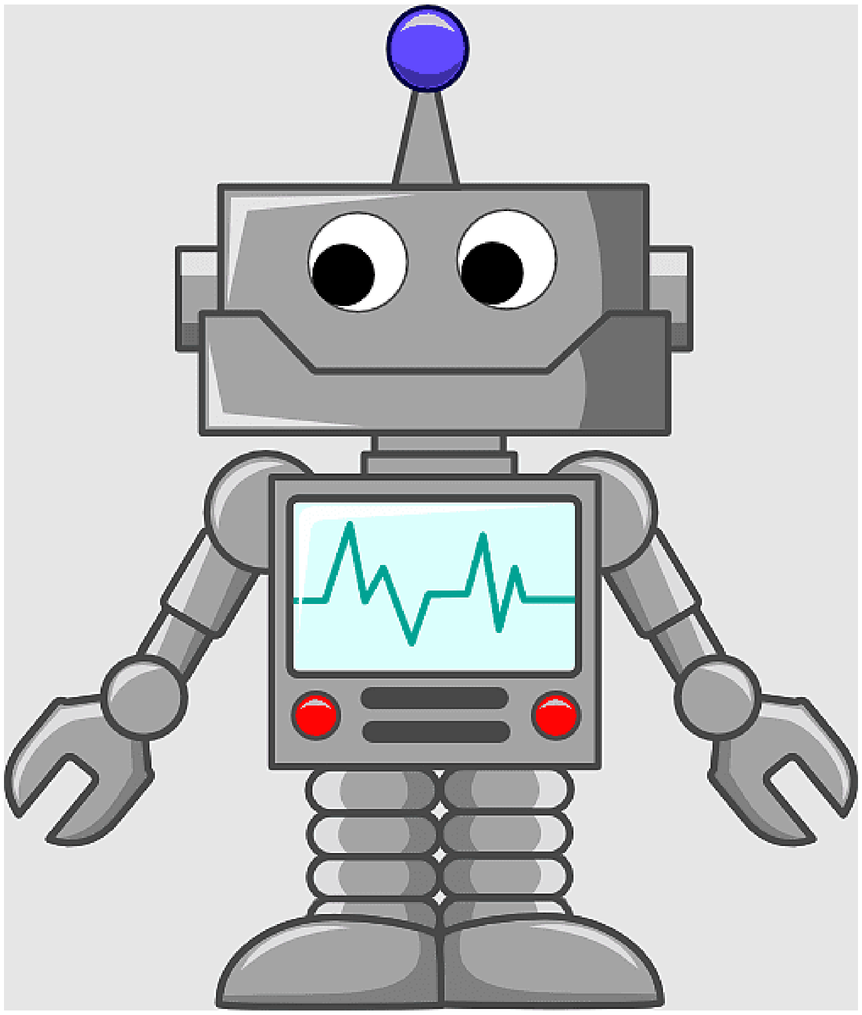
AI CHAT BOT

Machine Learning



GAM/IT/2021/F/0122

W.P.K.D. WEERASINGHE

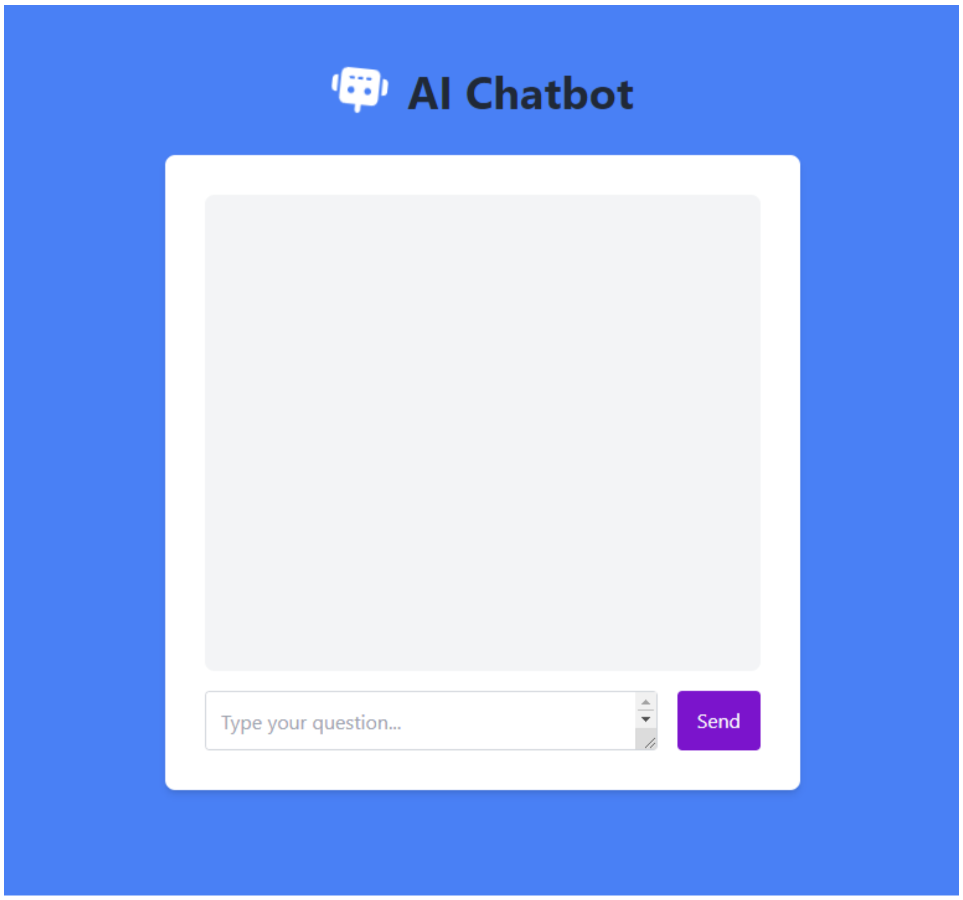
Contents

Introduction

The AI ChatBot project aims to leverage Google's Gemini AI within a MERN stack environment to provide users with conversational interactions. Targeting individuals seeking quick and accurate responses, this chatbot integrates advanced natural language processing for seamless communication.

Design and Development Process

The design process began with thorough research on conversational AI frameworks and platforms. Leveraging the MERN stack - MongoDB, Express.js, React.js, and Node.js - provided a robust foundation for both frontend and backend development. Iterative prototyping and user testing informed the implementation of features for intuitive user interactions.



Features and Functionality

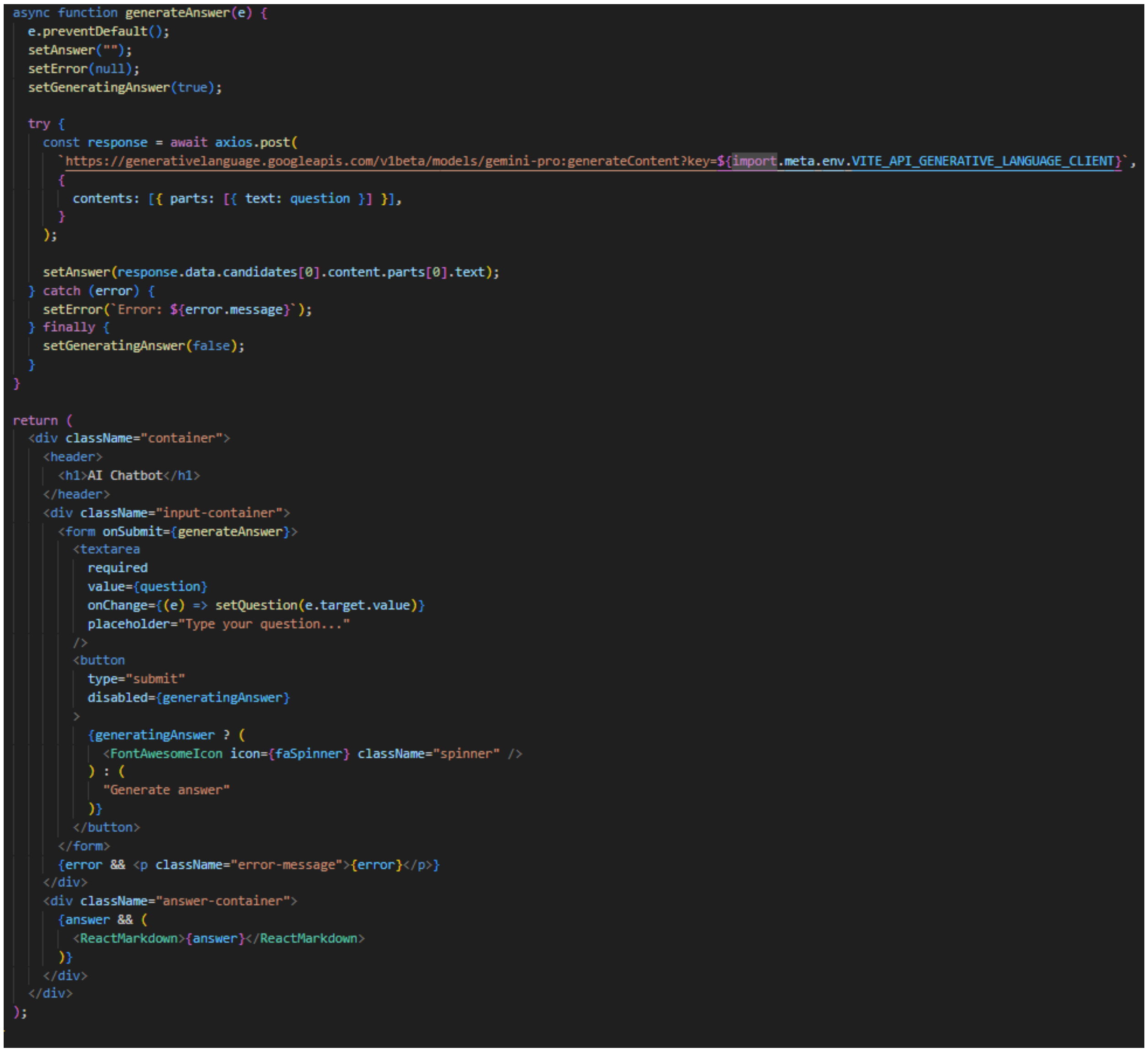
The AI ChatBot offers users a simple interface to input queries and receive responses generated by Google's Gemini AI. It leverages React for dynamic frontend rendering, Axios for seamless HTTP requests, and FontAwesome for visual elements. Key functionalities include real-time question generation and display of AI-generated responses.

User Experience

Designed with user-centric principles, the AI ChatBot ensures a seamless conversational experience. Its intuitive interface guides users through the query process, while feedback mechanisms and loading indicators enhance engagement. Conversational flow is prioritized to mimic human interactions, promoting user satisfaction.

Technical Details

Utilizing a MERN stack architecture, the AI ChatBot integrates Google's Gemini AI API for natural language processing. React.js facilitates responsive frontend design, while Express.js handles server-side logic. MongoDB serves as the database for storing user queries and responses, ensuring scalability and reliability.



```
async function generateAnswer(e) {
  e.preventDefault();
  setAnswer("");
  setError(null);
  setGeneratingAnswer(true);

  try {
    const response = await axios.post(
      `https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent?key=${import.meta.env.VITE_API_GENERATIVE_LANGUAGE_CLIENT}`,
      {
        contents: [{ parts: [{ text: question }] }],
      }
    );

    setAnswer(response.data.candidates[0].content.parts[0].text);
  } catch (error) {
    setError(`Error: ${error.message}`);
  } finally {
    setGeneratingAnswer(false);
  }
}

return (
  <div className="container">
    <header>
      <h1>AI Chatbot</h1>
    </header>
    <div className="input-container">
      <form onSubmit={generateAnswer}>
        <textarea
          required
          value={question}
          onChange={(e) => setQuestion(e.target.value)}
          placeholder="Type your question..."
        />
        <button
          type="submit"
          disabled={generatingAnswer}
        >
          {generatingAnswer ? (
            <FontAwesomeIcon icon={faSpinner} className="spinner" />
          ) : (
            "Generate answer"
          )}
        </button>
      </form>
      {error && <p className="error-message">{error}</p>}
    </div>
    <div className="answer-container">
      {answer && (
        <ReactMarkdown>{answer}</ReactMarkdown>
      )}
    </div>
  </div>
);
```

The App function defines the main component of the AI ChatBot application. Utilizing React.js, it manages the state of user input (question), generated responses (answer), loading status (generatingAnswer), and any encountered errors (error) using React's useState hook.

The generateAnswer function is an asynchronous function triggered upon form submission. It sends a POST request to Google's Gemini AI API endpoint (https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent) with the user's question as input data. The API key for authentication is dynamically provided through environment variables (import.meta.env.VITE_API_GENERATIVE_LANGUAGE_CLIENT).

Upon successful response retrieval, the generated answer is extracted from the API response and stored in the answer state. In case of any errors during the API request, error handling is implemented to display the error message to the user.

The user interface consists of a text input area where users can type their questions and a button to submit the query. While the response is being generated, a loading spinner icon (FontAwesomeIcon) is displayed to provide visual feedback to the user.

Once the answer is generated, it is displayed in the answer container using ReactMarkdown for rendering markdown-formatted content.

Future Enhancements

To enhance the AI ChatBot's capabilities, future iterations may explore advanced AI techniques such as context-aware responses and sentiment analysis. Integration with additional APIs for multimedia content generation and personalized recommendations could further enrich user experiences.

Conclusion

In conclusion, the AI ChatBot project successfully integrates conversational AI within a MERN stack environment, providing users with a seamless and efficient means of obtaining information. By leveraging Google's Gemini AI, the chatbot demonstrates promising potential for further advancements in conversational AI applications.