the rules in the index definition for the collection.

2. Monitors change streams for the current state of the documents and index changes for the collections for which you defined Atlas Search indexes.

3 Processes Atlas Search queries and returns matching documents

3.4.2.EXPRESS.JS:-

Express.js is a web application framework for Node.js that simplifies the process of building robust and scalable server-side applications. It provides various features for routing, middleware, and interacting with databases. Key features of Express.js include:

- **Routing-:** Express allows you to define routes for handling different HTTP requests, such as GET, POST, PUT, and DELETE.
- **Middleware:-** Middleware functions can be used to process requests before they reach the route handlers, enabling tasks like authentication, data validation, and logging.
- **Modularity:-** Express applications can be modularized using routers, allowing developers to organize code into manageable modules.
- **Integration:-** Express easily integrates with various template engines and NoSQL databases like MongoDB

3.4.3. React:-

React is a JavaScript library for building user interfaces. It's known for its component-based architecture and the ability to create dynamic, reusable UI components. React follows a "Virtual DOM" approach, which improves performance by minimizing direct manipulation of the actual DOM. Key features of React include:

- **Component-Based:-** React applications are built using reusable components, making it easier to manage and maintain complex UI structures.
- **Virtual DOM:-** React's Virtual DOM optimizes UI updates by efficiently determining and applying the minimum changes required to reflect state changes.
- **Declarative Syntax:-** React uses a declarative syntax, allowing developers to describe the desired UI state and letting React handle the underlying updates.
- **React Router:-** A library for handling client-side routing, allowing applications to have multiple views without full-page refreshes

3.4.4. Node.js:-

Node.js is a runtime environment that allows JavaScript to be executed on the server side. It provides a non-blocking, event-driven architecture that is particularly well-suited for building scalable and real-time applications. Key features of Node.js include:

- **Asynchronous I/O:-** Node.js uses non-blocking I/O operations, allowing it to handle a large number of concurrent connections efficiently.
- **Event Loop:-** Node.js uses an event loop to handle events asynchronously, improving responsiveness and scalability.
- **NPM (Node Package Manager):-** NPM provides a vast ecosystem of open-source libraries and packages that can be easily integrated into Node.js applications.
- **Microservices:-** Node.js is well-suited for building microservices architectures due to its lightweight nature and efficient handling of I/O operations. In summary, the MERN stack combines the power of MongoDB for data storage, Express.js for server-side development, React for building dynamic user interfaces, and Node.js for server- side runtime. This stack provides a seamless, end-to-end solution for creating modern, full- stack web applications with features like real-time updates, scalability, and flexibility.

- IDE:

- VS Code - OS used for testing:
- MacOS
- Ubuntu
- Windows

3.5 Future Scope –

The future scope section of a project report outlines potential enhancements, expansions, and improvements that can be made to the AI ChatGPT web app in the future. Below are some points you might consider for the future scope of your project report:

1. **Integration of Advanced AI Models-:**
   Explore the integration of newer versions or variants of AI language models, such as GPT-4 or models developed by other organizations, to improve the natural language understanding and generation capabilities.
2. **Multilingual Support-:**
   Enhance the language capabilities to support multiple languages, allowing a more diverse user base to interact with the chatbot in their preferred language.
3. **User Authentication and Personalization-:**

Implement user authentication mechanisms to provide a personalized experience. This can include features such as user profiles, preferences, and a history of interactions.

4. **Voice and Multimedia Interaction-:**
   Extend the chatbot's capabilities to handle voice inputs and outputs, as well as multimedia content. This could involve integrating with speech recognition and synthesis systems.
5. **Machine Learning for Improved Responses-:**
   Implement machine learning algorithms to continuously analyze and improve the chatbot's responses based on user interactions and feedback. This could involve a feedback loop where the model learns from its mistakes.
6. **Real-time Collaboration-:**
   Explore features for real-time collaboration where multiple users can interact with the chatbot simultaneously, enabling collaborative decision- making or brainstorming.
7. **Integration with External Systems-:**
   Allow the chatbot to interact with external systems and APIs, enabling it to provide information or perform actions beyond its initial capabilities. For example, integrate with weather APIs, news feeds, or task management systems.
8. **Enhanced Context Handling-:**
   Improve the chatbot's ability to maintain context over extended conversations, making interactions more natural and meaningful over time.
9. **Adaptive Learning-:**

   Implement mechanisms for the chatbot to adapt and learn from long-term user interactions, providing a more personalized and context-aware experience.

10. **Accessibility Features-:**

    Implement accessibility features to ensure that the chatbot is usable by individuals with disabilities. This includes support for screen readers, voice commands, and other accessibility tools.

11. **Security and Privacy Enhancements-:**

    Strengthen security measures and consider privacy-enhancing features to ensure user data is handled securely and with respect to privacy regulations.

CHAPTER 4

WORK FLOW

4.1.MVC MODEL

The Model-View-Controller (MVC) architecture is a design pattern commonly used in software development to organize and structure code in a way that separates concerns and improves maintainability. When building a chat application with GPT AI using the MERN stack, you can apply the MVC pattern to efficiently manage different aspects of the application. Let's break down how the MVC pattern can be applied in this context:

1. **Model:-**
   The model represents the data and business logic of the application. In the context of a chat application with GPT AI, the model encompasses the data structures and interactions that handle user data, messages, and AI responses.
2. **Database (MongoDB):**
   The MongoDB database serves as the model, storing user profiles, chat messages, and related data. MongoDB collections correspond to different types of data, such as users, messages, and contacts.
3. **View:-**
   The view is responsible for rendering the user interface and presenting data to the user. In the case of a chat application with GPT AI, the view involves displaying messages, user profiles, and AI-generated responses.
4. **React Components:-**
   React components make up the view layer. Components render the UI elements, manage user interactions, and display messages and user information. The components receive data from the model and update the UI accordingly

5. **Controller:-**

   The controller handles user interactions, processes requests, and manages the flow of data between the model and the view. In the context of a chat application with GPT AI, the controller coordinates user actions, AI interactions, and data updates.

6.Express.js Routes and Controllers:-

Express.js handles the controller responsibilities. Routes are defined to handle various HTTP requests, such as sending messages and receiving AI-generated responses. Controllers process the requests, interact with the model (MongoDB), and send appropriate data to the view (React components).

4.2 Project Lifecycle:

The waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach that was used for software development.

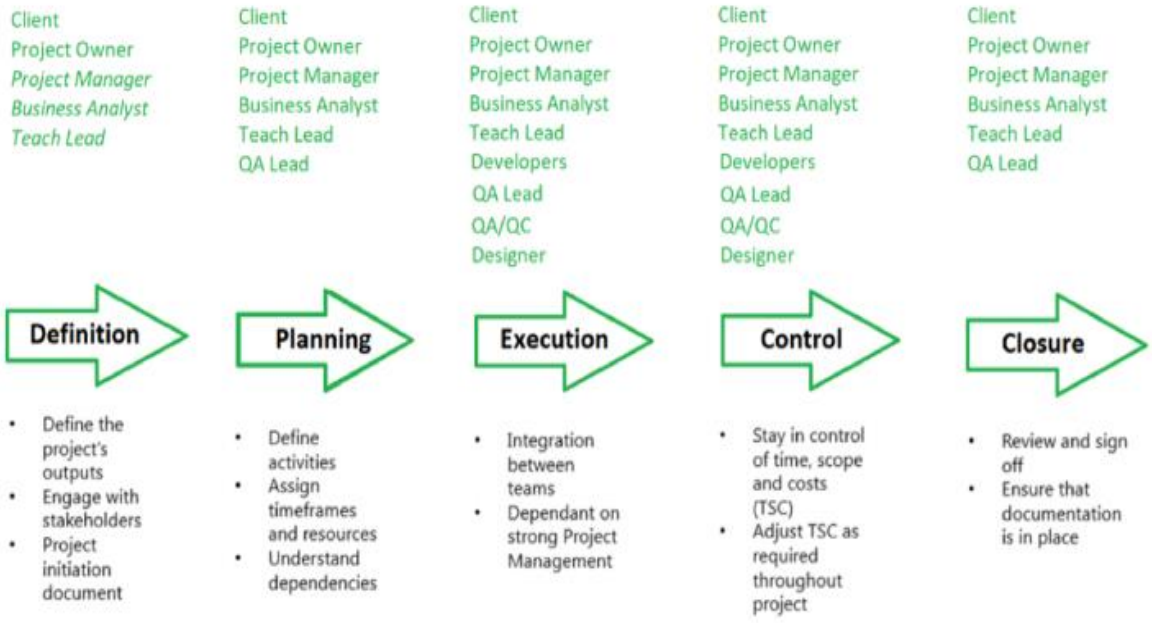

Fig.No:-4.2.1 Project lifecycle

4.3.Data Flow Diagram

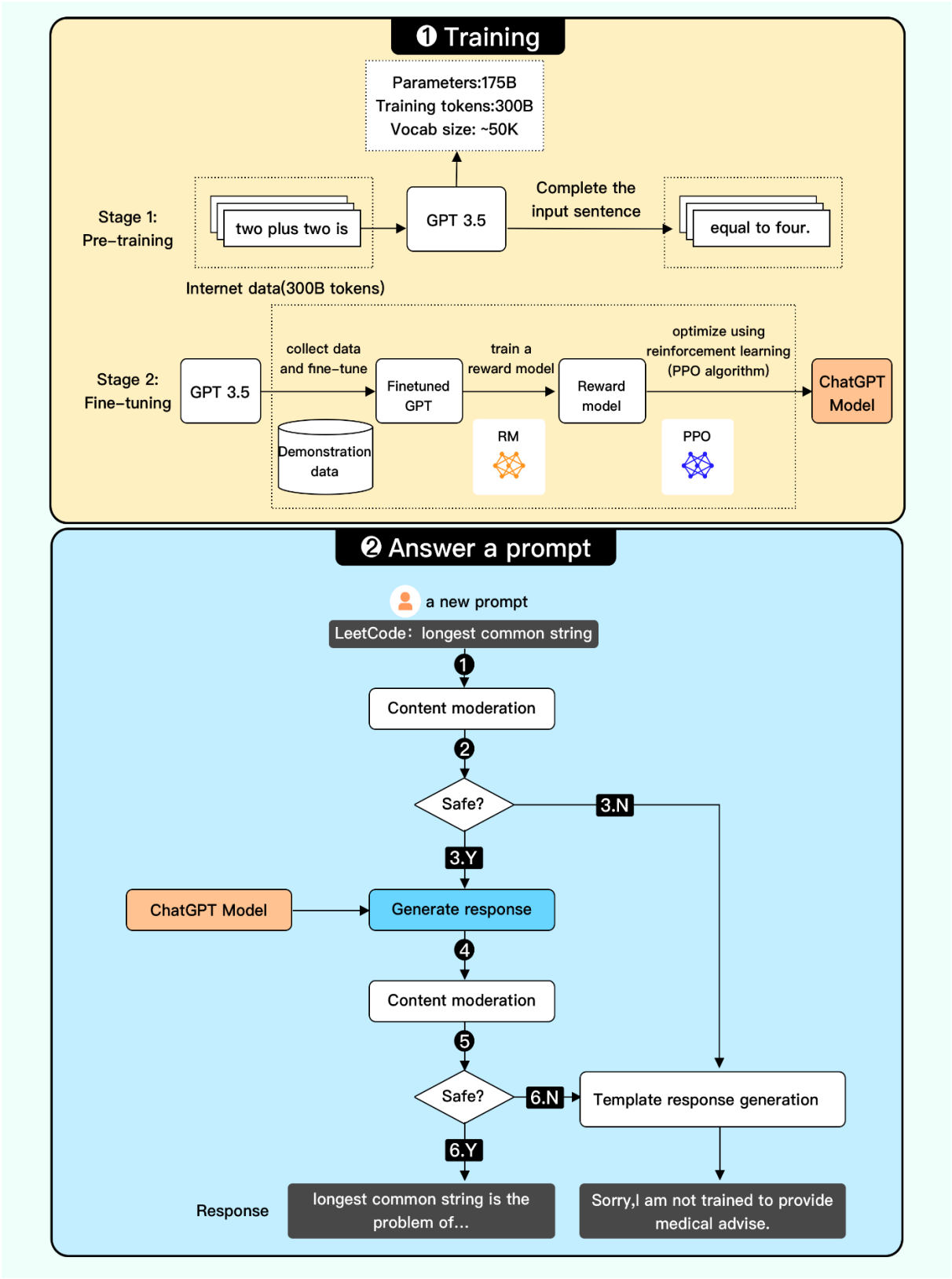

Fig.No:- 4.3.1 Data flow diagram

4.4.Use Case Diagram

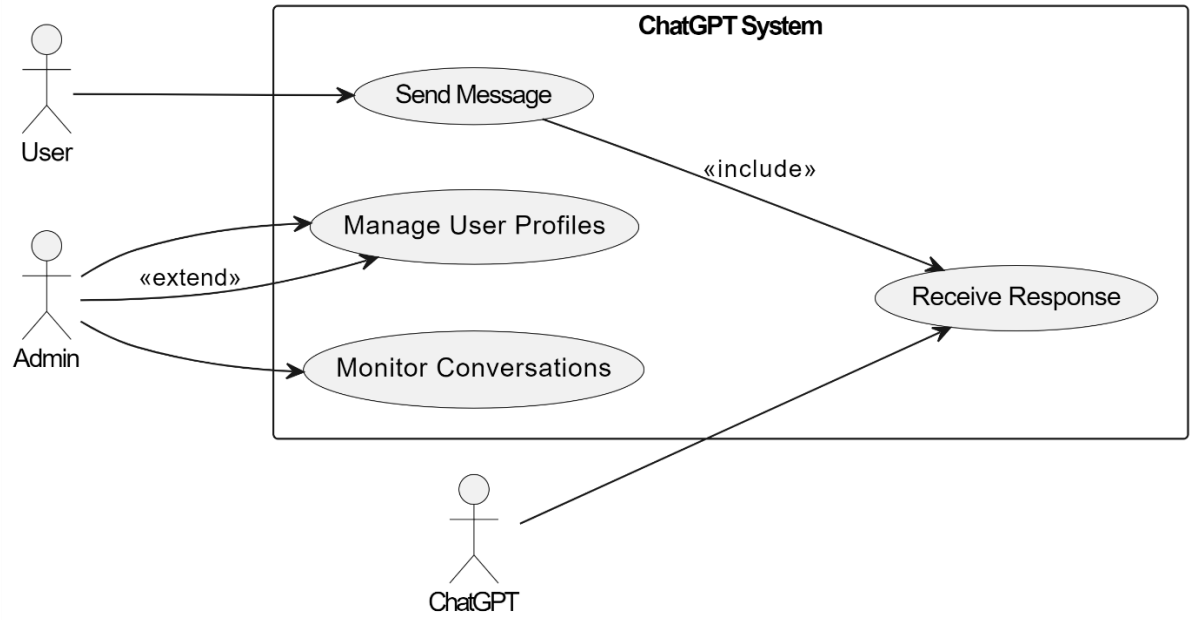

Fig.No:-4.4.1 use case diagram

4.5.ER-Diagram

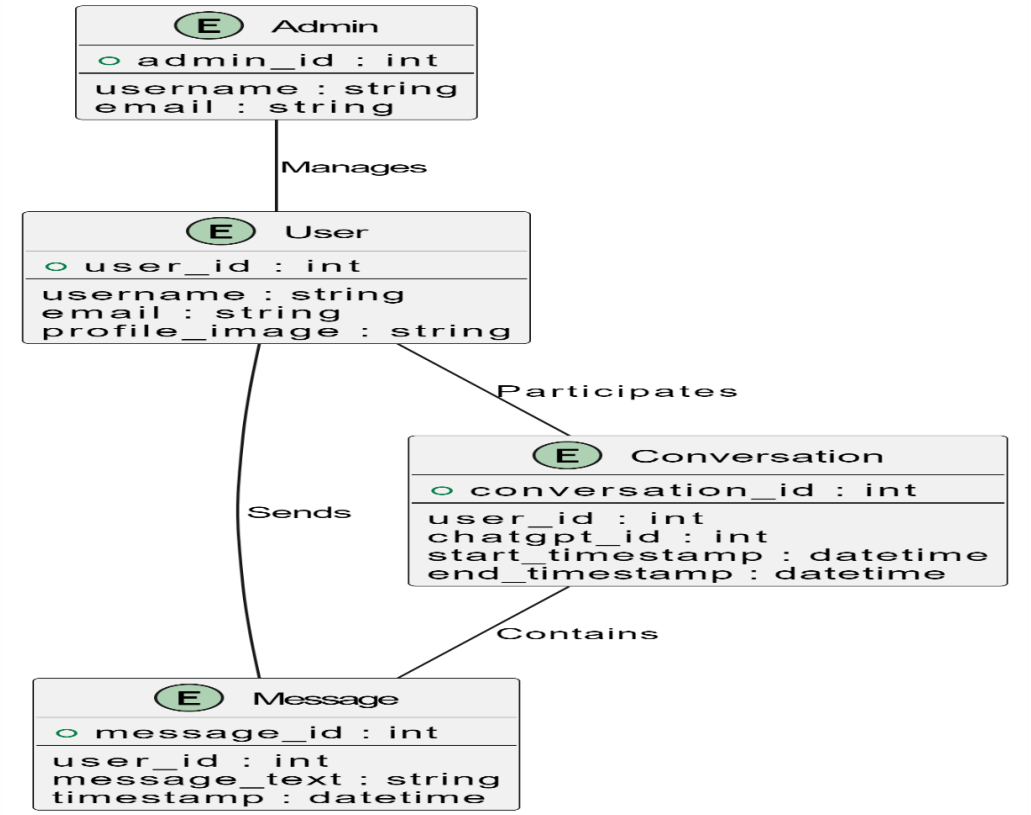

Fig.No:- 4.5.1 ER Diagram

CHAPTER 5
REQUIREMENT ANALYSIS

5.1.Hardware:

- Processor -Core i3, i5
- Hard Disk -512 GB
- Memory -4 GB RAM

5.2.Software:

- Windows 7 or higher
- MERN Stack
- MERN framework
- MONGO database
- OPEN APIs

5.3. Requirements:

1. Node.js and npm:
- Ensure Node.js is installed on your development machine.
- npm (Node Package Manager) is required for managing dependencies.

2. MongoDB:
- Install MongoDB to store user data, chat history, and other relevant information.
- Use Mongoose (a MongoDB object modeling tool for Node.js) to interact with MongoDB from Node.js.

3. Express.js:
- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Use Express to implement middleware for tasks like request parsing and authentication.

4. React.js:
- Develop the front-end chat interface using React.js.
- Utilize state management (e.g., Redux or React Context) to manage chat state and user interactions.

5. Socket.io (Optional but recommended for real-timecommunication):

- Use Socket.io to enable real-time, bidirectional communication between clients (users) and the server (chatbot).

5.3.1 Steps to Develop the Chatbot:

1. Initialize the Project:
- Set up a new project directory and initialize it with npm (`npm init`) to create a `package.json` file.

2. Backend Development (Node.js with Express):
- Install Express.js (`npm install express`) and create an Express server.
- Define routes and APIs to handle chatbot interactions (sending/receiving messages, managing users, etc.).
- Set up MongoDB connection using Mongoose (`npm install mongoose`) to store chat data and user information.

3. Implement Chatbot Logic:
- Define the chatbot's logic to process and respond to user messages.
- Use natural language processing (NLP) libraries like Dialogflow, Wit.ai, or Rasa for understanding user intents and extracting relevant information.

4. Frontend Development (React.js):
- Create a React.js application (`npx create-react-app client`) for the chat interface.
- Design and implement the chat UI components (message display, input box, etc.) using React components.
- Use Axios (`npm install axios`) or Fetch API to communicate with the backend server via RESTful APIs.

5. Real-time Communication (Socket.io):
- If real-time updates are required, integrate Socket.io (`npm install socket.io`) into both the client-side (React) and server-side (Express) applications.
- Implement Socket.io event handlers to handle message broadcasting and reception in real-time.

6. Authentication and Authorization:
- Implement user authentication and authorization mechanisms (e.g., JWT tokens, OAuth) to secure the chatbot application and manage user sessions.

7. Deployment:
- Deploy the MongoDB database to a cloud provider (e.g., MongoDB Atlas).
- Deploy the backend (Node.js/Express) and frontend (React.js) applications to a hosting service like Heroku, AWS, or Vercel.

5.3.2 Additional Considerations:

- **Error Handling:** Implement error handling mechanisms on both the client and server sides to gracefully handle exceptions.

- **Testing:** Write unit tests and integration tests to ensure the stability and functionality of the chatbot application.

- **Scalability:** Design the application with scalability in mind, considering potential growth in user base and message volume.

- **User Experience:** Focus on providing a smooth and intuitive user experience, including features like typing indicators, message timestamps, and error feedback.

By following these requirements and steps, you can successfully develop a BotStack using the MERN stack, integrating backend services with MongoDB and Express.js, and building a responsive chat interface using React.js.

CHAPTER 6:

OUTPUT AND SCREENSHOTS

6.1.INPUT

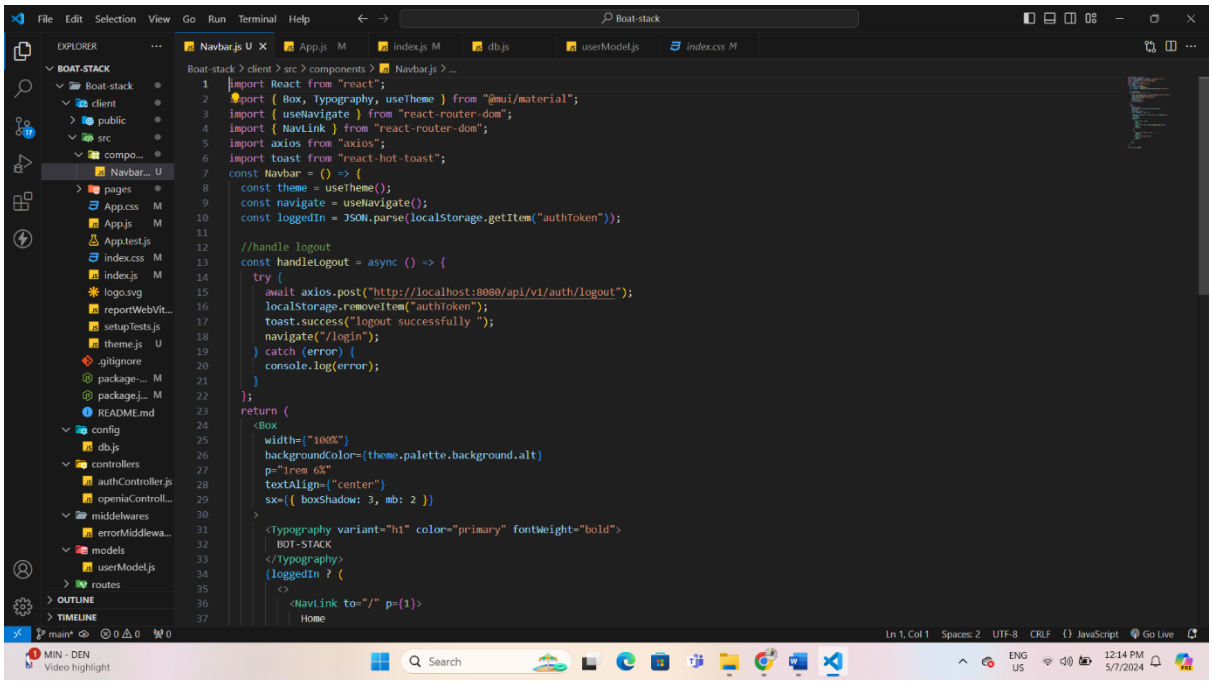

Fig.No: 6.1.1 Navbar

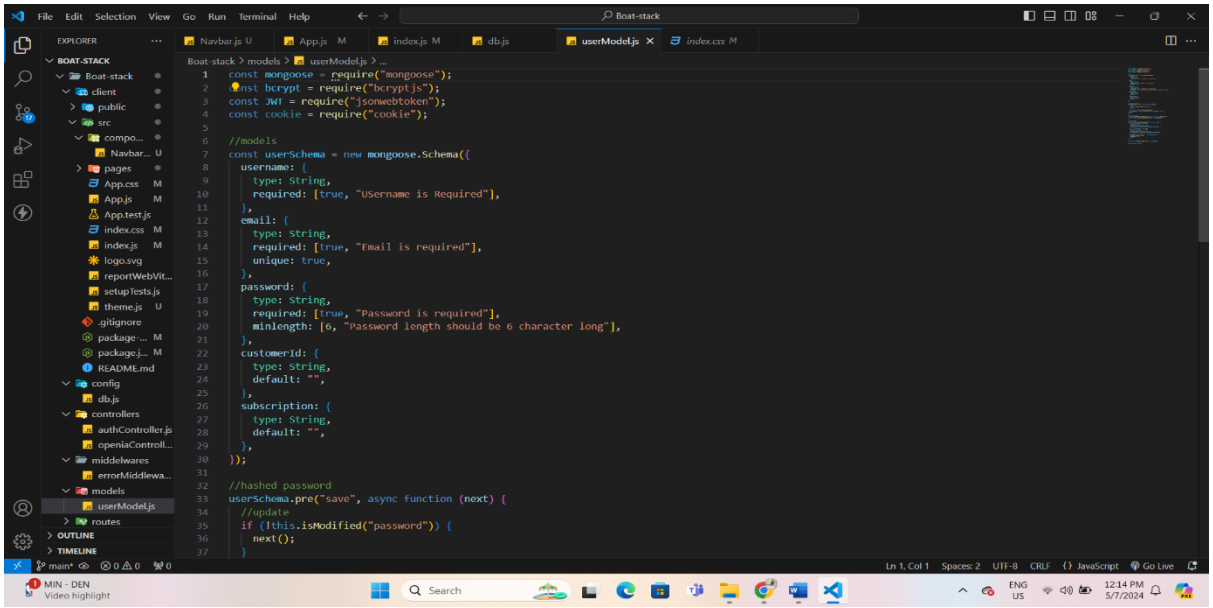

Fig.No: 6.1.2 UserModules

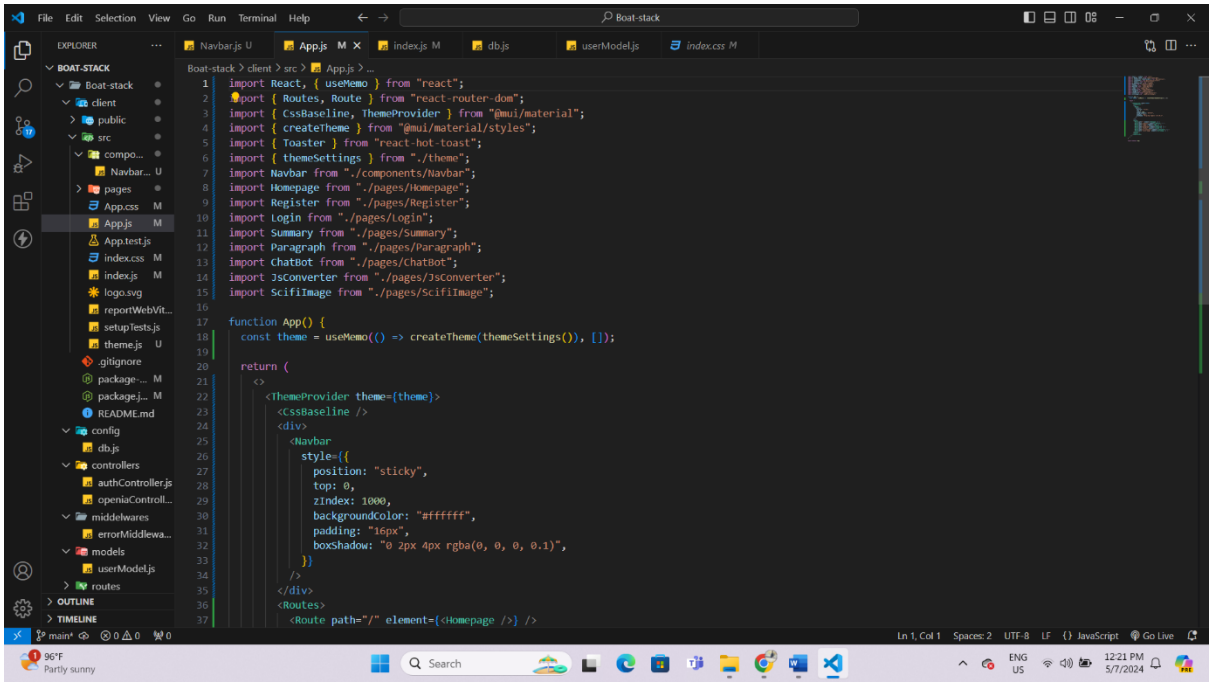

Fig.No: 6.1.3 App.js

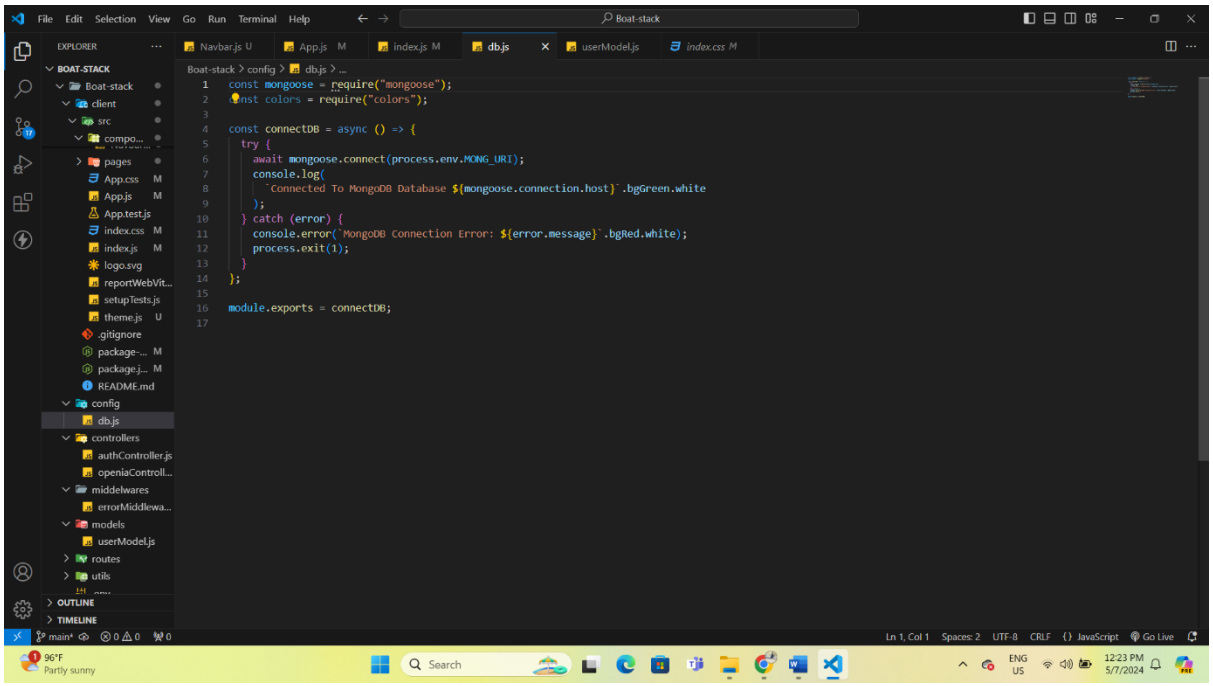

Fig.No: 6.1.4 db.js

6.2 OUTPUT

6.2.1 HOME PAGE

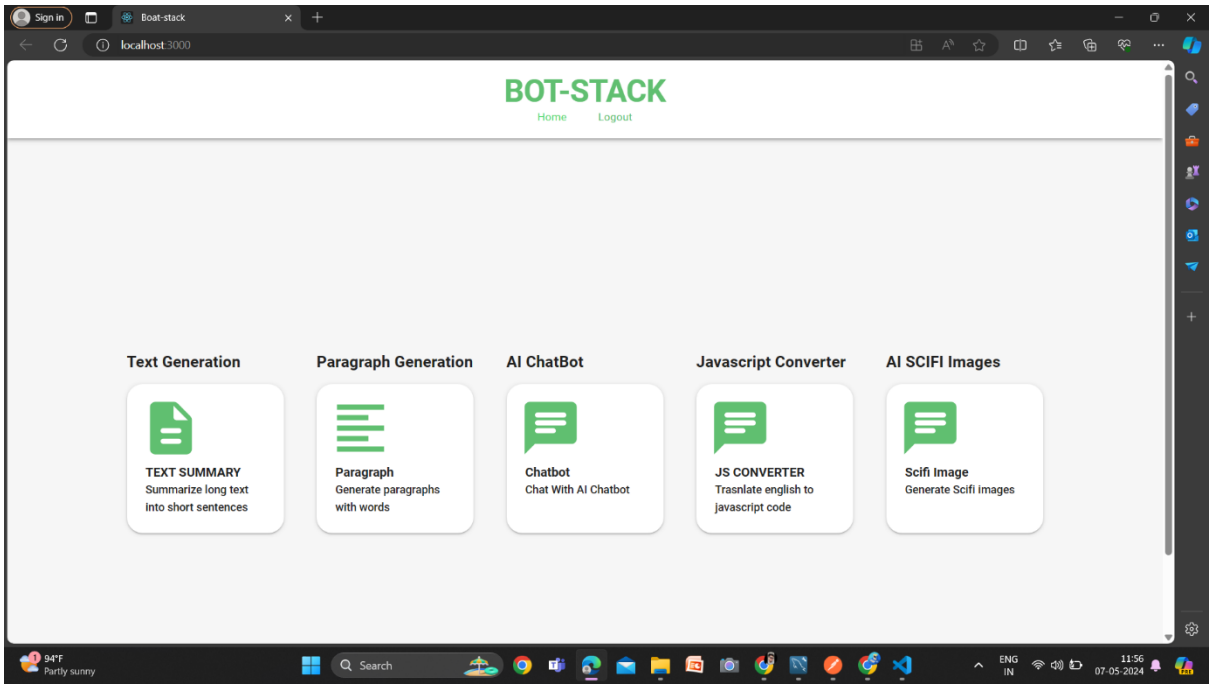

Fig.No: 6.2.1 Home Page

6.2.2 SIGN UP

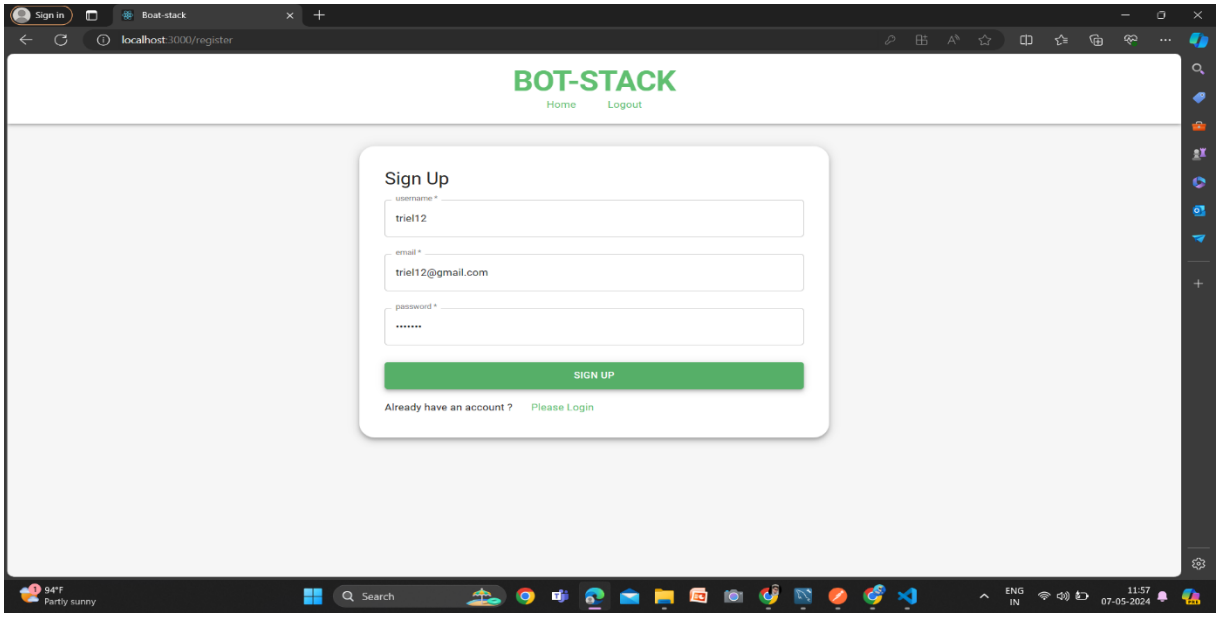

Fig.No: 6.2.2 Sign up

6.2.3 SIGN IN

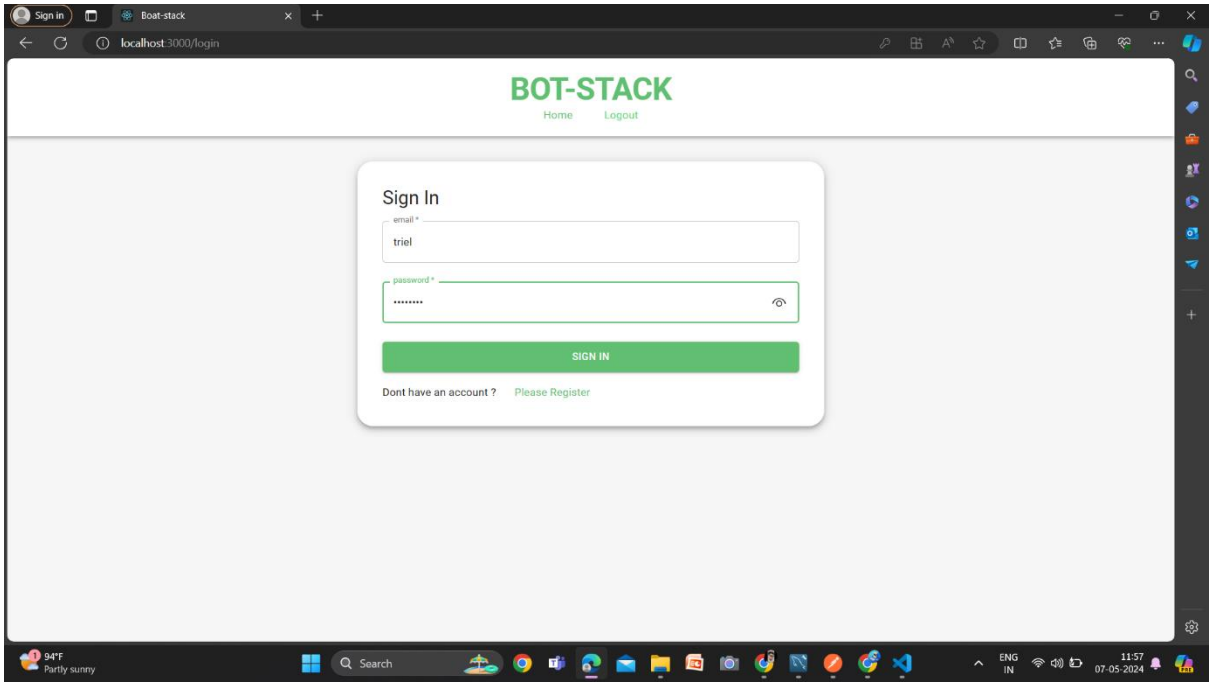

Fig.No: 6.2.3 Sign in

6.2.4 SUMMARIZE TEXT

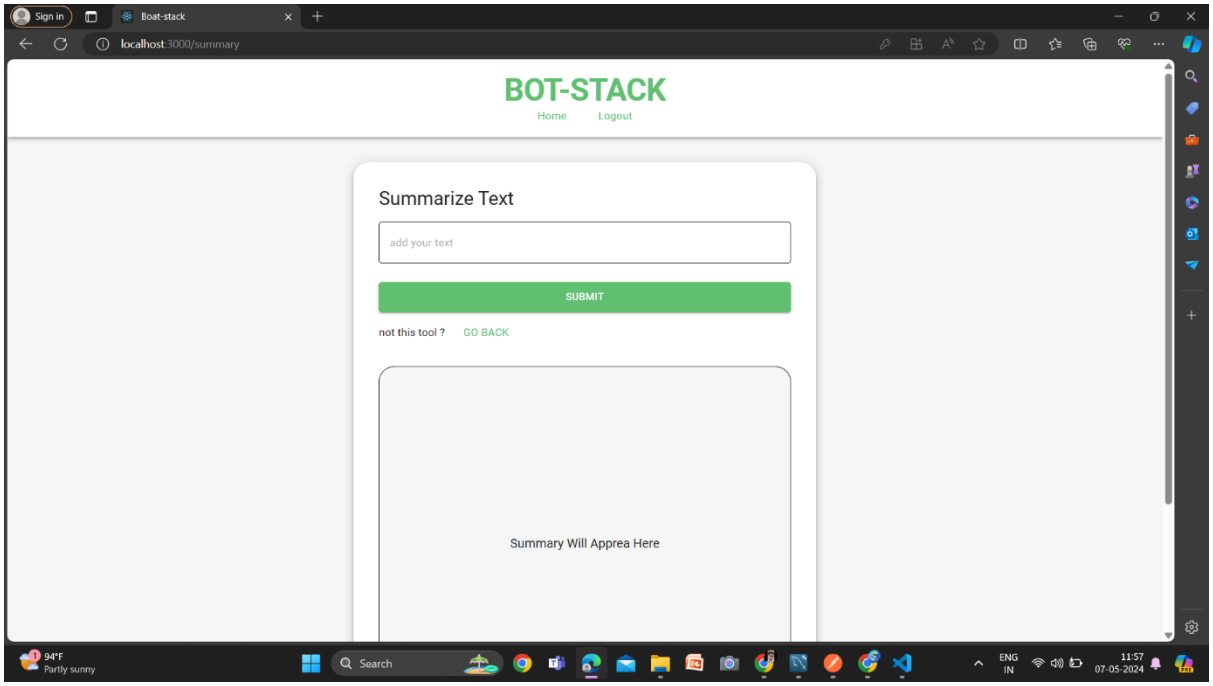

Fig.No: 6.2.4 Summarize Text

6.2.5 GENERATE PARAGRAPH

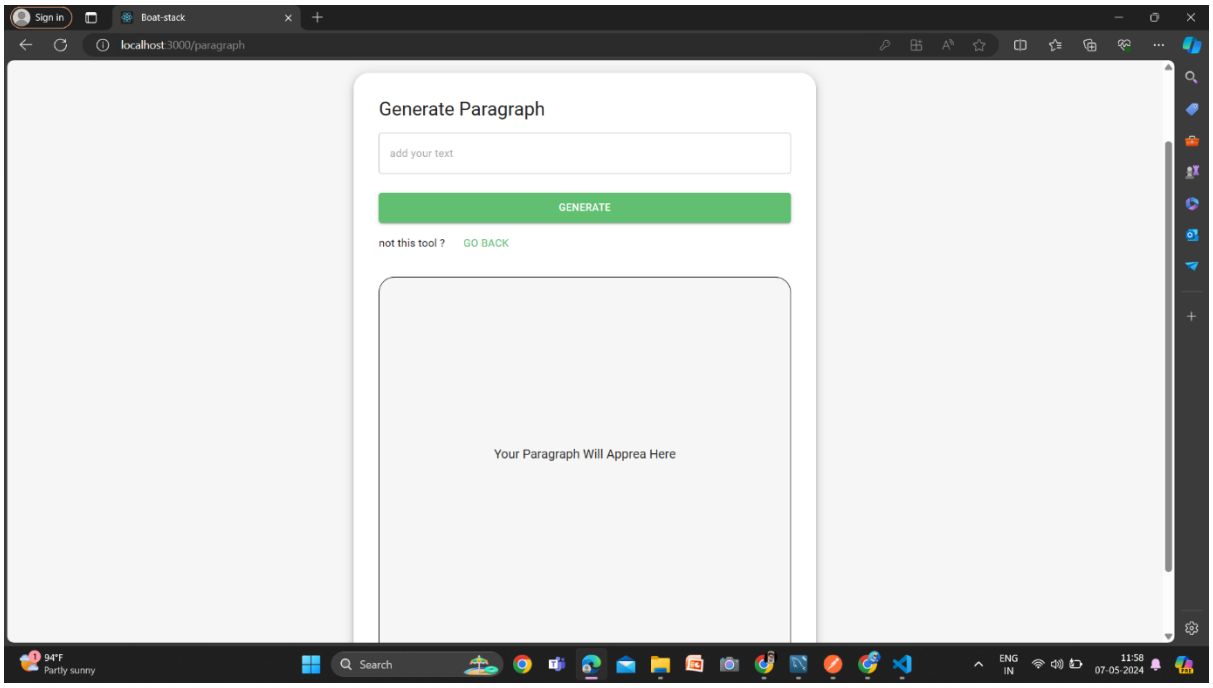

Fig.No. 6.2.5 Generate Paragraph

6.2.6 ASK WITH CHATBOT

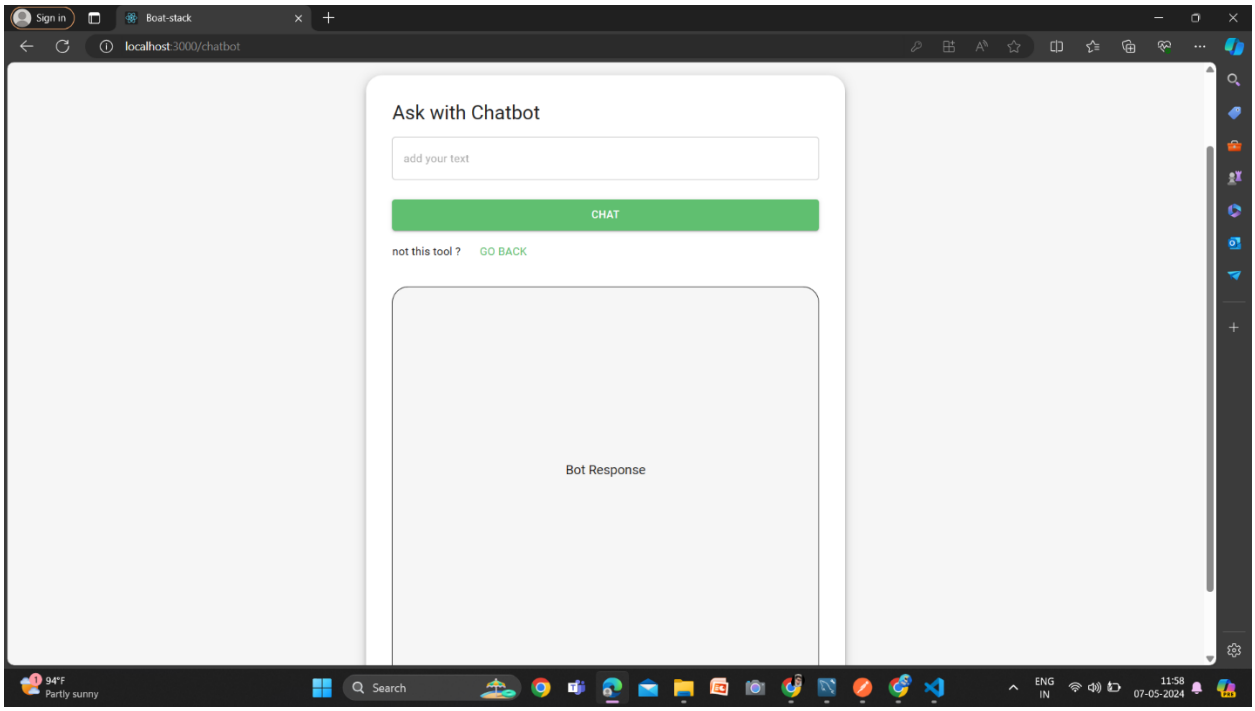

Fig.No: 6.2.6 Ask with Chatbot

6.2.7 SCIFI IMAGE

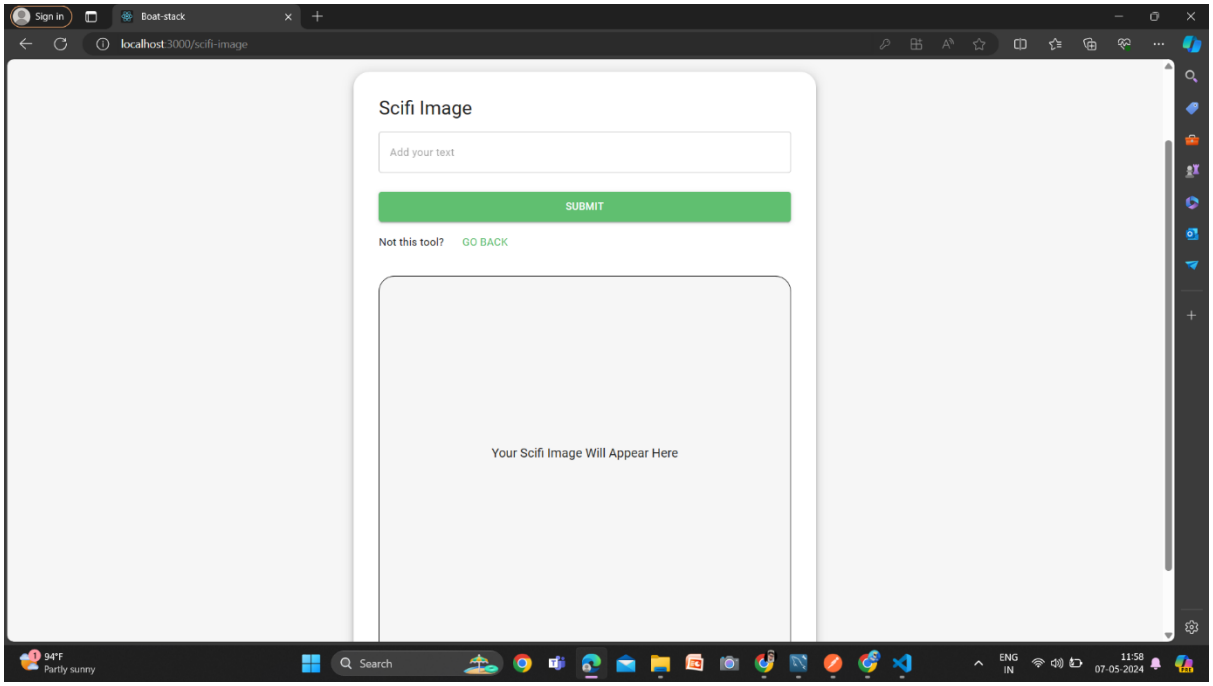

Fig.No: 6.2.7 Scific image

CONCLUSION

In conclusion, the AI BoatStack web app project has been a significant endeavor that aimed to harness the power of advanced natural language processing (NLP) models in creating an interactive and intelligent chatbot. Through the course of the project, several key observations and achievements have come to light. Boatstack no doubt is creating a scare in the public eye, nevertheless it also will highlight the individuals with real talent and shall help in voicing out who as an individual is at his or her best since many of the answers assisted by Boatstack is going to be identical than the person, who utilises his or her content, then the real part of the content the key for education will bring out the realities in education. The AI writing tools might be too homogenous, and can also become evident whether the article content is original or stolen from the Boatstack, or through any other technology or by a human. Boatstack also will bring informational content that could help the writers with as many assignments where authors will be released from the confinements of SEOs and would make the content getting to uniqueness as well as insightful so that newer floodgates could open. It will also clarify what humans need or what the needs of humans are, where content will become more human for not only reading but for producing

Achievements:

1. **Successful Integration of BoatStack:**
   The successful integration of the BoatStack model into the web application has proven to be a milestone. Users can initiate conversations, exchange messages, and experience a natural and dynamic interaction with the chatbot.

Functionality:

The chatbot successfully fulfills its intended purpose of [describe primary function.

- **User Engagement:** Positive feedback and engagement from users demonstrate the effectiveness of the chatbot.
- Technical Innovation: Leveraging cutting-edge technologies and methodologies to create an intelligent and responsive system.
- **Key Learnings**

- **User-Centric Design:** Understanding user needs and preferences is essential for designing a successful chatbot.
- **Continuous Improvement:** Iterative development and user feedback are crucial for enhancing chatbot performance.
- **Technical Challenges**: Overcoming challenges in natural language understanding, context management, and scalability.

Future Directions

Looking ahead, several opportunities exist for further enhancing our chatbot:

- **Advanced Functionality:** Introducing new features such as [list potential features].
- **Integration**: Expanding integrations with additional platforms and services.
- **Personalization:** Implementing personalized user experiences based on machine learning and user data.
- **Analytics and Optimization:** Utilizing data analytics to gain insights and optimize chatbot performance.

Final Reflection

The journey of developing this chatbot has been rewarding and insightful. It has not only expanded our technical capabilities but also deepened our understanding of user behavior and AI applications in real-world scenarios. As we continue to refine and evolve our chatbot, we remain committed to delivering innovative solutions that meet and exceed user expectations.

REFERENCES

1. Shidiq, M. (2023, March). The use of artificial intelligence-based chat-gpt and its challenges for the world of education; from the viewpoint of the development of creative writing skills. In Proceeding of International Conference on Education, Society, and Humanity (Vol. 1, No. 1, pp. 360-364).
2. Yan, D. (2023). Impact of ChatGPT on learners in an L2 writing practicum: An exploratory investigation. Education and Information Technologies, 1-25.
3. Mhlanga, D. (2023). Open AI in education, the responsible and ethical use of ChatGPT towards lifelong learning. Education, the Responsible and Ethical Use of ChatGPT Towards Lifelong Learning (February 11, 2023).
4. [4] Malinka, K., Perešíni, M., Firc, A., Hujňák, O., & Januš, F. (2023). On the Educational Impact of ChatGPT: Is Artificial Intelligence Ready to Obtain a University Degree? arXiv preprint arXiv:2303.11146
5. M. U. Haque, I. Dharmadasa et al., ""i think this is the most disruptive technology": Exploring sentiments of chatgpt early adopters using twitter data," 2022
6. C. A. Gao, F. M. Howard et al., "Comparing scientific abstracts generated by chatgpt to original abstracts using an artificial intelligence output detector, plagiarism detector, and blinded human reviewers," 2022.
7. ee,"
8. A case study of learning attitude change according to programming learning experience," Journal of the Korea Convergence Society, vol. 12, no. 9, pp. 93-98, 2021.
9. S. H. Kim, et al., "Applications of Educational Programming Languages in K-12 Information curriculum," The Journal of Korean association of computer education, vol. 12, no. 2, pp. 23-31, 2009.
10. Roose, Kevin (2022). "The Brilliance and Weirdness of ChatGPT". The New York Times. Archived from the original on January 18, 2023. Retrieved December 26, 2022. Like those tools, ChatGPT — which stands for "generative pre-trained transformer" — landed with a splash.
11. Quinn, Joanne (2020). Dive into deep learning: tools for engagement. Thousand Oaks, California. p. 551. ISBN 9781544361376. Archived from the original on January 10, 2023. Retrieved January 10, 2023.
12. aranasi, Lakshmi (2023). "ChatGPT creator OpenAI is in talks to sell shares in a tender offer that would double the startup's valuation to $29 billion". Insider. Archived from the original on January 18, 2023. Retrieved January 18, 2023.
13. CHATBOT BASED ON AI Information technology professors Nikita Hatwarl, Ashwini Patil 2, and Diksha Gondane 3123 are from Nagpur/RTMNU in India. Volume 3, Issue 2 (March-April 2016),

International Journal of Emerging Trends in Engineering and Basic Sciences (UEEBS), ISSN(Online)2349-6967)
14. Yuhua Li, David McLean, Zuhair A. Bandar, James D. O'Shea, Keeley Crockett, "Sentence Similarity Based on Semantic Nets and Corpus Statistics", IEEE Transactions on Knowledge and Data Engineering, Volume 18 - No. 8, August 2006.
15. Emanuela Haller, Traian Rebedea, "Designing a Chat-bot that Simulates an Historical Figure", IEEE Conference Publications, July 2013.
16. Prof. K. Bala, Mukesh Kumar, Sayali Hulawale, Sahil Pandita, "Chat-Bot For College Management System Using A.I", International Research Journal of Engineering and Technology, Volume 4, Issue 11, Nov 2017.
17. Mohammadi, H., Asadi, H., & Kazemi, A. (2019). An intelligent chatbot for education in autism. Journal of Educational Computing Research, 57(8), 1944-1958.IEEE Transactions on Industrial Informatics, 14(9):4127–4136, 2018.
18. Kopp, S., Gesellensetter, L., Krämer, N. C., & Wachsmuth, I. (2017). A conversational agent as museum guide—design and evaluation of a real-world application. International Journal of Human-Computer Studies, 105, 77-85.
19. Koczkodaj, W. W., & Alizadeh, M. (2020). Chatbots and Artificial Intelligence Applications in Educational Settings: A Review of Literature. In Advances in Artificial Intelligence (pp. 295-304). Springer.
20. Singh, R.; Paste, M.; Shinde, N.; Patel, H.; Mishra, N. Chatbot using TensorFlow for small Businesses. In Proceedings of the 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 20–21 April 2018; pp. 1614–1619.
21. Ikumoro, A.O.; Jawad, M.S. Intention to Use Intelligent Conversational Agents in e-Commerce among Malaysian SMEs: An Integrated Conceptual Framework Based on Tri-theories including Unified Theory of Acceptance, Use of Technology (UTAUT), and T-O-E. Int. J. Acad. Res. Bus. Soc. Sci. 2019, 9, 205–235.
22. Cui, L.; Huang, S.; Wei, F.; Tan, C.; Duan, C.; Zhou, M. SuperAgent: A Customer Service Chatbot for E-commerce Websites. In Proceedings of the ACL 2017, System Demonstrations, Vancouver, QC, Canada, 30 July–4 August 2017; Association for Computational Linguistics: Vancouver, QC, Canada, 2017; pp. 97–102

Source code

Client

1.App.js

```
import React, { useMemo } from "react";
import { Routes, Route } from "react-router-dom";
import { CssBaseline, ThemeProvider } from "@mui/material";
import { createTheme } from "@mui/material/styles";
import { Toaster } from "react-hot-toast";
import { themeSettings } from "./theme";
import Navbar from "./components/Navbar";
import Homepage from "./pages/Homepage";
import Register from "./pages/Register";
import Login from "./pages/Login";
import Summary from "./pages/Summary";
import Paragraph from "./pages/Paragraph";
import ChatBot from "./pages/ChatBot";
import JsConverter from "./pages/JsConverter";
import ScifiImage from "./pages/ScifiImage";

function App() {
  const theme = useMemo(() => createTheme(themeSettings()), []);

  return (
    <>
      <ThemeProvider theme={theme}>
```

```
<CssBaseline />
<div>
  <Navbar
    style={{
      position: "sticky",
      top: 0,
      zIndex: 1000,
      backgroundColor: "#ffffff",
      padding: "16px",
      boxShadow: "0 2px 4px rgba(0, 0, 0, 0.1)",
    }}
  />
</div>
<Routes>
  <Route path="/" element={<Homepage />} />
  <Route path="/register" element={<Register />} />
  <Route path="/login" element={<Login />} />
  <Route path="/summary" element={<Summary />} />
  <Route path="/paragraph" element={<Paragraph />} />
  <Route path="/chatbot" element={<ChatBot />} />
  <Route path="/js-converter" element={<JsConverter />} />
  <Route path="/scifi-image" element={<ScifiImage />} />
</Routes>
<Toaster />
```

```
      </ThemeProvider>
    </>
  );
}export default App;
```

2.NavBar.js

```
import React from "react";
import { Box, Typography, useTheme } from "@mui/material";
import { useNavigate } from "react-router-dom";
import { NavLink } from "react-router-dom";
import axios from "axios";
import toast from "react-hot-toast";
const Navbar = () => {
  const theme = useTheme();
  const navigate = useNavigate();
  const loggedIn = JSON.parse(localStorage.getItem("authToken"));

  //handle logout
  const handleLogout = async () => {
    try {
      await axios.post("http://localhost:8080/api/v1/auth/logout");
      localStorage.removeItem("authToken");
      toast.success("logout successfully ");
      navigate("/login");
    } catch (error) {
      console.log(error);
```

```
  }
};
return (
  <Box
    width={"100%"}
    backgroundColor={theme.palette.background.alt}
    p="1rem 6%"
    textAlign={"center"}
    sx={{ boxShadow: 3, mb: 2 }}
  >
    <Typography variant="h1" color="primary" fontWeight="bold">
      BOT-STACK
    </Typography>
    {loggedIn ? (
      <>
        <NavLink to="/" p={1}>
          Home
        </NavLink>
        <NavLink to="/login" onClick={handleLogout} p={1}>
          Logout
        </NavLink>
      </>
    ) : (
      <>
```

```
          <NavLink to="/register" p={1}>
            Sign Up
          </NavLink>
          <NavLink to="/login" p={1}>
            Sign In
          </NavLink>
        </>
      )}
    </Box>  );};export default Navbar;
```

3.Chatbot.js

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import toast from "react-hot-toast";
import axios from "axios";
import {
  Box,
  Typography,
  useTheme,
  useMediaQuery,
  TextField,
  Button,
  Alert,
  Collapse,
  Card,
} from "@mui/material";
```

```
const ChatBot = () => {
  const theme = useTheme();
  const navigate = useNavigate();
  //media
  const isNotMobile = useMediaQuery("(min-width: 1000px)");
  // states
  const [text, settext] = useState("");
  const [response, setResponse] = useState("");
  const [error, setError] = useState("");

  //register ctrl
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const { data } = await axios.post("/api/v1/openai/chatbot", { text });
      console.log(data);
      setResponse(data);
    } catch (err) {
      console.log(error);
      if (err.response.data.error) {
        setError(err.response.data.error);
      } else if (err.message) {
        setError(err.message);
```

```
    }
    setTimeout(() => {
      setError("");
    }, 5000);
  }
};
return (
  <Box
    width={isNotMobile ? "40%" : "80%"}
    p={"2rem"}
    m={"2rem auto"}
    borderRadius={5}
    sx={{ boxShadow: 5 }}
    backgroundColor={theme.palette.background.alt}
  >
    <Collapse in={error}>
      <Alert severity="error" sx={{ mb: 2 }}>
        {error}
      </Alert>
    </Collapse>
    <form onSubmit={handleSubmit}>
      <Typography variant="h3">Ask with Chatbot</Ty pography>

      <TextField
```

```
  placeholder="add your text"
  type="text"
  multiline={true}
  required
  margin="normal"
  fullWidth
  value={text}
  onChange={(e) => {
    settext(e.target.value);
  }}
/>

<Button
  type="submit"
  fullWidth
  variant="contained"
  size="large"
  sx={{ color: "white", mt: 2 }}
>
  Chat
</Button>
<Typography mt={2}>
  not this tool ? <Link to="/">GO BACK</Link>
</Typography>
```

```
</form>

{response ? (
  <Card
    sx={{
      mt: 4,
      border: 1,
      boxShadow: 0,
      height: "500px",
      borderRadius: 5,
      borderColor: "natural.medium",
      bgcolor: "background.default",
    }}
  >
    <Typography p={2}>{response}</Typography>
  </Card>
) : (
  <Card
    sx={{
      mt: 4,
      border: 1,
      boxShadow: 0,
      height: "500px",
      borderRadius: 5,
```

```
            borderColor: "natural.medium",
            bgcolor: "background.default",
          }}
        >
          <Typography
            variant="h5"
            color="natural.main"
            sx={{
              textAlign: "center",
              verticalAlign: "middel",
              lineHeight: "450px",
            }}
          >
            Bot Response
          </Typography>
        </Card>
      )}
    </Box>
  );
};

export default ChatBot;
```

4.Home page

```
import React from "react";
```

```
import { Box, Typography, Card, Stack } from "@mui/material";
import { useNavigate } from "react-router-dom";
import DescriptionRounded from "@mui/icons-material/DescriptionRounded";
import FormatAlignLeftOutlined from "@mui/icons-material/FormatAlignLeftOutlined";
import ChatRounded from "@mui/icons-material/ChatRounded";

const Homepage = () => {
  const navigate = useNavigate();

  return (
    <Box
      sx={{
        display: "flex",
        justifyContent: "center",
        alignItems: "center",
        minHeight: "100vh",

      }}
    >
      <Box sx={{ display: "flex", flexDirection: "row", gap:""}}>
        <Box p={2} sx={{ margin: "0 5px" }}>
          <Typography variant="h4" mb={2} fontWeight="bold">
            Text Generation
          </Typography>
```

```
<Card
  onClick={() => navigate("/summary")}
  sx={{
    boxShadow: 2,
    borderRadius: 5,
    height: 190,
    width: 200,
    "&:hover": {
      border: 2,
      boxShadow: 0,
      borderColor: "primary.dark",
      cursor: "pointer",
    },
  }}
>
  <DescriptionRounded
    sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}
  />
  <Stack p={3} pt={0}>
    <Typography fontWeight="bold" variant="h5">
      TEXT SUMMARY
    </Typography>
    <Typography variant="h6">
      Summarize long text into short sentences
```

```
      </Typography>
    </Stack>
  </Card>
</Box>
<Box p={2} sx={{ margin: "0 5px" }}>
  <Typography variant="h4" mb={2} fontWeight="bold">
    Paragraph Generation
  </Typography>
  <Card
    onClick={() => navigate("/paragraph")}
    sx={{
      boxShadow: 2,
      borderRadius: 5,
      height: 190,
      width: 200,
      "&:hover": {
        border: 2,
        boxShadow: 0,
        borderColor: "primary.dark",
        cursor: "pointer",
      },
    }}
  >
    <FormatAlignLeftOutlined
```

```
      sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}
    />
    <Stack p={3} pt={0}>
      <Typography fontWeight="bold" variant="h5">
        Paragraph
      </Typography>
      <Typography variant="h6">
        Generate paragraphs with words
      </Typography>
    </Stack>
  </Card>
</Box>
<Box p={2} sx={{ margin: "0 5px" }}>
  <Typography variant="h4" mb={2} fontWeight="bold">
    AI ChatBot
  </Typography>
  <Card
    onClick={() => navigate("/chatbot")}
    sx={{
      boxShadow: 2,
      borderRadius: 5,
      height: 190,
      width: 200,
      "&:hover": {
```

```
        border: 2,
        boxShadow: 0,
        borderColor: "primary.dark",
        cursor: "pointer",
      },
    }}
  >
    <ChatRounded
      sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}
    />
    <Stack p={3} pt={0}>
      <Typography fontWeight="bold" variant="h5">
        Chatbot
      </Typography>
      <Typography variant="h6">Chat With AI Chatbot</Typography>
    </Stack>
  </Card>
</Box>
<Box p={2} sx={{ margin: "0 5px" }}>
  <Typography variant="h4" mb={2} fontWeight="bold">
    Javascript Converter
  </Typography>
  <Card
    onClick={() => navigate("/js-converter")}
```

```
  sx={{
    boxShadow: 2,
    borderRadius: 5,
    height: 190,
    width: 200,
    "&:hover": {
      border: 2,
      boxShadow: 0,
      borderColor: "primary.dark",
      cursor: "pointer",
    },
  }}
>
  <ChatRounded
    sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}
  />
  <Stack p={3} pt={0}>
    <Typography fontWeight="bold" variant="h5">
      JS CONVERTER
    </Typography>
    <Typography variant="h6">
      Trasnlate english to javascript code
    </Typography>
  </Stack>
```

```
  </Card>
</Box>
<Box p={2} sx={{ margin: "0 5px" }}>
  <Typography variant="h4" mb={2} fontWeight="bold">
    AI SCIFI Images
  </Typography>
  <Card
    onClick={() => navigate("/scifi-image")}
    sx={{
      boxShadow: 2,
      borderRadius: 5,
      height: 190,
      width: 200,
      "&:hover": {
        border: 2,
        boxShadow: 0,
        borderColor: "primary.dark",
        cursor: "pointer",
      },
    }}
  >
    <ChatRounded
      sx={{ fontSize: 80, color: "primary.main", mt: 2, ml: 2 }}
    />
```

```
            <Stack p={3} pt={0}>
              <Typography fontWeight="bold" variant="h5">
                Scifi Image
              </Typography>
              <Typography variant="h6">Generate Scifi images</Typography>
            </Stack>
          </Card>
        </Box>
      </Box>
    </Box>
  );
};

export default Homepage;
```

5.Jsconvertor.js

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import toast from "react-hot-toast";
import axios from "axios";
import {
  Box,
  Typography,
  useTheme,
  useMediaQuery,
  TextField,
  Button,
  Alert,
  Collapse,
  Card,
} from "@mui/material";

const JsConverter = () => {
  const theme = useTheme();
  const navigate = useNavigate();
  //media
  const isNotMobile = useMediaQuery("(min-width: 1000px)");
  // states
  const [text, settext] = useState("");
```

```
const [code, setCode] = useState("");
const [error, setError] = useState("");

//register ctrl
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const { data } = await axios.post("/api/v1/openai/js-converter", {
      text,
    });
    console.log(data);
    setCode(data);
  } catch (err) {
    console.log(error);
    if (err.response.data.error) {
      setError(err.response.data.error);
    } else if (err.message) {
      setError(err.message);
    }
    setTimeout(() => {
      setError("");
    }, 5000);
  }
};
```

```
return (
  <Box
    width={isNotMobile ? "40%" : "80%"}
    p={"2rem"}
    m={"2rem auto"}
    borderRadius={5}
    sx={{ boxShadow: 5 }}
    backgroundColor={theme.palette.background.alt}
  >
    <Collapse in={error}>
      <Alert severity="error" sx={{ mb: 2 }}>
        {error}
      </Alert>
    </Collapse>
    <form onSubmit={handleSubmit}>
      <Typography variant="h3">JS Converter</Typography>

      <TextField
        placeholder="add your text"
        type="text"
        multiline={true}
        required
        margin="normal"
        fullWidth
```

```
    value={text}
    onChange={(e) => {
      settext(e.target.value);
    }}
  />

  <Button
    type="submit"
    fullWidth
    variant="contained"
    size="large"
    sx={{ color: "white", mt: 2 }}
  >
    Convert
  </Button>
  <Typography mt={2}>
    not this tool ? <Link to="/">GO BACK</Link>
  </Typography>
</form>

{code ? (
  <Card
    sx={{
      mt: 4,
```

```
        border: 1,
        boxShadow: 0,
        height: "500px",
        borderRadius: 5,
        borderColor: "natural.medium",
        bgcolor: "background.default",
        overflow: "auto",
      }}
    >
      <pre>
        <Typography p={2}>{code}</Typography>
      </pre>
    </Card>
  ) : (
    <Card
      sx={{
        mt: 4,
        border: 1,
        boxShadow: 0,
        height: "500px",
        borderRadius: 5,
        borderColor: "natural.medium",
        bgcolor: "background.default",
      }}
```

```
        >
          <Typography
            variant="h5"
            color="natural.main"
            sx={{
              textAlign: "center",
              verticalAlign: "middel",
              lineHeight: "450px",
            }}
          >
            Your Code Will Apprea Here
          </Typography>
        </Card>
      )}
    </Box>
  );
};

export default JsConverter;
```

6.Login.js

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import toast from "react-hot-toast";
```

```
import axios from "axios";
import {
  Box,
  Typography,
  useTheme,
  useMediaQuery,
  TextField,
  Button,
  Alert,
  Collapse,
} from "@mui/material";

const Login = () => {
  const theme = useTheme();
  const navigate = useNavigate();
  //media
  const isNotMobile = useMediaQuery("(min-width: 1000px)");
  // states
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  //register ctrl
  const handleSubmit = async (e) => {
```

```
    e.preventDefault();
    try {
      await axios.post("http://localhost:8080/api/v1/auth/login", { email, password
});
      toast.success("Login Successfully");
      localStorage.setItem("authToken", true);
      navigate("/");
    } catch (err) {
      console.log(error);
      if (err.response.data.error) {
        setError(err.response.data.error);
      } else if (err.message) {
        setError(err.message);
      }
      setTimeout(() => {
        setError("");
      }, 5000);
    }
  };
  return (
    <Box
      width={isNotMobile ? "40%" : "80%"}
      p={"2rem"}
      m={"2rem auto"}
      borderRadius={5}
```

```
  sx={{ boxShadow: 5 }}
  backgroundColor={theme.palette.background.alt}
>
  <Collapse in={error}>
    <Alert severity="error" sx={{ mb: 2 }}>
      {error}
    </Alert>
  </Collapse>
  <form onSubmit={handleSubmit}>
    <Typography variant="h3">Sign In</Typography>

    <TextField
      label="email"
      type="email"
      required
      margin="normal"
      fullWidth
      value={email}
      onChange={(e) => {
        setEmail(e.target.value);
      }}
    />
    <TextField
      label="password"
```

```
        type="password"
        required
        margin="normal"
        fullWidth
        value={password}
        onChange={(e) => {
          setPassword(e.target.value);
        }}
      />
      <Button
        type="submit"
        fullWidth
        variant="contained"
        size="large"
        sx={{ color: "white", mt: 2 }}
      >
        Sign In
      </Button>
      <Typography mt={2}>
        Dont have an account ? <Link to="/register">Please Register</Link>
      </Typography>
    </form>
  </Box>
);
```

```
};
export default Login;
```

7.Paragraph.js

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import toast from "react-hot-toast";
import axios from "axios";
import {
  Box,
  Typography,
  useTheme,
  useMediaQuery,
  TextField,
  Button,
  Alert,
  Collapse,
  Card,
} from "@mui/material";

const Paragraph = () => {
  const theme = useTheme();
  const navigate = useNavigate();
  //media
  const isNotMobile = useMediaQuery("(min-width: 1000px)");
```

```
// states
const [text, settext] = useState("");
const [para, setPara] = useState("");
const [error, setError] = useState("");

//register ctrl
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const { data } = await axios.post("/api/v1/openai/paragraph", { text });
    console.log(data);
    setPara(data);
  } catch (err) {
    console.log(error);
    if (err.response.data.error) {
      setError(err.response.data.error);
    } else if (err.message) {
      setError(err.message);
    }
    setTimeout(() => {
      setError("");
    }, 5000);
  }
};
```

```
return (
  <Box
    width={isNotMobile ? "40%" : "80%"}
    p={"2rem"}
    m={"2rem auto"}
    borderRadius={5}
    sx={{ boxShadow: 5 }}
    backgroundColor={theme.palette.background.alt}
  >
    <Collapse in={error}>
      <Alert severity="error" sx={{ mb: 2 }}>
        {error}
      </Alert>
    </Collapse>
    <form onSubmit={handleSubmit}>
      <Typography variant="h3">Generate Paragraph</Typography>

      <TextField
        placeholder="add your text"
        type="text"
        multiline={true}
        required
        margin="normal"
        fullWidth
```

```
    value={text}
    onChange={(e) => {
      settext(e.target.value);
    }}
  />

  <Button
    type="submit"
    fullWidth
    variant="contained"
    size="large"
    sx={{ color: "white", mt: 2 }}
  >
    Generate
  </Button>
  <Typography mt={2}>
    not this tool ? <Link to="/">GO BACK</Link>
  </Typography>
</form>

{para ? (
  <Card
    sx={{
      mt: 4,
```

```
      border: 1,
      boxShadow: 0,
      height: "500px",
      borderRadius: 5,
      borderColor: "natural.medium",
      bgcolor: "background.default",
    }}
  >
    <Typography p={2}>{para}</Typography>
  </Card>
) : (
  <Card
    sx={{
      mt: 4,
      border: 1,
      boxShadow: 0,
      height: "500px",
      borderRadius: 5,
      borderColor: "natural.medium",
      bgcolor: "background.default",
    }}
  >
    <Typography
      variant="h5"
```

```
            color="natural.main"
            sx={{
              textAlign: "center",
              verticalAlign: "middel",
              lineHeight: "450px",
            }}
          >
            Your Paragraph Will Apprea Here
          </Typography>
        </Card>
      )}
    </Box>
  );
};

export default Paragraph;
```

7. Register.js

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import toast from "react-hot-toast";
import axios from "axios";
import {
  Box,
```

```
  Typography,
  useTheme,
  useMediaQuery,
  TextField,
  Button,
  Alert,
  Collapse,
} from "@mui/material";

const Register = () => {
  const theme = useTheme();
  const navigate = useNavigate();
  //media
  const isNotMobile = useMediaQuery("(min-width: 1000px)");
  // states
  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  //register ctrl
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
```

```
      await axios.post("http://localhost:8080/api/v1/auth/register", { username,
email, password });
      toast.success("User Register Successfully");
      navigate("/login");
    } catch (err) {
      console.log(error);
      if (err.response.data.error) {
        setError(err.response.data.error);
      } else if (err.message) {
        setError(err.message);
      }
      setTimeout(() => {
        setError("");
      }, 5000);
    }
  };
  return (
    <Box
      width={isNotMobile ? "40%" : "80%"}
      p={"2rem"}
      m={"2rem auto"}
      borderRadius={5}
      sx={{ boxShadow: 5 }}
      backgroundColor={theme.palette.background.alt}
    >
```

```
<Collapse in={error}>
  <Alert severity="error" sx={{ mb: 2 }}>
    {error}
  </Alert>
</Collapse>
<form onSubmit={handleSubmit}>
  <Typography variant="h3">Sign Up</Typography>
  <TextField
    label="username"
    required
    margin="normal"
    fullWidth
    value={username}
    onChange={(e) => {
      setUsername(e.target.value);
    }}
  />
  <TextField
    label="email"
    type="email"
    required
    margin="normal"
    fullWidth
    value={email}
```

```
  onChange={(e) => {
    setEmail(e.target.value);
  }}
/>
<TextField
  label="password"
  type="password"
  required
  margin="normal"
  fullWidth
  value={password}
  onChange={(e) => {
    setPassword(e.target.value);
  }}
/>
<Button
  type="submit"
  fullWidth
  variant="contained"
  size="large"
  sx={{ color: "white", mt: 2 }}
>
  Sign Up
</Button>
```

```
        <Typography mt={2}>
          Already have an account ? <Link to="/login">Please Login</Link>
        </Typography>
      </form>
    </Box>
  );
};

export default Register;
```

8.Usermodule.js

```
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");
const JWT = require("jsonwebtoken");
const cookie = require("cookie");

//models
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: [true, "USername is Required"],
  },
  email: {
    type: String,
```

```
    required: [true, "Email is required"],
    unique: true,
  },
  password: {
    type: String,
    required: [true, "Password is required"],
    minlength: [6, "Password length should be 6 character long"],
  },
  customerId: {
    type: String,
    default: "",
  },
  subscription: {
    type: String,
    default: "",
  },
});

//hashed password
userSchema.pre("save", async function (next) {
  //update
  if (!this.isModified("password")) {
    next();
  }
```

```
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

//match password
userSchema.methods.matchPassword = async function (password) {
  return await bcrypt.compare(password, this.password);
};

//SIGN TOKEN
userSchema.methods.getSignedToken = function (res) {
  const acccesToken = JWT.sign(
    { id: this._id },
    process.env.JWT_ACCESS_SECRET,
    { expiresIn: process.env.JWT_ACCESS_EXPIREIN }
  );
  const refreshToken = JWT.sign(
    { id: this._id },
    process.env.JWT_REFRESH_TOKEN,
    { expiresIn: process.env.JWT_REFRESH_EXIPREIN }
  );
  res.cookie("refreshToken", `${refreshToken}`, {
    maxAge: 86400 * 7000,
```

```
    httpOnly: true,
  });
};

const User = mongoose.model("User", userSchema);

module.exports = User;
```