Advance AI chart bort:

- **React.js (JavaScript Library)**:
  - The core framework for building the user interface.
  - Uses **Hooks** like useState (for managing dynamic data like messages, input, loading states) and useEffect (for side effects like scrolling and initializing speech APIs), and useRef (for referencing DOM elements like the scrollable message area).
- **HTML/CSS (with Tailwind CSS)**:
  - **HTML**: Provides the structure of the chatbot interface (header, message area, input form, buttons).
  - **Tailwind CSS**: A utility-first CSS framework used for rapid and consistent styling, giving the app its modern, professional look. Custom CSS is also included for scrollbar styling and animations.
- **Google Gemini API Integration**:
  - This is how the chatbot interacts with powerful AI models.
  - **gemini-2.0-flash (for text-based AI)**: This Large Language Model (LLM) handles conversational responses, summarization, and prompt suggestions. It leverages advanced Natural Language Processing (NLP) techniques like text preprocessing (tokenization), and deep neural networks (NNLs) for contextual understanding.
  - **imagen-3.0-generate-002 (for image generation)**: This model takes text descriptions and generates AI-created images.
- **Web Speech API (JavaScript Browser API)**:
  - **Speech Recognition (SpeechRecognition)**: Converts spoken words from your microphone into text input for the chatbot ("Start Listening" button).
  - **Speech Synthesis (SpeechSynthesisUtterance)**: Converts the chatbot's text responses into spoken words ("Speak Response" button).
- **File Handling (Image Upload)**:
  - Allows users to upload image files from their device, which are converted to Base64 and sent to the Gemini AI for analysis and understanding, integrating visual input into the conversation.

(https://g.co/gemini/share/6119ae38f32a)or(https://g.co/gemini/share/6119ae38f32a)

(canvas to run it)

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

// Main App component for the AI Chatbot
const App = () => {
  // State to store the chat history (messages)
  const [messages, setMessages] = useState([]);
  // State to store the current input from the user
  const [input, setInput] = useState('');
  // State to manage the general loading status (when AI is thinking)
  const [isLoading, setIsLoading] = useState(false);
  // State to manage potential API errors
  const [error, setError] = useState(null);
  // State for selected image file (for sending to AI)
  const [selectedImage, setSelectedImage] = useState(null);
  // State to manage speech recognition (listening) status
  const [isListening, setIsListening] = useState(false);
  // State to manage text-to-speech (speaking) status
  const [isSpeaking, setIsSpeaking] = useState(false);
  // Ref for the messages container to enable auto-scrolling
  const messagesEndRef = useRef(null);

  // Reference for SpeechRecognition API instance
  const recognitionRef = useRef(null);

  // Initialize SpeechRecognition on component mount
  useEffect(() => {
    // Check if SpeechRecognition API is available
    const SpeechRecognition = window.SpeechRecognition ||
window.webkitSpeechRecognition;
    if (SpeechRecognition) {
      recognitionRef.current = new SpeechRecognition();
      recognitionRef.current.continuous = false; // Listen for a single utterance
      recognitionRef.current.interimResults = false; // Only return final results

      // Event handler for when speech is recognized
      recognitionRef.current.onresult = (event) => {
        const transcript = event.results[0][0].transcript;
        setInput(transcript); // Set the recognized speech as input
        console.log("Speech recognized:", transcript);
      };

      // Event handler for when speech recognition ends
      recognitionRef.current.onend = () => {
        setIsListening(false);
        console.log("Speech recognition ended.");
      };

      // Event handler for speech recognition errors
      recognitionRef.current.onerror = (event) => {
```

```
      console.error("Speech recognition error:", event.error);
      setError(`Speech recognition error: ${event.error}. Please ensure microphone access.`);
      setIsListening(false);
    };
  } else {
    console.warn("Web Speech API is not supported in this browser.");
    setError("Speech-to-text (microphone) is not supported in your browser.");
  }

  // Clean up function
  return () => {
    if (recognitionRef.current) {
      recognitionRef.current.onresult = null;
      recognitionRef.current.onend = null;
      recognitionRef.current.onerror = null;
      recognitionRef.current.stop(); // Ensure microphone is stopped
    }
  };
}, []); // Empty dependency array ensures this runs once on mount

// Scroll to the bottom of the chat whenever messages change
useEffect(() => {
  scrollToBottom();
}, [messages]);

/**
 * Function to scroll the chat window to the bottom.
 */
const scrollToBottom = () => {
  messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
};

/**
 * Converts a File object (image) to a base64 string.
 * @param {File} file - The image file to convert.
 * @returns {Promise<string>} A promise that resolves with the base64 string.
 */
const fileToBase64 = (file) => {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => resolve(reader.result.split(',')[1]); // Get only the base64 part
    reader.onerror = (error) => reject(error);
  });
};

/**
 * Handles sending a message to the AI, potentially including an image.
 * - Adds the user's message (and image) to chat history.
 * - Calls the Gemini API to get a response.
 * - Adds the AI's response to chat history.
```

```
   * @param {Event} e - The event object (e.g., form submission).
   */
  const handleSendMessage = async (e) => {
    e.preventDefault(); // Prevent default form submission behavior

    if (!input.trim() && !selectedImage) return; // Don't send empty messages or no
input/image

    setError(null);
    setIsLoading(true); // Show loading indicator immediately

    const currentUserInput = input; // Capture current input
    const currentSelectedImage = selectedImage; // Capture current selected image

    // Immediately update UI to show user's message and clear input
    setMessages((prevMessages) => [...prevMessages, {
      sender: 'user',
      text: currentUserInput,
      imageUrl: currentSelectedImage ? URL.createObjectURL(currentSelectedImage) : null
    }]);
    setInput(''); // Clear the input field
    setSelectedImage(null); // Clear the selected image preview

    const userMessageParts = [];
    if (currentUserInput.trim()) {
      userMessageParts.push({ text: currentUserInput });
    }
    let base64Image = null;
    if (currentSelectedImage) {
      try {
        base64Image = await fileToBase64(currentSelectedImage);
        userMessageParts.push({
          inlineData: {
            mimeType: currentSelectedImage.type,
            data: base64Image
          }
        });
      } catch (err) {
        // Revert message addition and show error if image processing fails
        setMessages((prevMessages) => prevMessages.slice(0, -1)); // Remove the last message
(the one that failed)
        setError("Failed to process image. Please try again.");
        setIsLoading(false);
        return;
      }
    }

    try {
      // Build chat history for the API payload. Note that previous image data
      // is not resent for context in this client-side example, only text.
      const apiChatHistory = messages.map(msg => {
```

```
      const parts = [];
      if (msg.text) parts.push({ text: msg.text });
      return { role: msg.sender === 'user' ? 'user' : 'model', parts: parts };
    });
    // Add the current user message (and its image) to the API history
    apiChatHistory.push({ role: 'user', parts: userMessageParts });

    const payload = {
      contents: apiChatHistory,
    };

    const apiKey = "";
    const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-
flash:generateContent?key=${apiKey}`;

    const response = await fetch(apiUrl, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload)
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`API error: ${response.status} ${response.statusText} -
${errorData.error.message}`);
    }

    const result = await response.json();

    let aiResponseText = "Sorry, I couldn't get a response. Please try again.";

    if (result.candidates && result.candidates.length > 0 &&
        result.candidates[0].content && result.candidates[0].content.parts &&
        result.candidates[0].content.parts.length > 0) {
      aiResponseText = result.candidates[0].content.parts[0].text;
    } else {
      console.error("Unexpected API response structure:", result);
    }

    const aiMessage = { sender: 'ai', text: aiResponseText };
    setMessages((prevMessages) => [...prevMessages, aiMessage]);

  } catch (error) {
    console.error("Error communicating with AI:", error);
    setError(`Failed to get response: ${error.message}. Please try again.`);
    const errorMessage = { sender: 'ai', text: `Error: ${error.message}. Please try again.` };
    setMessages((prevMessages) => [...prevMessages, errorMessage]);
  } finally {
    setIsLoading(false);
  }
};
```

```
/**
 * Handles file input change (for image selection).
 * @param {Event} e - The change event from the file input.
 */
const handleFileChange = (e) => {
  const file = e.target.files[0];
  if (file) {
    // Basic validation for image file type
    if (!file.type.startsWith('image/')) {
      setError("Only image files are supported.");
      setSelectedImage(null);
      return;
    }
    // Basic validation for file size (e.g., 5MB limit)
    if (file.size > 5 * 1024 * 1024) {
      setError("Image file is too large (max 5MB).");
      setSelectedImage(null);
      return;
    }
    setError(null);
    setSelectedImage(file);
  } else {
    setSelectedImage(null);
  }
};

/**
 * Toggles speech recognition (listening) on/off.
 */
const toggleListening = () => {
  if (!recognitionRef.current) {
    setError("Speech-to-text is not supported in your browser.");
    return;
  }
  if (isListening) {
    recognitionRef.current.stop();
  } else {
    setError(null);
    setInput(''); // Clear input before listening
    recognitionRef.current.start();
    setIsListening(true);
  }
};

/**
 * Initiates text-to-speech for the last AI message.
 */
const speakLastAiMessage = () => {
  if (!('speechSynthesis' in window)) {
    setError("Text-to-speech is not supported in your browser.");
```

```
      return;
    }

    const lastAiMessage = messages.slice().reverse().find(msg => msg.sender === 'ai');
    if (lastAiMessage && !isSpeaking) {
      setError(null);
      const utterance = new SpeechSynthesisUtterance(lastAiMessage.text);
      utterance.onstart = () => setIsSpeaking(true);
      utterance.onend = () => setIsSpeaking(false);
      utterance.onerror = (event) => {
        console.error("Speech synthesis error:", event);
        setError("Text-to-speech failed.");
        setIsSpeaking(false);
      };
      window.speechSynthesis.speak(utterance);
    } else if (isSpeaking) {
      window.speechSynthesis.cancel(); // Stop current speech
      setIsSpeaking(false);
    }
  };

  /**
   * Handles summarizing the current chat conversation using the Gemini API.
   * This also leverages the Natural Language Processing (NLP) capabilities
   * of the underlying LLM to understand and condense the conversation.
   */
  const handleSummarizeChat = async () => {
    if (messages.length === 0) return;

    setIsLoading(true);
    setError(null); // Clear previous errors
    const conversationText = messages.map(msg => `${msg.sender === 'user' ? 'User' : 'AI'}:
${msg.text}`).join('\n');
    const prompt = `Summarize the following conversation
concisely:\n\n${conversationText}`;

    try {
      const payload = {
        contents: [{ role: 'user', parts: [{ text: prompt }] }],
      };

      const apiKey = "";
      const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-
flash:generateContent?key=${apiKey}`;

      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });
```

```
      if (!response.ok) {
        const errorData = await response.json();
        throw new Error(`API error: ${response.status} ${response.statusText} -
${errorData.error.message}`);
      }

      const result = await response.json();

      let summaryText = "Could not generate summary. Please try again.";
      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content && result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        summaryText = `✨ Chat Summary: ${result.candidates[0].content.parts[0].text}`;
      } else {
        console.error("Unexpected API response structure for summary:", result);
      }

      setMessages((prevMessages) => [...prevMessages, { sender: 'ai', text: summaryText }]);
    } catch (error) {
      console.error("Error summarizing chat:", error);
      setError(`Failed to summarize chat: ${error.message}`);
      setMessages((prevMessages) => [...prevMessages, { sender: 'ai', text: `Error summarizing
chat: ${error.message}` }]);
    } finally {
      setIsLoading(false);
    }
  };

  /**
   * Handles suggesting follow-up questions based on the current chat context using the
Gemini API.
   * This feature also relies heavily on the LLM's understanding of conversation flow and
semantic relevance.
   */
  const handleSuggestPrompts = async () => {
    if (messages.length === 0) return;

    setIsLoading(true);
    setError(null); // Clear previous errors
    // Take the last 5 messages for context
    const recentConversation = messages.slice(-5).map(msg => `${msg.sender === 'user' ?
'User' : 'AI'}: ${msg.text}`).join('\n');
    const prompt = `Based on the following recent conversation, suggest 3 concise and distinct
follow-up questions the user might ask. Format them as a numbered list.
:\n\n${recentConversation}`;

    try {
      const payload = {
        contents: [{ role: 'user', parts: [{ text: prompt }] }],
      };
```

```
    const apiKey = "";
    const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${apiKey}`;

    const response = await fetch(apiUrl, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload)
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(`API error: ${response.status} ${response.statusText} - ${errorData.error.message}`);
    }

    const result = await response.json();

    let suggestionsText = "Could not suggest prompts. Please try again.";
    if (result.candidates && result.candidates.length > 0 &&
        result.candidates[0].content && result.candidates[0].content.parts &&
        result.candidates[0].content.parts.length > 0) {
      suggestionsText = `✨ Here are some questions you might ask:\n${result.candidates[0].content.parts[0].text}`;
    } else {
      console.error("Unexpected API response structure for suggestions:", result);
    }

    setMessages((prevMessages) => [...prevMessages, { sender: 'ai', text: suggestionsText }]);
  } catch (error) {
    console.error("Error suggesting prompts:", error);
    setError(`Failed to suggest prompts: ${error.message}`);
    setMessages((prevMessages) => [...prevMessages, { sender: 'ai', text: `Error suggesting prompts: ${error.message}` }]);
  } finally {
    setIsLoading(false);
  }
 };

 /**
  * Handles generating a visual aid (text-based figure) from the last AI message.
  * This function sends the last AI response to the LLM and asks it to present
  * that information in a structured, "figure-like" textual format (e.g., table, steps).
  */
 const handleGenerateVisualAid = async () => {
  // Only generate a visual aid if there's at least one AI message
  if (messages.length === 0 || messages[messages.length - 1].sender === 'user') {
    setError("Please wait for an AI response before generating a visual aid.");
    return;
  }
```

```
    setIsLoading(true);
    setError(null); // Clear previous errors
    const lastAiMessage = messages[messages.length - 1].text;
    const prompt = `Based on the following text, provide a concise and clear visual aid. This could be a table, a numbered list of key steps/points, or a textual description of a simple diagram. Focus on clarity and essential information. Format it using Markdown:\n\n"${lastAiMessage}"`;

    try {
      const payload = {
        contents: [{ role: 'user', parts: [{ text: prompt }] }],
      };

      const apiKey = "";
      const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${apiKey}`;

      const response = await fetch(apiUrl, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(payload)
      });

      if (!response.ok) {
        const errorData = await response.json();
        throw new Error(`API error: ${response.status} ${response.statusText} - ${errorData.error.message}`);
      }

      const result = await response.json();

      let visualAidText = "Could not generate visual aid. Please try again.";
      if (result.candidates && result.candidates.length > 0 &&
          result.candidates[0].content && result.candidates[0].content.parts &&
          result.candidates[0].content.parts.length > 0) {
        visualAidText = `📊 Visual Aid:\n${result.candidates[0].content.parts[0].text}`;
      } else {
        console.error("Unexpected API response structure for visual aid:", result);
      }

      setMessages((prevMessages) => [...prevMessages, { sender: 'ai', text: visualAidText }]);
    } catch (error) {
      console.error("Error generating visual aid:", error);
      setError(`Failed to generate visual aid: ${error.message}`);
      setMessages((prevMessages) => [...prevMessages, { sender: 'ai', text: `Error generating visual aid: ${error.message}` }]);
    } finally {
      setIsLoading(false);
    }
  };
```

```
 /**
  * Handles generating an AI image based on the current input text.
  */
 const handleGenerateImage = async () => {
  if (!input.trim()) {
   setError("Please enter a description for the image you want to generate.");
   return;
  }

  setIsLoading(true);
  setError(null);
  const imagePrompt = input.trim();
  setMessages((prevMessages) => [...prevMessages, { sender: 'user', text: `Generate image:
"${imagePrompt}"` }]);
  setInput(''); // Clear the input field

  try {
   // Payload for image generation model
   const payload = { instances: { prompt: imagePrompt }, parameters: { "sampleCount": 1} };
   const apiKey = ""; // Canvas will inject at runtime
   const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/imagen-3.0-
generate-002:predict?key=${apiKey}`;

   const response = await fetch(apiUrl, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload)
   });

   if (!response.ok) {
    const errorData = await response.json();
    throw new Error(`Image API error: ${response.status} ${response.statusText} -
${errorData.error.message}`);
   }

   const result = await response.json();

   if (result.predictions && result.predictions.length > 0 &&
result.predictions[0].bytesBase64Encoded) {
    const imageUrl =
`${result.predictions[0].bytesBase64Encoded}`;
    const aiMessage = { sender: 'ai', text: `Here is your generated image:`, imageUrl:
imageUrl };
    setMessages((prevMessages) => [...prevMessages, aiMessage]);
   } else {
    console.error("Unexpected image API response structure:", result);
    const errorMessage = { sender: 'ai', text: "Could not generate image. Please try a
different prompt." };
    setMessages((prevMessages) => [...prevMessages, errorMessage]);
   }
```

```
    } catch (error) {
      console.error("Error generating image:", error);
      setError(`Failed to generate image: ${error.message}.`);
      const errorMessage = { sender: 'ai', text: `Error generating image: ${error.message}.
Please try again.` };
      setMessages((prevMessages) => [...prevMessages, errorMessage]);
    } finally {
      setIsLoading(false);
    }
  };

  /**
   * Handles clearing all messages from the chat history.
   */
  const handleClearChat = () => {
    setMessages([]);
    setError(null); // Clear any existing errors
    setInput(''); // Clear input field
    setSelectedImage(null); // Clear selected image
    setIsLoading(false); // Ensure loading is off
    setIsListening(false); // Ensure listening is off
    setIsSpeaking(false); // Ensure speaking is off
    if (window.speechSynthesis) {
      window.speechSynthesis.cancel(); // Stop any ongoing speech
    }
    if (recognitionRef.current) {
      recognitionRef.current.stop(); // Stop any ongoing listening
    }
  };

  return (
    // Main container with full height and flex layout for chat interface
    <div className="flex flex-col h-screen overflow-hidden text-gray-100 unique-
background">
      {/* Header section */}
      <header className="unique-header text-white p-4 shadow-lg border-b border-purple-
700 relative z-10">
        <h1 className="text-4xl font-extrabold text-center mb-4 unique-header-text">
          <span className="sparkle-text">S</span>hawnik AI Chatbot
        </h1>
        <div className="flex flex-wrap justify-center gap-3">
          {messages.length > 0 && (
            <button
              onClick={handleSummarizeChat}
              className="unique-button action-button bg-blue-700 hover:bg-blue-600"
              disabled={isLoading}
            >
              <svg className="w-4 h-4" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M4 4a2 2 0 00-2 2v8a2
2 0 002 2h12a2 2 0 002-2V6a2 2 0 00-2-2H4zm10 2a1 1 0 00-1 1v4a1 1 0 001 1h2a1 1 0 001-
1V7a1 1 0 00-1-1h-2z" clipRule="evenodd"></path></svg>
```

```
          Summarize
        </button>
      )}
      {messages.length > 0 && (
        <button
          onClick={handleSuggestPrompts}
          className="unique-button action-button bg-green-700 hover:bg-green-600"
          disabled={isLoading}
        >
          <svg className="w-4 h-4" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M10 18a8 8 0 100-16 8
8 0 000 16zm-7-8a1 1 0 011-1h2a1 1 0 011 1v2a1 1 0 01-1 1H4a1 1 0 01-1-1v-2zm7 0a1 1 0
011-1h2a1 1 0 011 1v2a1 1 0 01-1 1h-2a1 1 0 01-1-1v-2zm7 0a1 1 0 011-1h2a1 1 0 011 1v2a1
1 0 01-1 1h-2a1 1 0 01-1-1v-2z" clipRule="evenodd"></path></svg>
          Suggest
        </button>
      )}
      {messages.length > 0 && (
        <button
          onClick={handleGenerateVisualAid}
          className="unique-button action-button bg-yellow-700 hover:bg-yellow-600"
          disabled={isLoading}
        >
          <svg className="w-4 h-4" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M11 3a1 1 0 100
2h2.09l3.426 6.852A.999.999 0 0017 13h-2v4a1 1 0 102 0v-4a1 1 0 00-.293-.707l-3.535-
3.536A1 1 0 0012 7V5a1 1 0 10-2 0v2a1 1 0 00.293.707l3.535 3.536A1 1 0 0015
13h2V9.148L13.906 5H11zM7 9a1 1 0 00-1 1v4a1 1 0 102 0v-4a1 1 0 00-1-1z"
clipRule="evenodd"></path></svg>
          Visual Aid
        </button>
      )}
      <button
        onClick={handleGenerateImage}
        className="unique-button action-button bg-purple-700 hover:bg-purple-600"
        disabled={isLoading || !input.trim()}
        title="Generate an AI Image from your text"
      >
        <svg className="w-4 h-4" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M4 3a2 2 0 00-2
2v10a2 2 0 002 2h12a2 2 0 002-2V5a2 2 0 00-2-2H4zm12 12H4l4-4 4 4 4-4V5h-2v3.586l-
1.707-1.707A1 1 0 0010.293 6.707L8.586 8.414 7 6.707a1 1 0 00-1.414 1.414l2.707 2.707L8
13.586l1.707 1.707A1 1 0 0012 15h4z" clipRule="evenodd"></path></svg>
        Generate Image
      </button>
      {messages.length > 0 && (
        <button
          onClick={handleClearChat}
          className="unique-button action-button bg-red-700 hover:bg-red-600"
          disabled={isLoading}
        >
```

```
          <svg className="w-4 h-4" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M9 2a1 1 0 00-
.894.553L7.382 4H4a1 1 0 000 2v10a2 2 0 002 2h8a2 2 0 002-2V6a1 1 0 100-2h-3.382l-.724-
1.447A1 1 0 0011 2H9zM7 8a1 1 0 012 0v6a1 1 0 11-2 0V8zm4 0a1 1 0 112 0v6a1 1 0 11-2
0V8z" clipRule="evenodd"></path></svg>
           Clear Chat
          </button>
         )}
        </div>
      </header>

      {/* Error message display */}
      {error && (
       <div className="bg-red-700 text-white p-3 text-center text-sm font-medium rounded-
b-lg shadow-md animate-fade-in z-20 relative">
         {error}
       </div>
      )}

      {/* Chat messages display area */}
      <main className="flex-1 overflow-y-auto p-6 space-y-4 relative z-0">
       {messages.length === 0 && (
        <div className="flex flex-col items-center justify-center h-full text-gray-400">
         <svg
          className="w-20 h-20 text-indigo-500 mb-6 opacity-75 animate-bounce-slow"
          fill="none"
          stroke="currentColor"
          viewBox="0 0 24 24"
          xmlns="http://www.w3.org/2000/svg"
         >
          <path
           strokeLinecap="round"
           strokeLinejoin="round"
           strokeWidth="1.5"
           d="M8 12h.01M12 12h.01M16 12h.01M21 12c0 4.418-4.03 8-9 8a9.863 9.863 0 01-
4.255-.949L3 20l1.395-3.72C3.512 15.042 3 13.574 3 12c0-4.418 4.03-8 9-8s9 3.582 9 8z"
          ></path>
         </svg>
         <p className="text-xl font-semibold mb-2 text-indigo-200">Start a
conversation!</p>
         <p className="text-md text-center max-w-sm text-indigo-300">
          Ask me anything, even if it's incomplete or has typos. I'm here to help.
         </p>
        </div>
       )}

       {messages.map((msg, index) => (
        <div
         key={index}
         className={`flex ${msg.sender === 'user' ? 'justify-end' : 'justify-start'}`}
        >
```

```
          <div
            className={`max-w-3xl p-3 shadow-lg animate-fade-in unique-message-bubble ${
              msg.sender === 'user'
                ? 'user-message-bg'
                : 'ai-message-bg'
            }`}
          >
            {msg.imageUrl && (
              <img
                src={msg.imageUrl}
                alt={msg.sender === 'user' ? "User upload" : "AI generated image"}
                className="max-w-full h-auto rounded-lg mb-2 border border-gray-600"
                style={{ maxHeight: '200px' }} // Limit image display size
                onError={(e) => { e.target.onerror = null; e.target.src =
'https://placehold.co/100x100/333/FFF?text=Image+Error'; }}
              />
            )}
            <p className="whitespace-pre-wrap leading-relaxed">{msg.text}</p>
          </div>
        </div>
      ))}
      {isLoading && (
        <div className="flex justify-start">
          <div className="max-w-xl p-3 shadow-lg ai-message-bg unique-message-bubble">
            <div className="flex space-x-1">
              <div className="w-2.5 h-2.5 bg-gray-400 rounded-full animate-pulse-slow"></div>
              <div className="w-2.5 h-2.5 bg-gray-400 rounded-full animate-pulse-slow delay-
150"></div>
              <div className="w-2.5 h-2.5 bg-gray-400 rounded-full animate-pulse-slow delay-
300"></div>
            </div>
          </div>
        </div>
      )}
      <div ref={messagesEndRef} /> {/* Element to scroll to */}
    </main>

    {/* Input form for sending messages */}
    <form onSubmit={handleSendMessage} className="p-4 relative z-20">
      <div className="flex items-center space-x-3 bg-gray-900 bg-opacity-70 backdrop-blur-
sm rounded-xl p-3 shadow-2xl border border-gray-700">
        {/* File Input for Images */}
        <label htmlFor="file-upload" className="cursor-pointer flex items-center p-2.5
rounded-full shadow-lg transition duration-200 transform hover:scale-105 focus:outline-
none focus:ring-2 focus:ring-green-500 focus:ring-offset-2 focus:ring-offset-gray-900
${selectedImage ? 'bg-green-600' : 'bg-gray-700 hover:bg-gray-600'}">
          <input id="file-upload" type="file" accept="image/*" onChange={handleFileChange}
className="hidden" disabled={isLoading} />
          <svg className="w-6 h-6 text-white" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M4 3a2 2 0 00-2
2v10a2 2 0 002 2h12a2 2 0 002-2V5a2 2 0 00-2-2H4zm12 12H4l4-4 4 4 4-4V5h-2v3.586l-
```

```
1.707-1.707A1 1 0 0010.293 6.707L8.586 8.414 7 6.707a1 1 0 00-1.414 1.414l2.707 2.707L8
13.586l1.707 1.707A1 1 0 0012 15h4z" clipRule="evenodd"></path></svg>
          <span className="hidden sm:inline ml-1 text-sm font-medium">Add File</span>
        </label>

        {/* Text Input */}
        <input
          type="text"
          value={input}
          onChange={(e) => setInput(e.target.value)}
          placeholder={selectedImage ? "Add a message for your image..." : "Type your
message here..."}
          className="flex-1 p-2.5 bg-gray-800 text-gray-100 border border-gray-600 rounded-
lg focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-transparent
placeholder-gray-400 text-base transition duration-200"
          disabled={isLoading}
        />

        {/* Speak Response Button */}
        {messages.length > 0 && messages.some(msg => msg.sender === 'ai') && (
          <button
            type="button"
            onClick={speakLastAiMessage}
            className={`p-2.5 rounded-full shadow-lg transition duration-200 transform
hover:scale-105 focus:outline-none focus:ring-2 focus:ring-pink-500 focus:ring-offset-2
focus:ring-offset-gray-900 ${isSpeaking ? 'bg-pink-600 animate-pulse-slow' : 'bg-gray-700
hover:bg-gray-600'} disabled:opacity-50 disabled:cursor-not-allowed`}
            disabled={isLoading}
            title={isSpeaking ? "Stop Speaking" : "Speak Last Response"}
          >
            <svg className="w-6 h-6 text-white" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M9.383 3.076A1 1 0
0110 4v12a1 1 0 01-1.707.707L4.586 13H2a1 1 0 01-1-1V8a1 1 0 011-1h2.586l3.707-3.707a1
1 0 011.09-.217zM14.004 7.583a1 1 0 10-1.416-1.416A5 5 0 0116 10c0 1.916-.76 3.65-2.004
4.917a1 1 0 001.416 1.416A7 7 0 0018 10a7 7 0 00-3.996-2.417z"
clipRule="evenodd"></path></svg>
          </button>
        )}

        {/* Start Listening Button */}
        <button
          type="button"
          onClick={toggleListening}
          className={`p-2.5 rounded-full shadow-lg transition duration-200 transform
hover:scale-105 focus:outline-none focus:ring-2 focus:ring-teal-500 focus:ring-offset-2
focus:ring-offset-gray-900 ${isListening ? 'bg-teal-600 animate-pulse-slow' : 'bg-gray-700
hover:bg-gray-600'} disabled:opacity-50 disabled:cursor-not-allowed`}
          disabled={isLoading || !recognitionRef.current}
          title={isListening ? "Stop Listening" : "Start Listening"}
        >
          <svg className="w-6 h-6 text-white" fill="currentColor" viewBox="0 0 20 20"
```

```
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M7 4a3 3 0 016 0v4a3
3 0 11-6 0V4zm4 10.602V17a1 1 0 11-2 0v-2.398l-4.593 4.593a1 1 0 01-1.414-1.414L7.586
12H4a1 1 0 110-2h3.586L4 6.414A1 1 0 015.414 5L10 9.586l4.593-4.593a1 1 0 111.414
1.414L12.414 10H16a1 1 0 110 2h-3.586l4.593 4.593a1 1 0 01-1.414 1.414L10 14.414z"
clipRule="evenodd"></path></svg>
        </button>

        {/* Send Button */}
        <button
          type="submit"
          className="p-2.5 bg-gradient-to-r from-purple-600 to-pink-600 text-white rounded-
full shadow-lg hover:from-purple-700 hover:to-pink-700 focus:outline-none focus:ring-2
focus:ring-purple-500 focus:ring-offset-2 focus:ring-offset-gray-900 transition duration-200
transform hover:scale-105 disabled:opacity-50 disabled:cursor-not-allowed"
          disabled={isLoading || (!input.trim() && !selectedImage)}
        >
          <svg
            className="w-6 h-6"
            fill="none"
            stroke="currentColor"
            viewBox="0 0 24 24"
            xmlns="http://www.w3.org/2000/svg"
          >
            <path
              strokeLinecap="round"
              strokeLinejoin="round"
              strokeWidth="2"
              d="M14 5l7 7m0 0l-7 7m7-7H3"
            ></path>
          </svg>
        </button>
      </div>
      {selectedImage && (
        <div className="flex items-center mt-2 p-2 bg-gray-700 rounded-lg border border-
gray-600 text-sm text-gray-300">
          <span className="truncate">{selectedImage.name}</span>
          <button
            type="button"
            onClick={() => setSelectedImage(null)}
            className="ml-2 text-red-400 hover:text-red-600 focus:outline-none"
            title="Remove image"
          >
            <svg className="w-4 h-4" fill="currentColor" viewBox="0 0 20 20"
xmlns="http://www.w3.org/2000/svg"><path fillRule="evenodd" d="M4.293 4.293a1 1 0
011.414 0L10 8.586l4.293-4.293a1 1 0 111.414 1.414L11.414 10l4.293 4.293a1 1 0 01-1.414
1.414L10 11.414l-4.293 4.293a1 1 0 01-1.414-1.414L8.586 10 4.293 5.707a1 1 0 010-1.414z"
clipRule="evenodd"></path></svg>
          </button>
        </div>
      )}
    </form>
```

```
    {/* Tailwind CSS Script - Must be at the end of body for JIT mode */}
    <script src="https://cdn.tailwindcss.com"></script>
    {/* Custom Styles for unique design elements and animations */}
    <style jsx>{`
      @import
url('https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&display=swap');
      @import
url('https://fonts.googleapis.com/css2?family=Share+Tech+Mono&display=swap');

      /* Apply unique fonts */
      body {
        font-family: 'Share Tech Mono', monospace; /* Techy, readable font */
      }

      .unique-header-text {
        font-family: 'Orbitron', sans-serif; /* Sci-fi / futuristic font */
        letter-spacing: 2px;
        text-shadow: 0 0 8px #a78bfa, 0 0 15px #c084fc;
      }

      .sparkle-text {
        color: #a78bfa; /* Purple hue */
        text-shadow: 0 0 5px #a78bfa, 0 0 10px #c084fc;
        animation: glow 1.5s infinite alternate;
      }

      @keyframes glow {
        from { text-shadow: 0 0 5px #a78bfa, 0 0 10px #c084fc; }
        to { text-shadow: 0 0 10px #8b5cf6, 0 0 20px #d8b4fe; }
      }

      /* Unique Background */
      .unique-background {
        background: linear-gradient(135deg, #0a0a0a 0%, #1a1a2e 50%, #0a0a0a 100%);
        animation: gradient-shift 15s ease infinite;
      }

      @keyframes gradient-shift {
        0% { background-position: 0% 50%; }
        50% { background-position: 100% 50%; }
        100% { background-position: 0% 50%; }
      }

      /* Unique Header Style */
      .unique-header {
        background: linear-gradient(90deg, #1f012c, #3a005c, #1f012c); /* Darker, richer
purple gradient */
        border-bottom: 3px solid #8a2be2; /* Blue-violet border */
        box-shadow: 0 5px 15px rgba(0, 0, 0, 0.6);
        padding: 1.5rem; /* More padding */
```

```
    }

    /* Unique Button Styling */
    .unique-button {
      padding: 0.6rem 1.2rem;
      border-radius: 9999px; /* Pill shape */
      font-weight: 600;
      box-shadow: 0 4px 10px rgba(0, 0, 0, 0.4);
      transition: all 0.3s ease-in-out;
      border: 1px solid rgba(255, 255, 255, 0.2); /* Subtle border */
    }

    .unique-button:hover {
      transform: translateY(-2px) scale(1.02);
      box-shadow: 0 6px 15px rgba(0, 0, 0, 0.5);
    }

    .unique-button:active {
      transform: translateY(0) scale(0.98);
      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);
    }

    .action-button {
       background-color: #334155; /* Neutral background for action buttons */
       color: #d1d5db; /* Lighter text */
    }
    .action-button:hover {
       background-color: #475569;
    }

    /* Unique Message Bubbles */
    .unique-message-bubble {
      border-radius: 1.5rem; /* More rounded */
      position: relative;
      overflow: hidden;
    }

    .user-message-bg {
      background: linear-gradient(45deg, #6a05ad, #3e0b8e); /* Deep violet gradient */
      color: white;
      border-top-right-radius: 0.5rem; /* Slightly pointed */
    }

    .ai-message-bg {
      background: linear-gradient(135deg, #2a2a3a, #1a1a2a); /* Darker, subtle gradient for
AI */
      color: #e0e0e0;
      border-top-left-radius: 0.5rem; /* Slightly pointed */
      border: 1px solid #4a148c; /* Purple border for AI messages */
    }
```

```
/* Loading indicator pulse */
@keyframes pulse-slow {
  0%, 100% { opacity: 0.7; }
  50% { opacity: 1; }
}
.animate-pulse-slow {
  animation: pulse-slow 1.5s infinite ease-in-out;
}
.delay-150 { animation-delay: 0.15s; }
.delay-300 { animation-delay: 0.3s; }

/* Fade-in animation for messages */
@keyframes fade-in {
  from { opacity: 0; transform: translateY(5px); }
  to { opacity: 1; transform: translateY(0); }
}
.animate-fade-in {
  animation: fade-in 0.2s ease-out;
}

/* Unique input area styling */
form > div {
    background-color: rgba(10, 10, 10, 0.8); /* Darker, more transparent */
    border: 1px solid #4a148c; /* Matching purple border */
    border-radius: 1.5rem; /* More rounded */
}

input[type="text"] {
    background-color: #1a1a1a; /* Even darker input field */
    border: 1px solid #333;
    color: #e0e0e0;
    padding-left: 1.25rem;
    padding-right: 1.25rem;
}

.input-button {
    padding: 0.75rem; /* Larger buttons */
    border-radius: 50%; /* Circular buttons */
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.5);
}

.input-button:hover {
    transform: scale(1.1);
}

/* Bounce animation for initial message icon */
@keyframes bounce-slow {
  0%, 100% {
    transform: translateY(0);
  }
  50% {
```

```
          transform: translateY(-10px);
        }
      }
      .animate-bounce-slow {
        animation: bounce-slow 2s infinite ease-in-out;
      }
    `}</style>
  </div>
 );
};

export default App;
```