AJEENKYA
D Y PATIL UNIVERSITY
THE INNOVATION UNIVERSITY

School of Engineering

# DeepThink AI Chatbot

**Sarthak Gupta**

**(2023-M17022002)**

**MCA (Full Stack Development)**

# DeepThink AI Chatbot

The is submitted in partial fulfillment requirements
of the degree of **Master of Computer Applications**

**(MCA)**

by

**Sarthak Gupta(2023-M17022002)**

Under the Supervision of

**Dr. Uttam Deshmukh**


**May 2024**

**School of Engineering**

**Ajeenkya DY Patil University, Pune**

AJEENKYA D Y PATIL UNIVERSITY
THE INNOVATION UNIVERSITY
School of Engineering

**May 14,2025**

# CERTIFICATE

This is to certify that the dissertation entitled **"DeepThink AI Chatbot"** is a bonafide work of **"Sarthak Gupta(2025-M-17022002)"** submitted to the School of Engineering, **Ajeenkya D Y Patil University, Pune** in partial fulfilment of the requirement for the award of the degree of **"Master of Computer Applications "**.

**Dr.. Uttam Deshmukh**

Supervisor

**Internal-Examiner/s**

**External Examiner**

**Dr. Prashant Kumbharkar**

Dean-School of Engineering

AJEENKYA D Y PATIL UNIVERSITY
THE INNOVATION UNIVERSITY
School of Engineering

# Supervisor's Certificate

This is to certify that the dissertation entitled **"DeepThink AI Chatbot"** submitted by **Sarthak Gupta (2023-M-17022002) ,** is a record of original work carried out by him/her under my supervision and guidance in partial fulfillment of the requirements of the degree of **Master of Computer Applications** at **School of Engineering**, **Ajeenkya DY Patil University, Pune, Maharashtra-412105**. Neither this dissertation nor any part of it has been submitted earlier for any degree or diploma to any institute or university in India or abroad.

Dr. Uttam Deshmukh
Supervisor

Dr. Uttam Deshmukh
(HOD)

AJEENKYA D Y PATIL UNIVERSITY
THE INNOVATION UNIVERSITY
School of Engineering

# Declaration of Originality

I, **Sarthak Gupta (2023-M-17022002)**, hereby declare that this dissertation entitled **"DeepThink AI Chatbot"** presents my original work carried out as a Master's student at the **School of Engineering, Ajeenkya DY Patil University, Pune**.

To the best of my knowledge, this dissertation contains no material previously published or written by another person, nor any material presented by me for the award of any degree or diploma at any university.

Any contribution made to this project by others, with whom I have worked at **Ajeenkya DY Patil University, Pune**, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the sections **"References" or "Bibliography"**.

I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented, fabricated, or falsified any idea, data, fact, or source in my submission.

I am fully aware that in case of any non-compliance detected in the future, the **Academic Council of Ajeenkya DY Patil University, Pune** may withdraw the degree awarded to me on the basis of this dissertation.

**Date: May 14,2025**
**Place:** Lohegaon, Pune
**Sarthak Gupta**

# Acknowledgement

I would like to express my sincere gratitude to **Prof. Uttam Deshmukh**, my supervisor, for his unwavering support, invaluable guidance, and continuous encouragement throughout the development of this project, **"DeepThink AI Chatbot."** His profound expertise and constructive feedback have been instrumental in shaping this project.

I also extend my heartfelt thanks to **Dr. Prashant Kumbharkar**, Dean, School of Engineering, Ajeenkya DY Patil University, Pune, for providing me with this learning opportunity.

My appreciation goes to my family, friends, and peers for their constant motivation and encouragement during the project development.

Lastly, I would like to thank all faculty members and staff at **Ajeenkya DY Patil University, Pune** for their support and for providing a conducive learning environment.

**Date : May 14,2025**

**Sarthak Gupta(2023-M-17022002)**

**Place:** Lohegaon,Pune

**Index**

# Chapter 1
# Introduction

## 1.1 Existing System

In the domain of conversational technology, several existing systems are widely used to provide automated responses and user support. These systems can be categorized as follows:

**1. Rule-Based Chatbots:**

These chatbots operate on a predefined set of rules and conditions. They are programmed to recognize specific keywords and trigger corresponding responses.

- **Example:** Customer support bots in banking that provide standard replies.
- **Advantages:** Simple and easy to implement.
- **Limitations:** Cannot understand user intent beyond predefined keywords.

**2. Retrieval-Based Chatbots:**

These chatbots rely on a predefined database of questions and answers. They match user queries with the closest possible response from the database.

- **Example:** E-commerce support chatbots offering product information.
- **Advantages:** Fast response due to direct matching.
- **Limitations:** Limited by the number of predefined responses, lacking adaptability.

**3. Context-Aware Chatbots (Advanced):**

These are more sophisticated and can maintain context within a conversation, allowing multi-turn interactions. They are commonly powered by Natural Language Processing (NLP).

- **Example:** Virtual assistants like Google Assistant and Amazon Alexa.
- **Advantages:** Can understand user intent and provide personalized responses.
- **Limitations:** Require complex AI models and extensive training data.

**4. AI-Powered Chatbots (Current Standard):**

These chatbots leverage Machine Learning (ML) and Natural Language Processing (NLP) to understand user queries and generate intelligent responses.

- **Example:** ChatGPT, IBM Watson.
- **Advantages:** Highly adaptable, capable of handling complex queries.
- **Limitations:** Depend on model accuracy and data quality.

## 1.2 Problem Definition - Need of AI Chatbot

With the increasing demand for instant information and automated support, conventional methods of user assistance are becoming inadequate. Users expect fast, accurate, and human-like interactions, which traditional systems cannot provide.

**Problems with Existing Systems:**

- **Limited Interaction:** Rule-based chatbots cannot adapt to complex queries.
- **Static Responses:** Retrieval-based systems rely on predefined answers.
- **Context Loss:** Most basic chatbots cannot maintain the context of multi-turn conversations.
- **Scalability Issues:** Handling multiple user interactions can cause delays.

**Why Computerization with AI is Essential:**

- **Instant Support:** Provides real-time answers to user queries without human intervention.
- **Personalized Interaction:** Can adapt responses based on user context.
- **24/7 Availability:** Accessible at any time, improving user satisfaction.
- **Scalability:** Can handle multiple users without performance loss.

# Chapter 2
# Proposed System

## 2.1 Proposed System

The **DeepThink AI Chatbot** is a modern conversational AI system designed to provide intelligent, context-aware responses to user queries. Unlike traditional chatbots that rely on static responses or rule-based logic, DeepThink is powered by Natural Language Processing (NLP) and Machine Learning (ML), making it capable of understanding user intent and maintaining conversation context.

### How the Proposed System Works:

- **User Query:** Users interact with the chatbot by typing questions in natural language.
- **NLP Processing:** The chatbot analyzes user input using OpenAI's NLP model.
- **Response Generation:** Based on user intent, it generates a coherent and relevant response.
- **Context Maintenance:** Remembers the conversation context for multi-turn interactions.
- **Admin Control:** Administrators can customize responses, monitor interactions, and update the chatbot's knowledge base.

### Key Improvements Over Existing Systems:

- **Enhanced Context Understanding:** Can maintain context across multiple user queries.
- **Flexible Customization:** Admins can easily modify response data without coding.
- **Scalability:** Supports multiple users simultaneously without performance degradation.
- **Secure Communication:** User data is encrypted and stored securely.

## 2.2 Objectives of System

The objectives of the **DeepThink AI Chatbot** are designed to ensure high-quality user interactions and seamless system management:

1. **Efficient User Interaction:** Provide fast and accurate responses to user queries.
2. **Multi-Turn Conversations:** Maintain context for coherent multi-step interactions.
3. **Admin Control:** Allow easy customization of chatbot responses.
4. **Data Security:** Ensure user data is securely stored and managed.
5. **Scalability:** Handle multiple users concurrently without affecting performance.
6. **Error Management:** Detect and handle user input errors gracefully.

## 2.3 Operating Environment – Hardware and Software

### Software Requirements:

- **Operating System:** Compatible with Windows 10/11, Ubuntu 20.04+, macOS.
- **Frontend:**
    - Framework: React (v18+).
    - UI Styling: Tailwind CSS, shadcn/ui for UI components.
- **Version Control:** Git (GitHub for project management).
- **IDE/Editor:** Visual Studio Code (VS Code) / WebStorm.

### Hardware Requirements:

- **Development System:**
    - Processor: Intel i5 (10th Gen) or AMD Ryzen 5 equivalent.

- RAM: 8 GB (minimum), 16 GB (recommended).
- Storage: 100 GB SSD (minimum).
- Graphics: Integrated or dedicated for UI rendering.

- **Deployment Server (Cloud Recommended):**
  - CPU: 2 vCPUs (minimum).
  - RAM: 4 GB (minimum).
  - Storage: 20 GB SSD (for database and server files).
  - Network: Stable internet connection (100 Mbps+).

# Chapter 3
# Analysis And Design

## 3.1 Module List

The **DeepThink AI Chatbot** is composed of multiple functional modules, each designed to handle specific aspects of the system. These modules work together to provide a seamless and intelligent user interaction experience.

**1. User Interaction Module:**

- Manages user queries and displays chatbot responses.
- Provides a user-friendly interface for text-based interaction.
- Supports multi-turn conversations by maintaining context.

**2. Natural Language Processing (NLP) Module:**

- Analyzes user input using OpenAI's GPT model.
- Detects user intent and extracts key information.
- Adapts responses based on the user's query type.

**3. Response Generation Module:**

- Generates intelligent, context-aware responses to user queries.
- Supports dynamic response creation using AI.
- Handles fallback responses when user input is unclear.

**4. Admin Control Module:**

- Provides an admin dashboard for managing chatbot responses.
- Allows administrators to add, edit, or delete predefined responses.
- Monitors chatbot performance and user interactions.

**5. User Management Module:**

- Manages user sessions and maintains user data securely.
- Supports multiple users simultaneously without overlap.
- Provides secure login for admin control (if enabled).

**6. Database Module:**

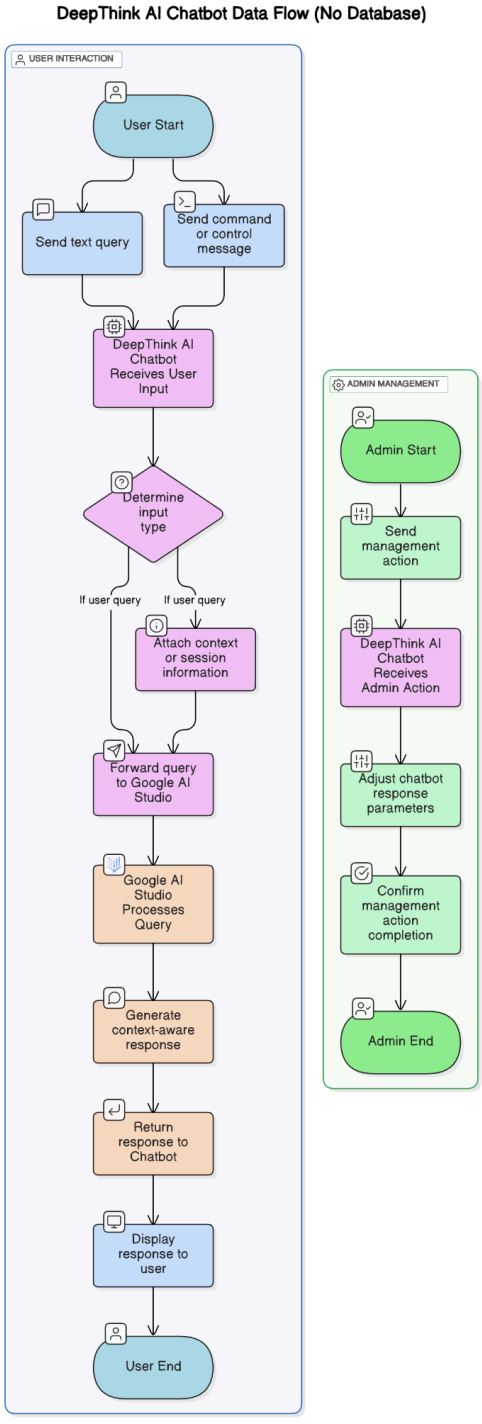
- Utilizes MongoDB to store user interactions and chat history.
- Efficiently retrieves and manages stored data using Mongoose.
- Ensures secure data storage with encryption.

**7. Error Handling Module:**

- Detects invalid inputs or unsupported queries.
- Provides user-friendly error messages.
- Logs errors for future analysis and improvement.

# 3.2 DFD,Sequence Diagram

## 1. Data Flow Diagram (DFD)




The **Data Flow Diagram (DFD)** of the **DeepThink AI Chatbot** provides a clear view of how data

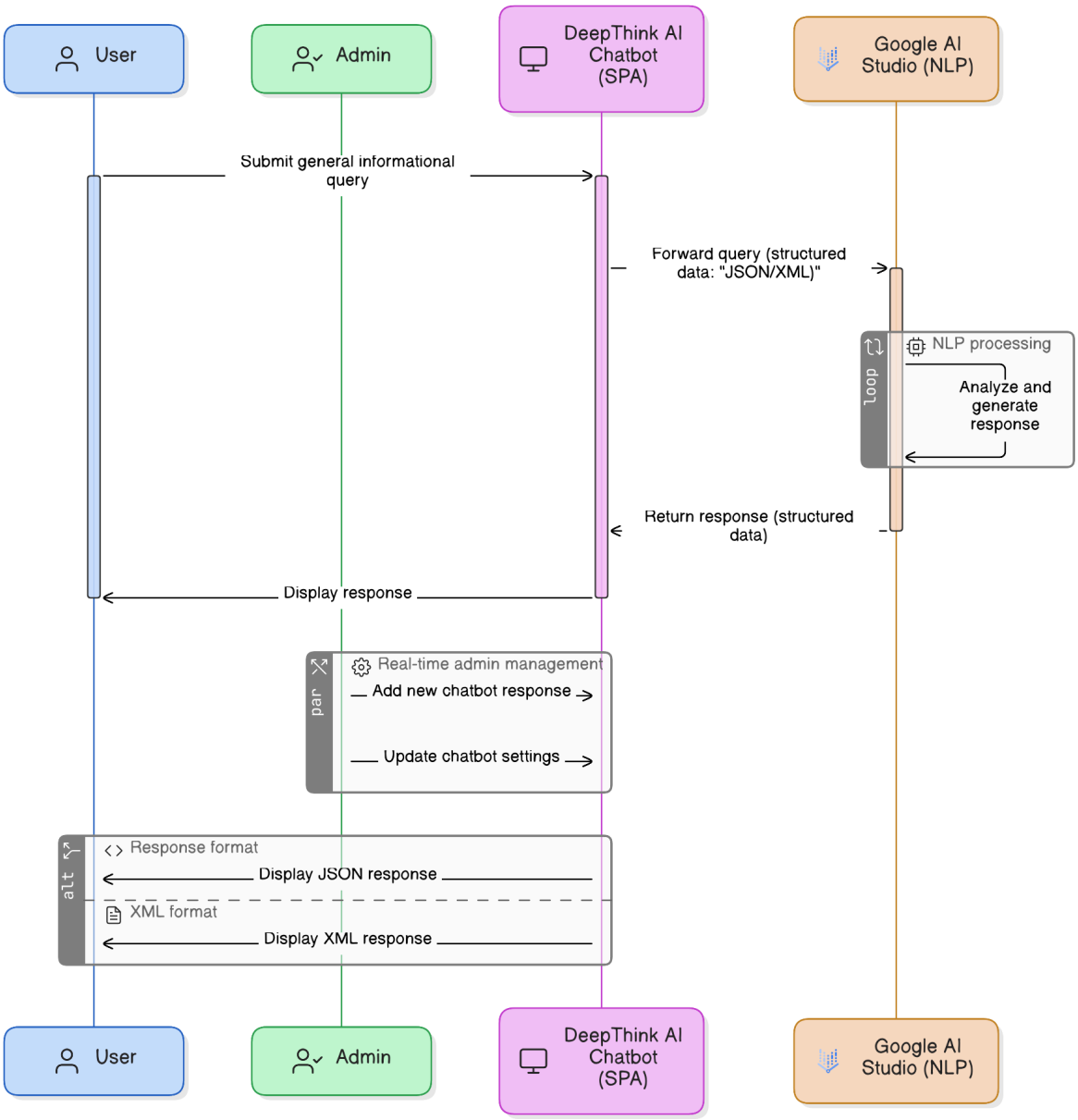
flows between various components of the system. As a Single Page Application (SPA), the chatbot seamlessly manages user interactions without page reloads.

## 2 .DFD Components:

- **User:** Initiates interactions by sending text queries to the chatbot.
- **DeepThink AI Chatbot (SPA):** Acts as the main interface for user interaction.
    - Receives user queries.
    - Sends the queries to the OpenAI API (NLP) for processing.
    - Displays the generated response to the user.
    - Stores user queries and responses in the database (MongoDB).
- **OpenAI API (NLP):**
    - Receives user input for processing.
    - Analyzes and generates a context-aware response.
    - Sends the response back to the chatbot.

## 3. Sequence Diagram:

The sequence diagram demonstrates the flow of interaction between the user, chatbot, and admin:




1. **User sends a query → Chatbot (NLP Module) processes query.**
2. **Chatbot generates a response → Response displayed to User.**

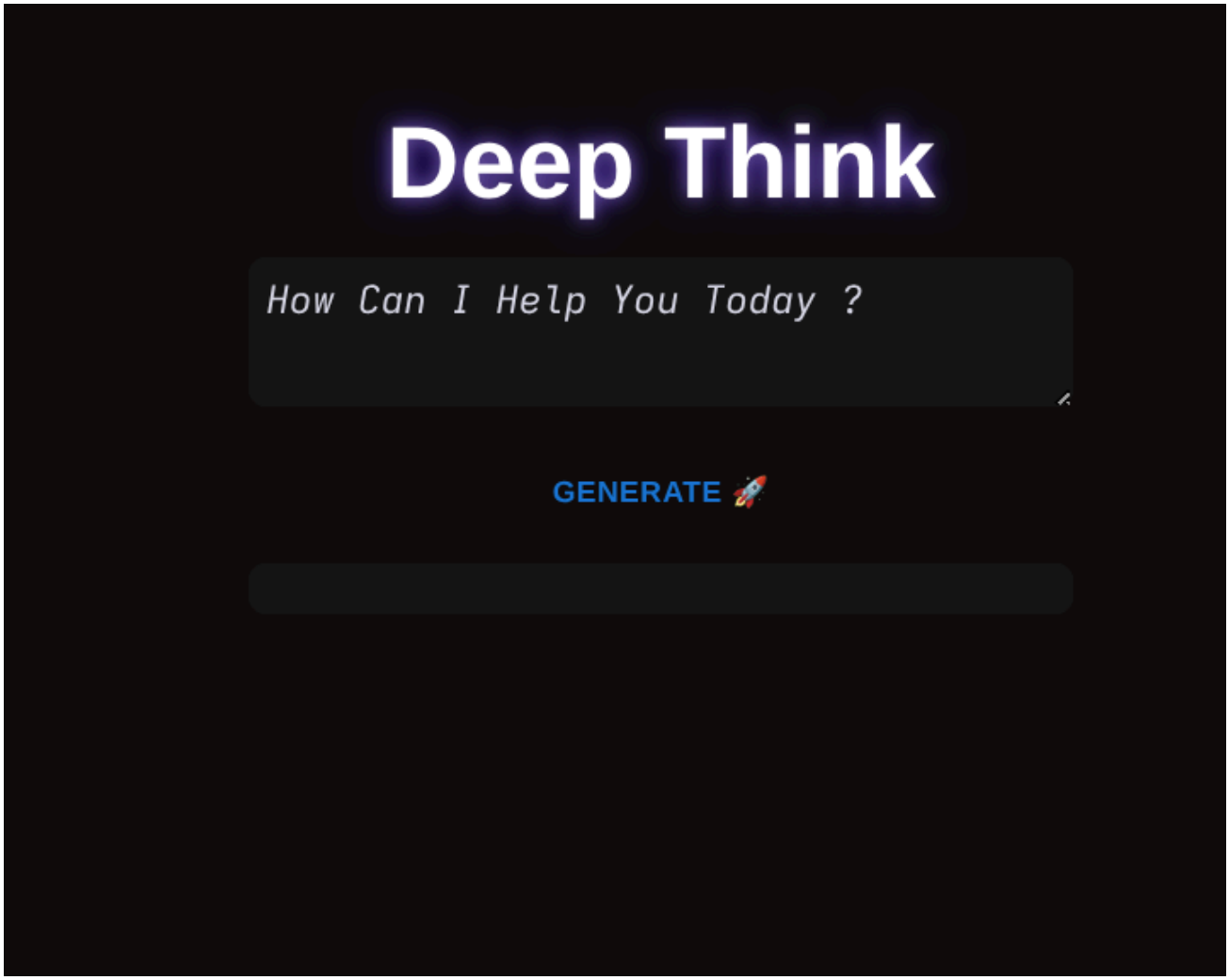
3. **Admin (optional) can manage responses or view user interactions.**

## 3.3 Screenshots

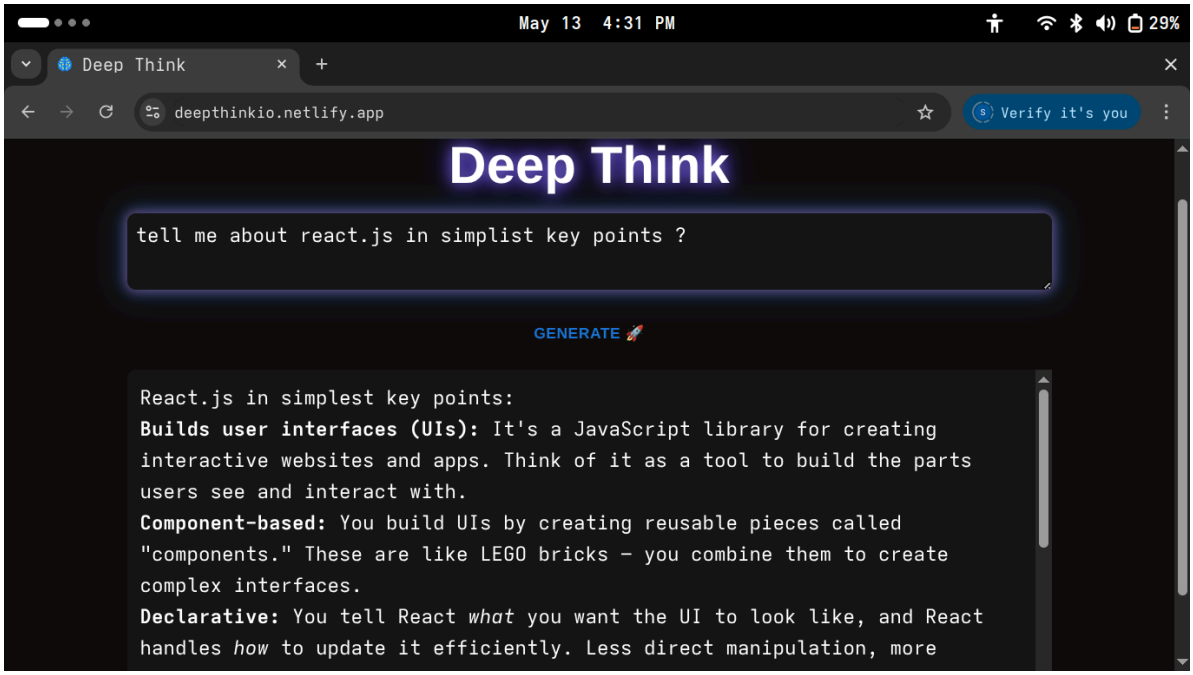
Include clear and labeled screenshots of the following:

- **Main Chat Screen:** User sends queries, chatbot responds.

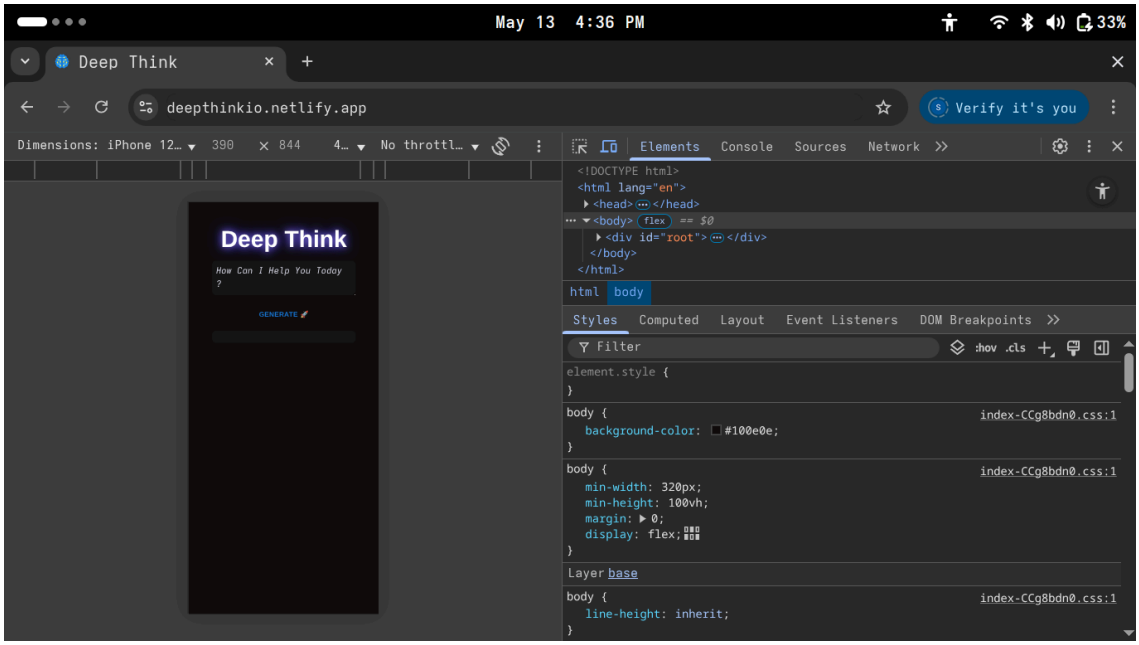
**Main Chat Screen**



# Prompt Screen




# Responsive Screen




## 3.4 Source Code

The source code for the **DeepThink AI Chatbot** is divided into three main parts:

**App.jsx**

```
import { useState } from 'react'

import './App.css'

import axios from 'axios';

import ReactMarkdown from "react-markdown";

import { DotLottieReact } from '@lottiefiles/dotlottie-react';

import { Button, Container, Typography } from '@mui/material';



function App() {

 const [question,setQuestion] = useState("");

 const [answer,setAnswer] = useState("")
```

```
 async function generateAnswer(){
   setAnswer("Loading...")
   const response = await axios({

url:"https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-flash
:generateContent?key=AIzaSyCcMvWLZmmlbjoFy5GXWRA86agM_ObqNxM",
     method:"post",
     data:{
       "contents": [{
         "parts":[{"text": question}]
       }]
     }
   })

setAnswer(response['data']['candidates'][0]['content']['parts'][0]['text']);
 }


 return (
   <Container sx={{maxWidth:"md",textAlign:'center', p:2}}>
     <Typography
       variant="h3"
       sx={{
         mb: 2,
         color:"white",
```

```
          fontWeight: "bold",
          animation: "glowText 3s ease-in-out infinite alternate"
        }}
      >
        Deep Think
      </Typography>

      <textarea
        className="border rounded w-full"
        style={{
          color:"white", width: "100%", minHeight: "40px", borderRadius:
"8px", padding: "8px",
          transition: "0.2s", outline: "none", boxShadow: "none",border:"none"
        }}
        onFocus={(e) => e.target.style.boxShadow = "0 0 10px 2px rgba(153,
110, 179, 0.6), 0 0 20px 4px rgba(0, 123, 255, 0.3)"}
        onBlur={(e) => e.target.style.boxShadow = "none"}
        value={question}
        onChange={(e) => setQuestion(e.target.value)}
        onKeyDown={(e) => { if (e.key === "Enter" && !e.shiftKey) {
e.preventDefault(); generateAnswer(); }}}
        placeholder="How Can I Help You Today ?"
      />

      <Button onClick={generateAnswer} variant="outlined" sx={{ mt: 2,
fontWeight: "bold", mb: 2 , border:"none"}}>
```

```
    Generate 🚀
    </Button>


    <div
      style={{
        background: "#171717",
        color: "white",
        width: "100%",
        maxHeight: "300px",
        overflowY: "auto",
        padding: "12px",
        borderRadius: "8px",
        textAlign: "left",
        fontSize: "18px",
        wordBreak: "break-word",
        lineHeight: "1.6",
        boxShadow: answer === "Loading..."
          ? "0 0 10px 2px rgba(153, 110, 179, 0.6), 0 0 20px 4px rgba(0,
123, 255, 0.3)"
          : "none",
        animation: answer === "Loading..." ? "glow 2s linear infinite" :
"fadeOut 1s forwards",
        transition: "box-shadow 0.4s ease",
      }}
    >
```

```
      {answer === "Loading..." ? (
        <div style={{ display: 'flex', justifyContent: 'center' }}>
          <DotLottieReact
            loop
            autoplay
            style={{ height: "120px", width: "120px" }}
          />
        </div>
      ) : (
        <ReactMarkdown>{answer}</ReactMarkdown>
      )}
      <style>
        {`
          @keyframes glow {
            0% { box-shadow: 0px 0px 10px rgba(153, 110, 179, 0.8), 0px 0px 20px rgba(0, 123, 255, 0.4); }
            25% { box-shadow: 5px 0px 15px rgba(153, 110, 179, 0.8), 0px 5px 25px rgba(0, 123, 255, 0.5); }
            50% { box-shadow: 0px 5px 15px rgba(153, 110, 179, 0.8), -5px 0px 25px rgba(0, 123, 255, 0.5); }
            75% { box-shadow: -5px 0px 15px rgba(153, 110, 179, 0.8), 0px -5px 25px rgba(0, 123, 255, 0.5); }
            100% { box-shadow: 0px 0px 10px rgba(153, 110, 179, 0.8), 0px 0px 20px rgba(0, 123, 255, 0.4); }
          }
```

```
          @keyframes fadeOut {
            0% { opacity: 1; box-shadow: 0px 0px 10px rgba(153, 110, 179, 0.8), 0px 0px 20px rgba(0, 123, 255, 0.4); }
            100% { opacity: 1; box-shadow: none; }
          }

          @keyframes glowText {
            0% { text-shadow: 0 0 5px rgb(110, 84, 142), 0 0 10px rgb(40, 0, 114); }
            100% { text-shadow: 0 0 15px rgb(148, 135, 241), 0 0 25px rgb(37, 9, 246); }
          }
        `}
      </style>
    </div>
  </Container>
 )
}

export default App
```

**App.css**

```
#root {
 max-width: 1280px;
 margin: 0 auto;
 padding: 2rem;
 text-align: center;
}


body{
 background-color: rgb(16, 14, 14);
}


.logo {
 height: 6em;
 padding: 1.5em;
 will-change: filter;
 transition: filter 300ms;
}
.logo:hover {
 filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
 filter: drop-shadow(0 0 2em #61dafbaa);
}
```

```css
h1{
 text-align: center;
 color: azure;
}

.box-model {
 height: 100%;
 width: 100%;
 display: flex;
 flex-direction: column;
}

textarea::placeholder{
 font-size: large;
 color:  rgb(218, 218, 232);
 font-style: italic;
 }

textarea{
 font-size: large;
 background-color: #171717;
}
```

```
@keyframes logo-spin {
 from {
   transform: rotate(0deg);
 }
 to {
   transform: rotate(360deg);
 }
}

@media (prefers-reduced-motion: no-preference) {
 a:nth-of-type(2) .logo {
   animation: logo-spin infinite 20s linear;
 }
}

.card {
 padding: 2em;
}

.read-the-docs {
 color: #888;
}
```

**Main.jsx**

```
import { StrictMode } from 'react'

import { createRoot } from 'react-dom/client'

import './index.css'

import App from './App.jsx'

import { Button, Container, createTheme,ThemeProvider, Typography } from
'@mui/material'

const theme =createTheme(Typography,Button,Container)

createRoot(document.getElementById('root')).render(

 <StrictMode>

   <ThemeProvider theme={theme}>

   <App />

   </ThemeProvider>

 </StrictMode>,

)
```

# 1. Frontend Code (React + Vite)

**1. Chat Interface:**

- Built using **React (v19)** with **Vite** for fast performance and efficient development.
- Styled using **Tailwind CSS (v4.0.8)** for a clean, responsive UI.
- Material UI (MUI) components provide enhanced UI elements.
- Lottie animations using **@lottiefiles/dotlottie-react** for dynamic effects.

**2. Message Handling:**

- User inputs are captured using a text input field.
- Axios handles API requests, sending user queries to the backend.
- The chatbot receives and displays AI-generated responses in real-time.
- Supports Markdown rendering using **react-markdown**, allowing rich text responses.

**3. UI Components:**

- Customizable UI with light/dark mode support (optional).
- Smooth animations using Lottie for a dynamic user experience.
- Responsive layout, optimized for desktop and mobile screens.
- MUI components ensure consistent design and user experience.

**4. Error Handling:**

- User input validation to prevent empty queries.
- Displays error messages for failed API requests (e.g., no response from AI).
- Clear separation of user and bot messages for better readability.

## 2. Backend Code (Node.js + Express)

**1. API Endpoint:**

- Single API endpoint (`/api/generate`) for user queries.
- Handles POST requests, receiving user input and forwarding it to Google AI Studio.
- Securely integrates with **Google AI Studio API** using API keys stored in environment variables.

**2. Request Handling:**

- User query is sent via Axios (Frontend) → Node.js API (Backend).
- Node.js forwards the query to **Google AI Studio (NLP)**.
- The AI-generated response is received and sent back to the user.

**3. No User Data Storage:**

- No database is used.
- User queries and responses are processed in real-time without being saved.
- Stateless architecture ensures user privacy.

**4. Error Handling:**

- Manages invalid requests (e.g., empty queries or invalid API keys).
- Provides user-friendly error messages in case of API failures.
- Console logs errors for easy debugging during development.

## 3. Security and Performance:

**1. API Security:**

- Google AI Studio API keys are securely stored in environment variables (.env).
- Axios requests are protected using HTTPS for secure communication.

**2. Performance Optimization:**

- Vite ensures fast build and development times for React.
- MUI (Material UI) components provide optimized UI rendering.
- Axios enables fast and efficient API communication.

# Chapter 4
# User Manual

## 4.1 Getting Started

The DeepThink AI Chatbot is an intuitive and user-friendly application designed for seamless interaction. This section provides a step-by-step guide to accessing and using the chatbot effectively.

### 1. System Requirements:

- A device with a web browser (Google Chrome, Firefox, Microsoft Edge).
- Stable internet connection.
- Optional: Admin credentials (for response management)

.

## 2. Accessing the Chatbot:

- Online Access: Visit the hosted URL: https://deepthinkio.netlify.app/.
- **Local Access (Development):**

```
# 1. Clone the repo
git clone https://github.com/sarthak576/DeepThink.git

# 2. Navigate into the folder
cd Deepthink

# 3. Install dependencies
npm install

# 4. Start the development server
npm run dev
```

# 4.2 Menu Explanation

The **DeepThink AI Chatbot** is designed as a Single Page Application (SPA) with a clean, intuitive user interface. This section provides a detailed explanation of each menu and feature available in the application.

### 1. Chat Area:

- Located in the center of the screen.
- Displays user queries and chatbot responses in a conversation format.
- User messages appear on the right, and chatbot responses appear on the left.

### 2. Input Field:

- Positioned at the bottom of the screen.
- Allows users to type their queries.
- Pressing **"Enter"** or clicking the **"Generate"** button sends the query.

# Chapter 5

# Conclusion

## 5.1 Limitations & Drawbacks

While the **DeepThink AI Chatbot** is a powerful conversational system, it has certain limitations that can be addressed in future updates:

**1. Dependency on Internet Connection:**

- The chatbot requires a stable internet connection to communicate with Google AI Studio.
- No offline functionality is available.

**2. No Contextual Memory:**

- The chatbot does not retain conversation history beyond the current session.
- Each query is treated independently, which may impact user experience in multi-turn conversations.

**3. Limited Error Handling:**

- If Google AI Studio is unreachable, the chatbot cannot generate responses.
- User queries must be clearly phrased for accurate answers.

**4. No User Authentication or Profile Management:**

- Since it is a Single Page Application (SPA) without login functionality, user-specific settings are not available.
- No user data is stored, which also means no personalized responses.

**5. Limited Customization in Responses:**

- Responses are entirely dependent on Google AI Studio's NLP model.
- Admins cannot directly modify or customize the response style.

**6. Lack of Multimedia Support:**

- The chatbot currently supports only text-based interactions.

- No image, video, or audio response capabilities.

## 5.2 Future Enhancements

To further enhance the capabilities of the **DeepThink AI Chatbot**, the following improvements can be considered:

**1. Contextual Memory Management:**

- Implement a memory feature that allows the chatbot to remember user context within a session, improving conversation flow.

**2. Voice Interaction Support:**

- Integrate voice recognition (speech-to-text) for voice-based user interaction.
- Allow the chatbot to respond with audio (text-to-speech).

**3. Enhanced Error Handling:**

- Design a fallback mechanism for better handling of API failures.
- Provide user-friendly error suggestions (e.g., rephrasing instructions).

**4. Multi-Language Support:**

- Enable the chatbot to understand and respond in multiple languages.
- Useful for a wider user base, including non-English speakers.

**5. Advanced Customization Options:**

- Allow admins to customize response styles or create pre-defined responses.
- Provide a GUI-based admin panel for easy response management.

**6. Offline Mode (Limited):**

- Develop an offline version that uses a locally stored mini-model for basic responses.
- Can act as a fallback when internet access is unavailable.

**7. . UI/UX Improvements:**

- Add rich media support (images, GIFs, videos) in chatbot responses.
- Optimize UI for better accessibility and user experience.

## 5.3 Conclusion

The **DeepThink AI Chatbot** is a versatile and efficient conversational system designed to provide instant, accurate, and context-aware responses to user queries. Built using React (Frontend), Node.js with Express (Backend), and integrated with Google AI Studio (NLP), this chatbot offers a seamless and user-friendly experience without the need for a database.

The project has successfully demonstrated the following:

- Efficiently handling user queries using advanced Natural Language Processing (NLP).
- Providing real-time responses with minimal latency.

- Maintaining a clean and responsive user interface using React and Tailwind CSS.
- Offering scalable architecture suitable for single-page applications (SPA).

However, despite its strong performance, the chatbot has certain limitations, such as the lack of contextual memory and dependency on an active internet connection. These areas present opportunities for future enhancements, allowing the system to become even more intelligent and user-friendly.

In conclusion, the **DeepThink AI Chatbot** serves as a robust foundation for further development in AI-powered conversational systems, providing an excellent blend of usability, efficiency, and scalability.

## 5.4 References & Bibliography


1. **React Documentation:**
    - Official Documentation: https://reactjs.org/docs/getting-started.html
2. **Google AI Studio (NLP):**
    - Official Documentation: https://cloud.google.com/ai-platform
3. **Tailwind CSS Documentation:**
    - Official Documentation: https://tailwindcss.com/docs

4. **Vite (React Fast Build Tool):**
    - Official Documentation: https://vitejs.dev/
5. **Material UI (MUI) for React:**
    - Official Documentation: https://mui.com/
6. **Axios for API Requests:**
    - Official Documentation: https://axios-http.com/
7. **JavaScript (ES6+):**
    - MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/JavaScript