**Latest always at top**

return (

    <div className="chatbot-wrapper">

      {/\* Sidebar with chat history \*/}

      <div className="sidebar">

        <h2>Chat History</h2>

        <ul>

          {chatHistories.map((history, index) => (

            <li key={index} onClick={() => handleHistoryClick(history)}>

              {history.length > 0 ? history[0].text.substring(0, 20) + '...' : 'Empty Conversation'}

            </li>

          ))}

        </ul>

      </div>

      {/\* Main chat area container \*/}

      <div className="chatbot-container">

        <div className="chat-area" ref={chatAreaRef}

             style={{

                    justifyContent: showButtons ? 'center' : 'flex-start',

                    alignItems: showButtons ? 'center' : 'flex-start',

             }}>

          {/\* Render main dynamic buttons if showButtons is true \*/}

          {showButtons && (

          <div className="dynamic-button-grid">

            {Object.keys(dynamicPrompts).map((prompt, index) => (

              <button

                key={index}

                className="dynamic-button"

                onClick={() => handlePromptClick(prompt)}

              >

                {prompt}

              </button>

            ))}

          </div>

        )}

        {/\* Render chat messages \*/}

        {messages.map((message, index) => (

          <div key={index} className={`message-container ${message.sender}`}

          ref={index === messages.length - 1 ? latestMessageRef : null}>

            {/\* Render the message bubble \*/}

            <div className={`message-bubble ${message.sender}`}>

              <strong>{message.sender === "user" ? "You" : "NutriBot"}:</strong>

              {/\* Use dangerouslySetInnerHTML to render parsed HTML content for AI responses \*/}

              {/\*{message.sender === 'ai'? (\*/}

              {message.sender === 'ai' || message.sender === 'bot' ? (

                <span dangerouslySetInnerHTML={{ \_\_html: parseResponseToHtml(message.text) }} />

              ) : (

                <span>{message.text}</span>

              )}

            </div>

            {/\* Render sub-prompt buttons below the AI message \*/}

            {message.sender === 'ai' && dynamicPrompts[selectedPrompt]?.subPrompts && (

              <div className="sub-prompt-grid">

                {dynamicPrompts[selectedPrompt].subPrompts.map((subPrompt, subIndex) => (

                  <button

                    key={subIndex}

                    className="sub-prompt-button"

                    onClick={() => handleSubPromptClick(subPrompt)}

                  >

                    {subPrompt}

                  </button>

                ))}

              </div>

            )}

          </div>

        ))}

      </div>

      {/\* Input box for sending messages \*/}

      <form className="input-box" onSubmit={(e) => handleSend(null, e)}>

        <input

          type="text"

          className="chat-input"

          value={inputValue}

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

          placeholder="I am Nutribot,your trusted health&wellness guide.Ask me anything......."

          required

        />

        <button type="submit" className="send-button">Send</button>

        <button type="button" className="reset-button" onClick={handleReset}>Reset</button>

      </form>

    </div>

  </div>

);

// Import necessary dependencies

import React, { useState } from 'react';

import axios from 'axios';

import './Chatbot.css';

let eventSource = null;

let controller = null;

if (eventSource) {

  eventSource.close();

}

// Message Component

const Message = ({ sender, text }) => {

  return (

    <div className={`message ${sender === "user" ? "user" : "ai"}`}>

      <div className="message-bubble">

        <strong>{sender === "user" ? "You" : "AI"}:</strong> {text}

      </div>

    </div>

  );

};

// Main Chatbot Component

const Chatbot = () => {

  const [messages, setMessages] = useState([]);

  const [inputValue, setInputValue] = useState('');

  const [chatHistories, setChatHistories] = useState([]);

  const [showButtons, setShowButtons] = useState(true);

  const [subPrompts, setSubPrompts] = useState([]);

  const [selectedPrompt, setSelectedPrompt] = useState(null);

  const dynamicPrompts = {

    "How to select the right protein powders/bars": {

      airesponse : "Check out some curated topics on Protein Powders below or ask your own question",

      subPrompts : [

      "Whey Concentrate versus Whey Isolate",

      "Plant versus Whey protein",

      "Protein Powder Red Flags",

      "Right protein for Specific health concerns",

     ],

    },

    "Get precise supplement insights for specific health needs":{

      airesponse : "Check out some curated topics on Supplements below or ask your own question",

      subPrompts : [

        "Weight Loss & Metabolism",

        "Sleep",

        "Longevity",

        "Hair & Skin Care",

        "Brain Health",

      ],

    },

      "How to consume snack foods consciously": {

        airesponse : "Check out some curated topics on Snack foods below or ask your own question",

        subPrompts : [

        "How to look out for  healthier snack options ",

        "'Healthy Foods' Red flags",

        "Trending healthy food  options",

      ],

    },

      "Brand Specfic Nutrition/Ingredient List": {

        airesponse : "Mention the specifc brand to konw thier nuteitonal content and ingredient list"

      }

 };

  // Handle sending user input to the server

  const handleSend = async (e) => {

    e.preventDefault(); // Prevent form submission

    //if (inputValue.trim() === '') return;

    // Append the user's message to the chat

    //const newMessages = [...messages, { sender: 'user', text: inputValue }];

    // const userMessage = inputValue;

    // setMessages(newMessages);

    setShowButtons(false);

    setMessages((prev) => [...prev, { sender: 'user', text: inputValue }]);

    //console.log("User input value:", inputValue);

    setInputValue(''); // Clear the input field

    //console.log("User input value:", inputValue);

    // try {

    //   // Send user input to the Flask server using axios

    //   // await axios.post('http://127.0.0.1:5001/ask', { inputValue }, {

    //   //   headers: { 'Content-Type': 'application/json' },

    //   // });

    controller = new AbortController();

      try {

        // Send user input to the Flask server using fetch

        const response = await fetch('http://127.0.0.1:5001/ask', {

          method: 'POST',

          headers: {

            'Content-Type': 'application/json', // Specify the content type

          },

          body: JSON.stringify({ question: inputValue }), // Convert data to JSON string

          signal: controller.signal,

        });

        // Check if the response is okay

        if (response.ok) {

          //console.log("Question sent successfully to the Flask server.");

          //listenToFlaskServer(inputValue); // Start listening to the server's streaming response

        } else {

          console.error("Error: Failed to send question to the server.", response.status);

        }

      const eventSource = new EventSource('http://127.0.0.1:5001/stream');

      let responseText = '';

      eventSource.onmessage = (event) => {

        responseText += event.data; // Collect the streaming response

        //console.log(responseText);

        setMessages((prev) =>

          prev.map((entry, index) =>

            index === prev.length - 1 ? { ...entry, text: responseText } : entry

          )

        );

      };

      eventSource.onerror = () => {

        eventSource.close();

      };

      // eventSource.onerror = (error) => {

      //   console.error('Error with EventSource:', error);

      //   eventSource.close();

      // };

        // Add an empty response entry to the chat history

      setMessages((prev) => [...prev, { sender: 'bot', text: '' }]);

      // Start listening to the server's streaming response

      //listenToFlaskServer();

    } catch (error) {

      if (error.name === 'AbortError') {

        console.log('Fetch request was canceled');

      }

      else {

        console.error('Error sending message to the Flask server:', error);

        eventSource.close();

      }

  };

};

  // Function to listen to the Flask server using EventSource

  // const listenToFlaskServer = () => {

  //   const eventSource = new EventSource(`http://127.0.0.1:5000/stream`);

  //   let responseText = '';

  //   eventSource.onmessage = (event) => {

  //     responseText += event.data;

  //     // Update the AI message in the chat area with the streamed response

  //     setMessages((prevMessages) => {

  //       const updatedMessages = [...prevMessages];

  //       const lastMessageIndex = updatedMessages.length - 1;

  //       if (updatedMessages[lastMessageIndex] && updatedMessages[lastMessageIndex].sender === 'ai') {

  //         updatedMessages[lastMessageIndex].text = responseText;

  //       } else {

  //         updatedMessages.push({ sender: 'ai', text: responseText });

  //       }

  //       return updatedMessages;

  //     });

  //   };

  const handleReset = () => {

    setChatHistories([...chatHistories, messages]);

    setMessages([]);

    setInputValue('');

    setShowButtons(true);

  };

  // const handleInputChange = (e) => {

  //   setInputValue(e.target.value);

  // };

  const handleHistoryClick = (history) => {

    setMessages(history);

    setShowButtons(false);

  };

  const handlePromptClick = (prompt) => {

    setInputValue(prompt);

    setShowButtons(false);

    console.log("Clicked prompt:", prompt); // Debugging check

    console.log("Sub-prompts for this prompt:", dynamicPrompts[prompt]?.subPrompts);

    // Add the selected main prompt to the chat area

    setMessages([...messages, { sender: 'user', text: prompt }]);

    // Display the specific AI response for the selected prompt

    //const aiResponse = dynamicPrompts[prompt].aiResponse;

    const aiResponse = dynamicPrompts[prompt]?.airesponse || 'No response available for this prompt';

    // Print debug information

    //console.log("AI Response for the prompt:", aiResponse); // Debug print statement

    setMessages((prevMessages) => [

      ...prevMessages,

      { sender: 'ai', text: aiResponse }

    ]);

    setInputValue('');

    // Display the corresponding sub-prompts

    //const relatedSubPrompts = dynamicPrompts[prompt]?.subPrompts || [];

    //setSubPrompts(relatedSubPrompts);

    //setSubPrompts(dynamicPrompts[prompt].subPrompts);

    setSelectedPrompt(prompt);

  };

  const handleSubPromptClick = (subPrompt) => {

    setMessages([...messages, { sender: 'user', text: subPrompt }]);

    setSubPrompts([]); // Clear sub-prompts after selection

  };

  return (

    <div className="chatbot-wrapper">

      {/\* Sidebar with chat history \*/}

      <div className="sidebar">

        <h2>Chat History</h2>

        <ul>

          {chatHistories.map((history, index) => (

            <li key={index} onClick={() => handleHistoryClick(history)}>

              {history.length > 0 ? history[0].text.substring(0, 20) + '...' : 'Empty Conversation'}

            </li>

          ))}

        </ul>

      </div>

      {/\* Chat area container \*/}

      <div className="chatbot-container">

        <div className="chat-area">

          {/\* Display dynamic buttons if enabled \*/}

          {showButtons && (

            <div className="dynamic-button-grid">

              {Object.keys(dynamicPrompts).map((prompt, index) => (

                <button

                  key={index}

                  className="dynamic-button"

                  onClick={() => handlePromptClick(prompt)}

                >

                  {prompt}

                </button>

              ))}

            </div>

          )}

          {/\* Display chat messages \*/}

          {messages.map((message, index) => (

            <div key={index} className="message-container">

              <Message sender={message.sender} text={message.text} />

              {/\* Render sub-prompt buttons right below the AI message \*/}

              {message.sender === 'ai' && dynamicPrompts[selectedPrompt]?.subPrompts && (

                <div className="sub-prompt-grid">

                  {dynamicPrompts[selectedPrompt].subPrompts.map((subPrompt, subIndex) => (

                    <button

                      key={subIndex}

                      className="sub-prompt-button"

                      onClick={() => handleSubPromptClick(subPrompt)}

                    >

                      {subPrompt}

                    </button>

                  ))}

                </div>

              )}

            </div>

          ))}

        </div>

        {/\* Input box for sending messages \*/}

        <form className="input-box" onSubmit={handleSend}>

          <input

            type="text"

            className="chat-input"

            value={inputValue}

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

            placeholder="Type your message..."

            required

          />

          <button type="submit" className="send-button">Send</button>

          <button type="button" className="reset-button" onClick={handleReset}>Reset</button>

        </form>

      </div>

    </div>

  );

};

export default Chatbot;