NewPrompt.jsx

import { useEffect, useRef, useState } from "react";

import "./newPrompt.css";

import Upload from "../upload/Upload";

import { IKImage } from "imagekitio-react";

import model from "../../lib/gemini";

import Markdown from "react-markdown";

import { useMutation, useQueryClient } from "@tanstack/react-query";

import axios from "axios";

const NewPrompt = ({ data }) => {

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

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

  const [loading, setLoading] = useState(false);

  const [img, setImg] = useState({

    isLoading: false,

    error: "",

    dbData: {},

    aiData: {},

  });

  const endRef = useRef(null);

  const formRef = useRef(null);

  const queryClient = useQueryClient();

  const mutation = useMutation({

    mutationFn: async () => {

      console.log("🔄 Updating chat history:", { question, answer, img });

      return axios.put(

        `${import.meta.env.VITE\_API\_URL}/api/chats/${data.\_id}`,

        {

          question: question.length ? question : undefined,

          answer,

          img: img.dbData?.filePath ? { filePath: img.dbData.filePath } : undefined,

        },

        { withCredentials: true }

      );

    },

    onSuccess: () => {

      console.log("✅ Chat history updated!");

      queryClient.invalidateQueries({ queryKey: ["chat", data.\_id] });

      // 🔹 Immediately update local UI instead of waiting for re-fetch

      data.history.push({ role: "user", parts: [{ text: question }] });

      data.history.push({ role: "model", parts: [{ text: answer }] });

      formRef.current.reset();

      setQuestion("");

      setAnswer("");

      setImg({ isLoading: false, error: "", dbData: {}, aiData: {} });

    },

  });

  useEffect(() => {

    endRef.current?.scrollIntoView({ behavior: "smooth" });

  }, [data, question, answer, img.dbData]);

  // 🔹 Enhanced Weather Query Detection

  const isWeatherQuery = (text) => {

    const keywords = ["weather", "temperature", "forecast", "humidity", "wind"];

    return keywords.some((keyword) => text.toLowerCase().includes(keyword));

  };

  // 🔹 Improved Location Extraction

  const extractLocation = (text) => {

    const words = text.toLowerCase().split(" ");

    const stopWords = [

      "weather", "temperature", "forecast", "humidity", "wind",

      "is", "the", "in", "what", "how", "whats", "was", "will", "be",

      "yesterday", "today", "tomorrow", "next", "day", "week", "month", "sunny", "cloudy"

    ];

    // Filter out stopwords and join the rest as the location

    const filteredWords = words.filter(word => !stopWords.includes(word));

    // Capitalize first letter of each word (for correct formatting)

    const location = filteredWords.map(word => word.charAt(0).toUpperCase() + word.slice(1)).join(" ");

    console.log("🔍 Corrected Extracted Location:", location); // Debugging log

    return location || null;

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    const text = e.target.text.value.trim();

    if (!text) return;

    setQuestion(text);

    setLoading(true);

    console.log("🟢 User input received:", text);

    // Weather Query Handling

    if (isWeatherQuery(text)) {

      console.log("🌤️ Detected a weather-related query. Processing...");

      let location = extractLocation(text);

      if (location) {

        location = location.charAt(0).toUpperCase() + location.slice(1);

        console.log("🌍 Extracted location:", location);

        try {

          const weatherRes = await axios.get(

            `${import.meta.env.VITE\_API\_URL}/api/weather/${encodeURIComponent(location)}`,

            { withCredentials: true }

          );

          console.log("✅ Weather API Request:", `${import.meta.env.VITE\_API\_URL}/api/weather/${encodeURIComponent(location)}`); // Debugging log

          console.log("✅ Weather API Response:", weatherRes.data);

          if (!location || location.trim() === "") {

            setAnswer("❌ I couldn't detect a valid location. Please specify a city name.");

            setLoading(false);

            return;

          }

          const weatherInfo = weatherRes.data;

          const weatherText = `🌤️ The current weather in \*\*${weatherInfo.location}\*\* is \*\*${weatherInfo.weather}\*\*.

            - 🌡️ Temperature: \*\*${weatherInfo.temperature}°C\*\*

            - 💧 Humidity: \*\*${weatherInfo.humidity}%\*\*

            - 💨 Wind Speed: \*\*${weatherInfo.wind\_speed} m/s\*\*`;

          console.log("📩 Before setting answer:", answer);

          setAnswer(weatherText);

          await new Promise((resolve) => setTimeout(resolve, 100)); // Ensures state updates

          console.log("📩 After setting answer:", answer);  // Confirm that answer is set

          await mutation.mutateAsync();  // Wait for chat update

          setLoading(false);

          return;

        } catch (error) {

          console.error("❌ Weather API error:", error.response?.data || error.message);

          setAnswer("⚠️ Unable to fetch the weather. Please try again later.");

          setLoading(false);

          return;

        }

      } else {

        console.log("❌ No valid location found in query. Sending to AI model.");

      }

    }

    console.log("🔵 Not a weather query. Sending to AI model...");

    try {

      const chat = model.startChat({

        history: data?.history?.map((message) => ({

          role: message.role,

          parts: [{ text: message.parts[0].text }],

        })) || [],

        generationConfig: {},

      });

      const input = [

        ...data?.history?.map((message) => message.parts[0].text) || [],

        img.dbData?.filePath ? [img.dbData, text] : text,

      ];

      const result = await chat.sendMessageStream(input);

      let accumulatedText = "";

      for await (const chunk of result.stream) {

        accumulatedText += chunk.text();

        console.log("📝 AI Response Chunk:", chunk.text());

        // If AI asks for an image, log the request

        if (chunk.text().toLowerCase().includes("please provide an image")) {

          console.warn("⚠️ AI is asking for an image, but none was detected.");

        }

        setAnswer((prev) => prev + chunk.text());

      }

      // Ensure the final AI answer is set before updating chat history

      await new Promise((resolve) => setTimeout(resolve, 100));

      console.log("✅ Final AI answer:", accumulatedText);

      // Force re-render by calling setAnswer again

      setAnswer(accumulatedText);

      // Ensure chat history is updated before UI refresh

      data.history.push({ role: "user", parts: [{ text: question }] });

      data.history.push({ role: "model", parts: [{ text: answer }] });

      await mutation.mutateAsync();

      setLoading(false);

    } catch (err) {

      console.error("❌ AI model error:", err);

      setAnswer("⚠️ An error occurred, please try again.");

    } finally {

      setLoading(false);

    }

    formRef.current.reset();

  };

  return (

    <>

      {loading && <div>⏳ Processing your request...</div>}

      {img.isLoading && <div>⏳ Uploading image...</div>}

      {img.dbData?.filePath && (

        <IKImage

          urlEndpoint={import.meta.env.VITE\_IMAGE\_KIT\_ENDPOINT}

          path={img.dbData?.filePath}

          width="380"

          transformation={[{ width: 380 }]}

        />

      )}

      {question && <div className="message user">{question}</div>}

      {answer && <div className="message"><Markdown>{answer}</Markdown></div>}

      <div className="endChat" ref={endRef}></div>

      <form className="newForm" onSubmit={handleSubmit} ref={formRef}>

        <Upload setImg={setImg} />

        <input type="text" name="text" placeholder="Ask me anything..." required />

        <button type="submit">

          <img src="/arrow.png" alt="Send" />

        </button>

      </form>

    </>

  );

};

export default NewPrompt;

Upload.jsx ----------

import { IKContext, IKUpload } from 'imagekitio-react';

import { useRef } from 'react';

const urlEndpoint = import.meta.env.VITE\_IMAGE\_KIT\_ENDPOINT;

const publicKey = import.meta.env.VITE\_IMAGE\_KIT\_PUBLIC\_KEY;

const authenticator =  async () => {

    try {

        const response = await fetch('http://localhost:3000/api/upload');

        if (!response.ok) {

            const errorText = await response.text();

            throw new Error(`Request failed with status ${response.status}: ${errorText}`);

        }

        const data = await response.json();

        const { signature, expire, token } = data;

        return { signature, expire, token };

    } catch (error) {

        throw new Error(`Authentication request failed: ${error.message}`);

    }

};

const Upload  = ({setImg}) => {

  const ikUploadRef =useRef(null)

    const onError = err => {

        console.log("Error", err);

      };

      const onSuccess = res => {

        console.log("Success", res);

        setImg(prev=>({...prev, isLoading: false, dbData:res}))

      };

      const onUploadProgress = progress => {

        console.log("Progress", progress);

      };

      const onUploadStart = evt => {

        const file = evt.target.files[0];

        if (!file) {

            console.error("❌ No file selected!");

            return;

        }

        console.log("📤 Uploading file:", file);

        console.log("📄 File type:", file.type);

        const reader = new FileReader();

        reader.onloadend = () => {

            console.log("✅ File read successfully!");

            console.log("🖼️ Base64 Data (first 100 chars):", reader.result.substring(0, 100));

            setImg(prev => ({

                ...prev,

                isLoading: true,

                aiData: {

                    inlineData: {

                        data: reader.result.split(",")[1], // 🔹 Ensure correct format

                        mimeType: file.type,

                    },

                },

            }));

        };

        reader.readAsDataURL(file);

    };

    return(

        <IKContext

        urlEndpoint={urlEndpoint}

        publicKey={publicKey}

        authenticator={authenticator}

      >

        <IKUpload

          fileName="test-upload.png"

          onError={onError}

          onSuccess={onSuccess}

          useUniqueFileName={true}

          onUploadProgress={onUploadProgress}

          onUploadStart={onUploadStart}

          style={{display:"none"}}

          ref={ikUploadRef}

        />

        {<label onClick={()=>ikUploadRef.current.click()}>

          <img src="/attachment.png" alt="" /></label>}

      </IKContext>

    )

}

export default  Upload

**New Prompt.jsx new**

import { useEffect, useRef, useState } from "react";

import "./newPrompt.css";

import Upload from "../upload/Upload";

import { IKImage } from "imagekitio-react";

import model from "../../lib/gemini";

import Markdown from "react-markdown";

import { useMutation, useQueryClient } from "@tanstack/react-query";

import axios from "axios";

const NewPrompt = ({ data }) => {

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

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

  const [chatHistory, setChatHistory] = useState(data.history || []);

  const [img, setImg] = useState({

    isLoading: false,

    error: "",

    dbData: {},

    aiData: {},

  });

  const endRef = useRef(null);

  const formRef = useRef(null);

  const queryClient = useQueryClient();

  // ✅ Function to detect if query is related to weather using LLM

  const checkWeatherQuery = async (text) => {

    console.log("🤖 Checking if query is about weather...");

    const prompt = `Determine if this query is related to weather and extract the location if applicable: "${text}". If it's about weather, respond with "weather: LOCATION". If not, respond with "other".`;

    try {

      const chat = model.startChat({ history: [], generationConfig: {} });

      const result = await chat.sendMessageStream(prompt);

      let llmResponse = "";

      for await (const chunk of result.stream) {

        llmResponse += chunk.text();

      }

      console.log("🧠 LLM Weather Detection Response:", llmResponse);

      if (llmResponse.toLowerCase().startsWith("weather:")) {

        const location = llmResponse.split(":")[1]?.trim();

        return location || null;

      }

      return null;

    } catch (err) {

      console.error("❌ Error in LLM weather detection:", err);

      return null;

    }

  };

  const mutation = useMutation({

    mutationFn: async () => {

      console.log("🔄 Updating chat history:", { question, answer, img });

      return axios.put(

        `${import.meta.env.VITE\_API\_URL}/api/chats/${data.\_id}`,

        {

          question: question.length ? question : undefined,

          answer,

          img: img.dbData?.filePath ? { filePath: img.dbData.filePath } : null,

        },

        { withCredentials: true }

      );

    },

    onSuccess: () => {

      console.log("✅ Chat history updated!");

      queryClient.invalidateQueries({ queryKey: ["chat", data.\_id] });

      setChatHistory((prevHistory) => [

        ...prevHistory,

        { role: "user", parts: [{ text: question }] },

        { role: "model", parts: [{ text: answer }] },

      ]);

      formRef.current.reset();

      setQuestion("");

      setAnswer("");

      setImg({ isLoading: false, error: "", dbData: {}, aiData: {} });

    },

  });

  useEffect(() => {

    if (chatHistory.length > 0) {

      endRef.current?.scrollIntoView({ behavior: "smooth" });

    }

  }, [chatHistory]);

  // ✅ Handle form submission

  const handleSubmit = async (e) => {

    e.preventDefault();

    const text = e.target.text.value.trim();

    if (!text) return;

    setQuestion(text);

    console.log("🟢 User input received:", text);

    // ✅ Use LLM to check if the query is weather-related

    const location = await checkWeatherQuery(text);

    if (location) {

      console.log("🌤️ Fetching weather for:", location);

      try {

        const weatherRes = await axios.get(

          `${import.meta.env.VITE\_API\_URL}/api/weather/${encodeURIComponent(location)}`,

          { withCredentials: true }

        );

        console.log("✅ Weather API Response:", weatherRes.data);

        const weatherInfo = weatherRes.data;

        const weatherText = `🌤️ The weather in \*\*${weatherInfo.location}\*\* is \*\*${weatherInfo.weather}\*\*.

          - 🌡️ Temperature: \*\*${weatherInfo.temperature}°C\*\*

          - 💧 Humidity: \*\*${weatherInfo.humidity}%\*\*

          - 💨 Wind Speed: \*\*${weatherInfo.wind\_speed} m/s\*\*`;

        setAnswer(weatherText);

        await mutation.mutateAsync();

        return;

      } catch (error) {

        console.error("❌ Weather API error:", error.response?.data || error.message);

        setAnswer("⚠️ Unable to fetch the weather. Please try again later.");

        return;

      }

    }

    console.log("🔵 Not a weather query. Sending to AI model...");

    try {

      const chat = model.startChat({

        history: chatHistory.map((message) => ({

          role: message.role,

          parts: [{ text: message.parts[0].text }],

        })),

        generationConfig: {},

      });

      const input = img.aiData.inlineData ? [{ ...img.aiData }, text] : text;

      const result = await chat.sendMessageStream(input);

      let accumulatedText = "";

      for await (const chunk of result.stream) {

        accumulatedText += chunk.text();

        console.log("📝 AI Response Chunk:", chunk.text());

        setAnswer((prev) => prev + chunk.text());

      }

      await mutation.mutateAsync();

    } catch (err) {

      console.error("❌ AI model error:", err);

      setAnswer("⚠️ An error occurred, please try again.");

    }

  };

  return (

    <>

      {img.isLoading && <div>Loading...</div>}

      {img.dbData?.filePath && (

        <IKImage

          urlEndpoint={import.meta.env.VITE\_IMAGE\_KIT\_ENDPOINT}

          path={img.dbData?.filePath}

          width="380"

          transformation={[{ width: 380 }]}

        />

      )}

      {chatHistory.map((message, i) => (

        <div key={i} className={message.role === "user" ? "message user" : "message"}>

          <Markdown>{message.parts[0].text}</Markdown>

        </div>

      ))}

      <div className="endChat" ref={endRef}></div>

      <form className="newForm" onSubmit={handleSubmit} ref={formRef}>

        <Upload setImg={setImg} />

        <input type="text" name="text" placeholder="Ask me anything..." />

        <button type="submit">

          <img src="/arrow.png" alt="Send" />

        </button>

      </form>

    </>

  );

};

export default NewPrompt;

**Index.js**

**import express from "express";**

**import cors from "cors";**

**import axios from "axios";**

**import ImageKit from "imagekit";**

**import mongoose from "mongoose";**

**import Chat from "./models/chat.js";**

**import UserChats from "./models/userChats.js";**

**import { ClerkExpressRequireAuth } from "@clerk/clerk-sdk-node";**

**const port = process.env.PORT || 3000;**

**const app = express();**

**app.use(cors({**

**origin: process.env.CLIENT\_URL,**

**credentials: true,**

**}));**

**app.use(express.json());**

**const connect = async () => {**

**try {**

**await mongoose.connect(process.env.MONGO);**

**console.log("Connected to MongoDB");**

**} catch (err) {**

**console.log(err);**

**}**

**};**

**const imagekit = new ImageKit({**

**urlEndpoint: process.env.IMAGE\_KIT\_ENDPOINT,**

**publicKey: process.env.IMAGE\_KIT\_PUBLIC\_KEY,**

**privateKey: process.env.IMAGE\_KIT\_PRIVATE\_KEY,**

**});**

**// Image upload authentication**

**app.get("/api/upload", (req, res) => {**

**const result = imagekit.getAuthenticationParameters();**

**res.send(result);**

**});**

**// Weather API route**

**app.get("/api/weather/:location", async (req, res) => {**

**const location = req.params.location;**

**console.log(`🌍 Fetching weather data for: ${location}`);**

**if (!location) {**

**return res.status(400).json({ error: "❌ Invalid location parameter." });**

**}**

**try {**

**const response = await axios.get(`https://api.openweathermap.org/data/2.5/weather`, {**

**params: {**

**q: location,**

**appid: process.env.WEATHER\_API\_KEY,  // Ensure API Key is correct**

**units: "metric",**

**},**

**});**

**console.log("✅ Weather API Response:", response.data);**

**res.json({**

**location: response.data.name,**

**weather: response.data.weather[0].description,**

**temperature: response.data.main.temp,**

**humidity: response.data.main.humidity,**

**wind\_speed: response.data.wind.speed,**

**});**

**} catch (error) {**

**if (error.response) {**

**console.error(`❌ OpenWeather API Error:`, error.response.data);**

**res.status(error.response.status).json({ error: error.response.data.message || "Error fetching weather data!" });**

**} else {**

**console.error(`❌ Internal Server Error:`, error.message);**

**res.status(500).json({ error: "Internal server error while fetching weather data!" });**

**}**

**}**

**});**

**// Create new chat**

**app.post("/api/chats", ClerkExpressRequireAuth(), async (req, res) => {**

**const userId = req.auth.userId;**

**const { text } = req.body;**

**try {**

**const newChat = new Chat({**

**userId,**

**history: [{ role: "user", parts: [{ text }] }],**

**});**

**const savedChat = await newChat.save();**

**const userChats = await UserChats.find({ userId });**

**if (!userChats.length) {**

**const newUsersChats = new UserChats({**

**userId,**

**chats: [{ \_id: savedChat.\_id, title: text.substring(0, 40) }],**

**});**

**await newUsersChats.save();**

**} else {**

**await UserChats.updateOne(**

**{ userId },**

**{ $push: { chats: { \_id: savedChat.\_id, title: text.substring(0, 40) } } }**

**);**

**}**

**res.status(201).send(savedChat.\_id);**

**} catch (err) {**

**console.log(err);**

**res.status(500).send("Error creating chat!");**

**}**

**});**

**// Fetch user chats**

**app.get("/api/userchats", ClerkExpressRequireAuth(), async (req, res) => {**

**const userId = req.auth.userId;**

**try {**

**const userChats = await UserChats.findOne({ userId });**

**if (!userChats) return res.status(200).json([]);**

**res.status(200).send(userChats.chats);**

**} catch (err) {**

**console.log(err);**

**res.status(500).send("Error fetching userchats!");**

**}**

**});**

**// Fetch chat by ID**

**app.get("/api/chats/:id", ClerkExpressRequireAuth(), async (req, res) => {**

**const userId = req.auth.userId;**

**try {**

**const chat = await Chat.findOne({ \_id: req.params.id, userId });**

**res.status(200).send(chat);**

**} catch (err) {**

**console.log(err);**

**res.status(500).send("Error fetching chat!");**

**}**

**});**

**// Update chat by ID**

**app.put("/api/chats/:id", ClerkExpressRequireAuth(), async (req, res) => {**

**const userId = req.auth.userId;**

**const { question, answer, img } = req.body;**

**const newItems = [**

**...(question ? [{ role: "user", parts: [{ text: question }], ...(img && { img }) }] : []),**

**{ role: "model", parts: [{ text: answer }] },**

**];**

**try {**

**const updatedChat = await Chat.findOneAndUpdate(**

**{ \_id: req.params.id, userId },**

**{ $push: { history: { $each: newItems } } },**

**{ new: true } // 🔹 Ensures the updated chat is returned**

**);**

**res.status(200).send(updatedChat);**

**} catch (err) {**

**console.log(err);**

**res.status(500).send("Error updating chat!");**

**}**

**});**

**app.listen(port, () => {**

**connect();**

**console.log("Server running on port", port);**

**});**

**LLMS Newprompt.jsx**

import { useEffect, useRef, useState } from "react";

import "./newPrompt.css";

import Upload from "../upload/Upload";

import { IKImage } from "imagekitio-react";

import model from "../../lib/gemini";

import Markdown from "react-markdown";

import { useMutation, useQueryClient } from "@tanstack/react-query";

import axios from "axios";

const NewPrompt = ({ data }) => {

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

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

  const [loading, setLoading] = useState(false);

  const [img, setImg] = useState({

    isLoading: false,

    error: "",

    dbData: {},

    aiData: {},

  });

  const endRef = useRef(null);

  const formRef = useRef(null);

  const queryClient = useQueryClient();

  const mutation = useMutation({

    mutationFn: async () => {

      console.log("🔄 Updating chat history:", { question, answer, img });

      return axios.put(

        `${import.meta.env.VITE\_API\_URL}/api/chats/${data.\_id}`,

        {

          question: question.length ? question : undefined,

          answer,

          img: img.dbData?.filePath ? { filePath: img.dbData.filePath } : undefined,

        },

        { withCredentials: true }

      );

    },

    onSuccess: () => {

      console.log("✅ Chat history updated!");

      queryClient.invalidateQueries({ queryKey: ["chat", data.\_id] });

      data.history.push({ role: "user", parts: [{ text: question }] });

      data.history.push({ role: "model", parts: [{ text: answer }] });

      formRef.current.reset();

      setQuestion("");

      setAnswer("");

      setImg({ isLoading: false, error: "", dbData: {}, aiData: {} });

    },

  });

  useEffect(() => {

    endRef.current?.scrollIntoView({ behavior: "smooth" });

  }, [data, question, answer, img.dbData]);

  const handleSubmit = async (e) => {

    e.preventDefault();

    const text = e.target.text.value.trim();

    if (!text) return;

    setQuestion(text);

    setLoading(true);

    console.log("🟢 User input received:", text);

    try {

      const chat = model.startChat({

        history: data?.history?.map((message) => ({

          role: message.role,

          parts: [{ text: message.parts[0].text }],

        })) || [],

        generationConfig: {},

      });

      // 🔹 Enhanced prompt to detect weather queries dynamically

      const input = [{

        text: `You are an AI chatbot. Identify if the user is asking about the weather.

        - If it's a weather query, extract the city and return JSON like:

          { "weather\_query": true, "location": "Mannheim" }

        - If not, return a normal response.

        User message: "${text}"`

      }];

      console.log("🔵 Sending query to Gemini...");

      const result = await chat.sendMessageStream(input);

      let accumulatedText = "";

      for await (const chunk of result.stream) {

        accumulatedText += chunk.text();

      }

      console.log("📝 Gemini Raw Response:", accumulatedText);

      try {

        // 🔹 Remove markdown formatting if present

        const cleanText = accumulatedText.replace(/```json|```/g, "").trim();

        // 🔹 Parse the cleaned response

        const parsedResponse = JSON.parse(cleanText);

        if (parsedResponse.weather\_query) {

          console.log("🌤️ Gemini detected a weather query for:", parsedResponse.location);

          // 🔹 Fetch actual weather data

          const weatherRes = await axios.get(

            `${import.meta.env.VITE\_API\_URL}/api/weather/${encodeURIComponent(parsedResponse.location)}`,

            { withCredentials: true }

          );

          const weatherText = `🌤️ The current weather in \*\*${parsedResponse.location}\*\* is \*\*${weatherRes.data.weather}\*\*.

          - 🌡️ Temperature: \*\*${weatherRes.data.temperature}°C\*\*

          - 💧 Humidity: \*\*${weatherRes.data.humidity}%\*\*

          - 💨 Wind Speed: \*\*${weatherRes.data.wind\_speed} m/s\*\*`;

          console.log("✅ Weather response generated:", weatherText);

          // 🔹 Update UI with actual weather data

          setAnswer(weatherText);

          await new Promise((resolve) => setTimeout(resolve, 100));

          await mutation.mutateAsync();

          setLoading(false);

          return; // 🔹 Prevents incorrect response from showing in UI

        }

      } catch (err) {

        console.log("📝 Not a JSON response, using normal Gemini response.");

      }

      // 🔹 If it's not a weather query, use Gemini's normal response

      setAnswer(accumulatedText);

      await new Promise((resolve) => setTimeout(resolve, 100)); // Ensure UI update before mutation

      await mutation.mutateAsync();

      setLoading(false);

    } catch (err) {

      console.error("❌ Gemini API error:", err);

      setAnswer("⚠️ An error occurred, please try again.");

    } finally {

      setLoading(false);

    }

    formRef.current.reset();

  };

  return (

    <>

      {loading && <div>⏳ Processing your request...</div>}

      {img.isLoading && <div>⏳ Uploading image...</div>}

      {img.dbData?.filePath && (

        <IKImage

          urlEndpoint={import.meta.env.VITE\_IMAGE\_KIT\_ENDPOINT}

          path={img.dbData?.filePath}

          width="380"

          transformation={[{ width: 380 }]}

        />

      )}

      {question && <div className="message user">{question}</div>}

      {answer && <div className="message"><Markdown>{answer}</Markdown></div>}

      <div className="endChat" ref={endRef}></div>

      <form className="newForm" onSubmit={handleSubmit} ref={formRef}>

        <Upload setImg={setImg} />

        <input type="text" name="text" placeholder="Ask me anything..." required />

        <button type="submit">

          <img src="/arrow.png" alt="Send" />

        </button>

      </form>

    </>

  );

};

export default NewPrompt;

**WORKING Fully newprompt.jsx**

import { useEffect, useRef, useState } from "react";

import "./newPrompt.css";

import Upload from "../upload/Upload";

import { IKImage } from "imagekitio-react";

import model from "../../lib/gemini";

import Markdown from "react-markdown";

import { useMutation, useQueryClient } from "@tanstack/react-query";

import axios from "axios";

const NewPrompt = ({ data }) => {

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

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

  const [loading, setLoading] = useState(false);

  const [img, setImg] = useState({

    isLoading: false,

    error: "",

    dbData: {},

    aiData: {},

  });

  const endRef = useRef(null);

  const formRef = useRef(null);

  const queryClient = useQueryClient();

  const mutation = useMutation({

    mutationFn: async () => {

      console.log("🔄 Updating chat history:", { question, answer, img });

      return axios.put(

        `${import.meta.env.VITE\_API\_URL}/api/chats/${data.\_id}`,

        {

          question: question.length ? question : undefined,

          answer,

          img: img.dbData?.filePath ? { filePath: img.dbData.filePath } : undefined,

        },

        { withCredentials: true }

      );

    },

    onSuccess: () => {

      console.log("✅ Chat history updated!");

      queryClient.invalidateQueries({ queryKey: ["chat", data.\_id] });

      data.history.push({ role: "user", parts: [{ text: question }] });

      data.history.push({ role: "model", parts: [{ text: answer }] });

      formRef.current.reset();

      setQuestion("");

      setAnswer("");

      setImg({ isLoading: false, error: "", dbData: {}, aiData: {} });

    },

  });

  useEffect(() => {

    endRef.current?.scrollIntoView({ behavior: "smooth" });

  }, [data, question, answer, img.dbData]);

  const handleSubmit = async (e) => {

    e.preventDefault();

    const text = e.target.text.value.trim();

    if (!text) return;

    setQuestion(text);

    setLoading(true);

    console.log("🟢 User input received:", text);

    try {

      const chat = model.startChat({

        history: data?.history?.map((message) => ({

          role: message.role,

          parts: [{ text: message.parts[0].text }],

        })) || [],

        generationConfig: {},

      });

      // 🔹 Ask Gemini to detect if the query is about weather

      const input = [{

        text: `You are an AI chatbot. Identify if the user is asking about the weather.

        - If it's a weather query, extract the city and return JSON like:

          { "weather\_query": true, "location": "Mannheim" }

        - If it's NOT a weather query, return:

          { "weather\_query": false }

        User message: "${text}"`

      }];

      console.log("🔵 Sending query to Gemini...");

      const result = await chat.sendMessageStream(input);

      let accumulatedText = "";

      for await (const chunk of result.stream) {

        accumulatedText += chunk.text();

      }

      console.log("📝 Gemini Raw Response:", accumulatedText);

      try {

        // 🔹 Clean response (removes markdown formatting if present)

        const cleanText = accumulatedText.replace(/```json|```/g, "").trim();

        const parsedResponse = JSON.parse(cleanText);

        // 🔹 If Gemini says it's NOT a weather query, get a normal AI response

        if (parsedResponse.weather\_query === false) {

          console.log("🔵 Not a weather query, fetching normal AI response...");

          const aiResponse = await chat.sendMessageStream([{ text }]);

          let aiText = "";

          for await (const chunk of aiResponse.stream) {

            aiText += chunk.text();

          }

          setAnswer(aiText);

          await new Promise((resolve) => setTimeout(resolve, 100));

          await mutation.mutateAsync();

          setLoading(false);

          return;

        }

        // 🔹 If Gemini detected a weather query, fetch weather data

        if (parsedResponse.weather\_query && parsedResponse.location) {

          console.log("🌤️ Gemini detected a weather query for:", parsedResponse.location);

          const weatherRes = await axios.get(

            `${import.meta.env.VITE\_API\_URL}/api/weather/${encodeURIComponent(parsedResponse.location)}`,

            { withCredentials: true }

          );

          const {

            weather, temperature, humidity, wind\_speed, feels\_like,

            rain\_chance, sunrise, sunset, air\_quality

          } = weatherRes.data;

          const formatTime = (timestamp) => {

              const date = new Date(timestamp \* 1000);

              return date.toLocaleTimeString([], { hour: "2-digit", minute: "2-digit" });

          };

          // 🔹 Generate a realistic weather report

          const weatherText = `🌤️ \*\*Weather Report for ${parsedResponse.location}:\*\*

          - 🌡️ Temperature: \*\*${temperature}°C\*\* )

          - ☁️ Condition: \*\*${weather}\*\*

          - 💧 Humidity: \*\*${humidity}%\*\*

          - 💨 Wind Speed: \*\*${wind\_speed} m/s\*\*

          ${rain\_chance ? `- 🌧️ Chance of Rain: \*\*${rain\_chance}%\*\*` : ""}

          ${air\_quality ? `- 🌍 Air Quality Index (AQI): \*\*${air\_quality}\*\*` : ""}

          - 🌅 Sunrise: \*\*${formatTime(sunrise)}\*\*

          - 🌇 Sunset: \*\*${formatTime(sunset)}\*\*`

          ;

          console.log("✅ Enhanced Weather Response Generated:", weatherText);

          setAnswer(weatherText);

          await new Promise((resolve) => setTimeout(resolve, 100));

          await mutation.mutateAsync();

          setLoading(false);

          return;

        }

      } catch (err) {

        console.log("📝 Gemini did not return JSON, using normal response.");

      }

      setAnswer(accumulatedText);

      await new Promise((resolve) => setTimeout(resolve, 100));

      await mutation.mutateAsync();

      setLoading(false);

    } catch (err) {

      console.error("❌ Gemini API error:", err);

      setAnswer("⚠️ An error occurred, please try again.");

    } finally {

      setLoading(false);

    }

    formRef.current.reset();

  };

  return (

    <>

      {loading && <div>⏳ Processing your request...</div>}

      {img.isLoading && <div>⏳ Uploading image...</div>}

      {img.dbData?.filePath && (

        <IKImage

          urlEndpoint={import.meta.env.VITE\_IMAGE\_KIT\_ENDPOINT}

          path={img.dbData?.filePath}

          width="380"

          transformation={[{ width: 380 }]}

        />

      )}

      {question && <div className="message user">{question}</div>}

      {answer && <div className="message"><Markdown>{answer}</Markdown></div>}

      <div className="endChat" ref={endRef}></div>

      <form className="newForm" onSubmit={handleSubmit} ref={formRef}>

        <Upload setImg={setImg} />

        <input type="text" name="text" placeholder="Ask me anything..." required />

        <button type="submit">

          <img src="/arrow.png" alt="Send" />

        </button>

      </form>

    </>

  );

};

export default NewPrompt;

**Newprompt.jsx LLM Implementation working**

import { useEffect, useRef, useState } from "react";

import "./newPrompt.css";

import Upload from "../upload/Upload";

import { IKImage } from "imagekitio-react";

import model from "../../lib/gemini";

import Markdown from "react-markdown";

import { useMutation, useQueryClient } from "@tanstack/react-query";

import axios from "axios";

const NewPrompt = ({ data }) => {

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

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

  const [loading, setLoading] = useState(false);

  const [img, setImg] = useState({

    isLoading: false,

    error: "",

    dbData: {},

    aiData: {},

  });

  const endRef = useRef(null);

  const formRef = useRef(null);

  const queryClient = useQueryClient();

  const mutation = useMutation({

    mutationFn: async () => {

      console.log("🔄 Updating chat history:", { question, answer, img });

      return axios.put(

        `${import.meta.env.VITE\_API\_URL}/api/chats/${data.\_id}`,

        {

          question: question.length ? question : undefined,

          answer,

          img: img.dbData?.filePath || undefined,

        },

        { withCredentials: true }

      );

    },

    onSuccess: () => {

      console.log("✅ Chat history updated!");

      queryClient.invalidateQueries({ queryKey: ["chat", data.\_id] });

      data.history.push({ role: "user", parts: [{ text: question }] });

      data.history.push({ role: "model", parts: [{ text: answer }] });

      formRef.current.reset();

      setQuestion("");

      setAnswer("");

      setImg({ isLoading: false, error: "", dbData: {}, aiData: {} });

    },

  });

  useEffect(() => {

    endRef.current?.scrollIntoView({ behavior: "smooth" });

  }, [data, question, answer, img.dbData]);

  const handleSubmit = async (e) => {

    e.preventDefault();

    const text = e.target.text.value.trim();

    if (!text) return;

    setQuestion(text);

    setLoading(true);

    console.log("🟢 User input received:", text);

    try {

      const chat = model.startChat({

        history: data?.history?.map((message) => ({

          role: message.role,

          parts: [{ text: message.parts[0].text }],

        })) || [],

        generationConfig: {},

      });

      // 🔹 Ask Gemini to detect if the query is about weather

      const input = [{

        text: `You are an AI chatbot. Identify if the user is asking about the weather.

        - If it's a weather query, extract the city and return JSON like:

          { "weather\_query": true, "location": "Mannheim" }

        - If it's NOT a weather query, return:

          { "weather\_query": false }

        User message: "${text}"`

      }];

      console.log("🔵 Sending query to Gemini...");

      const result = await chat.sendMessageStream(input);

      let accumulatedText = "";

      for await (const chunk of result.stream) {

        accumulatedText += chunk.text();

      }

      console.log("📝 Gemini Raw Response:", accumulatedText);

      try {

        // 🔹 Clean response (removes markdown formatting if present)

        const cleanText = accumulatedText.replace(/```json|```/g, "").trim();

        const parsedResponse = JSON.parse(cleanText);

        // 🔹 If Gemini says it's NOT a weather query, get a normal AI response.

        // Here we include the inline image data if it exists.

        if (parsedResponse.weather\_query === false) {

          console.log("🔵 Not a weather query, fetching normal AI response...");

          const inputMessage = Object.entries(img.aiData).length

            ? [img.aiData, { text }]

            : [{ text }];

          const aiResponse = await chat.sendMessageStream(inputMessage);

          let aiText = "";

          for await (const chunk of aiResponse.stream) {

            aiText += chunk.text();

          }

          setAnswer(aiText);

          await new Promise((resolve) => setTimeout(resolve, 100));

          await mutation.mutateAsync();

          setLoading(false);

          return;

        }

        // 🔹 If Gemini detected a weather query, fetch weather data

        if (parsedResponse.weather\_query && parsedResponse.location) {

          console.log("🌤️ Gemini detected a weather query for:", parsedResponse.location);

          const weatherRes = await axios.get(

            `${import.meta.env.VITE\_API\_URL}/api/weather/${encodeURIComponent(parsedResponse.location)}`,

            { withCredentials: true }

          );

          const {

            weather, temperature, humidity, wind\_speed,

            rain\_chance, air\_quality

          } = weatherRes.data;

          const formatTime = (timestamp) => {

            const date = new Date(timestamp \* 1000);

            return date.toLocaleTimeString([], { hour: "2-digit", minute: "2-digit" });

          };

          // 🔹 Generate a realistic weather report

          const weatherText = `🌤️ \*\*Weather Report for ${parsedResponse.location}:\*\*

          - 🌡️ Temperature: \*\*${temperature}°C\*\*

          - ☁️ Condition: \*\*${weather}\*\*

          - 💧 Humidity: \*\*${humidity}%\*\*

          - 💨 Wind Speed: \*\*${wind\_speed} m/s\*\*

          ${rain\_chance ? `- 🌧️ Chance of Rain: \*\*${rain\_chance}%\*\*` : ""}

          ${air\_quality ? `- 🌍 Air Quality Index (AQI): \*\*${air\_quality}\*\*` : ""} `;

          console.log("✅ Enhanced Weather Response Generated:", weatherText);

          setAnswer(weatherText);

          await new Promise((resolve) => setTimeout(resolve, 100));

          await mutation.mutateAsync();

          setLoading(false);

          return;

        }

      } catch (err) {

        console.log("📝 Gemini did not return JSON, using normal response.");

      }

      // Fallback: if Gemini's JSON parsing fails, use the raw accumulated text.

      setAnswer(accumulatedText);

      await new Promise((resolve) => setTimeout(resolve, 100));

      await mutation.mutateAsync();

      setLoading(false);

    } catch (err) {

      console.error("❌ Gemini API error:", err);

      setAnswer("⚠️ An error occurred, please try again.");

    } finally {

      setLoading(false);

    }

    formRef.current.reset();

  };

  return (

    <>

      {loading && <div>⏳ Processing your request...</div>}

      {img.isLoading && <div>⏳ Uploading image...</div>}

      {img.dbData?.filePath && (

        <IKImage

          urlEndpoint={import.meta.env.VITE\_IMAGE\_KIT\_ENDPOINT}

          path={img.dbData?.filePath}

          width="380"

          transformation={[{ width: 380 }]}

        />

      )}

      {question && <div className="message user">{question}</div>}

      {answer && <div className="message"><Markdown>{answer}</Markdown></div>}

      <div className="endChat" ref={endRef}></div>

      <form className="newForm" onSubmit={handleSubmit} ref={formRef}>

        <Upload setImg={setImg} />

        <input type="text" name="text" placeholder="Ask me anything..." required />

        <button type="submit">

          <img src="/arrow.png" alt="Send" />

        </button>

      </form>

    </>

  );

};

export default NewPrompt;