Backend

Npm init -y

Npm install typescript

Npx tsc –init

npm install dotenv

npm install @Qtypes/express

npm install express

telegram api chutiya hai

baseURL: "https://models.inference.ai.azure.com",

apiKey: ghp\_38Xp7W5i3Iv6Ash6RsCrFeaVrnR53K2W9bNO

npm i @types/cors

npm i cors

whenever we write a prompt as user first it will hit template route where it determines wether it is a react proj or nodejs proj means backend and according give prompts to AI

import OpenAI from "openai";

import \* as dotenv from "dotenv";

import express, { Request, Response, Application } from "express";

import { BASE\_PROMPT, getSystemPrompt } from "./prompts";

import { basePrompt as nodeBasePrompt } from "./defaults/node";

import { basePrompt as reactBasePrompt } from "./defaults/react";

// Load environment variables

dotenv.config();

const API\_KEY = process.env.CLAUDE\_API\_KEY || "your-api-key-here";

const BASE\_URL = "https://models.inference.ai.azure.com";

const app: Application = express();

app.use(express.json());

const client = new OpenAI({

  apiKey: API\_KEY,

  baseURL: BASE\_URL,

});

app.post("/template", async (req: Request, res: Response): Promise<void> => {

  try {

    const { prompt }: { prompt: string } = req.body;

    if (!prompt) {

      res.status(400).json({ message: "❌ Prompt is required" });

      return;

    }

    // 🔹 STRICT SYSTEM PROMPT: Ensures AI only returns JSON

    const systemPrompt = `

        You are a JSON response generator. Your task is to return ONLY a JSON object in this format:

  { "projectType": "react" } or { "projectType": "node" }.

  Do NOT include any explanations, code, or extra text. Your response must be valid JSON.

    `;

    const response = await client.chat.completions.create({

      model: "gpt-4o",

      messages: [

        { role: "system", content: systemPrompt },

        { role: "user", content: prompt },

      ],

      max\_tokens: 8000,

      response\_format: { type: "json\_object" }, // 🔹 Ensures structured output

    });

    // console.log("🟢 Full API Response:", JSON.stringify(response, null, 2));

    // 🔹 Validate AI response

    const messageContent = response.choices?.[0]?.message?.content;

    if (!messageContent) {

      console.error("❌ AI returned empty response.");

      res.status(400).json({ message: "❌ AI response is empty" });

      return;

    }

    let projectType: string;

    try {

      const parsedContent = JSON.parse(messageContent); // 🔹 Ensure valid JSON

      if (!parsedContent.projectType) {

        throw new Error("Missing 'projectType' in AI response");

      }

      projectType = parsedContent.projectType;

    } catch (parseError) {

      console.error("❌ Error parsing AI response:", parseError);

      res.status(400).json({ message: "❌ AI response is not in valid JSON format", rawResponse: response });

      return;

    }

    // 🔹 Respond based on AI decision

    if (projectType === "react") {

      res.json({

        prompts: [

          BASE\_PROMPT,

          `Here is  a react  artifact that contains all files of the project visible to you.\nConsider the contents of ALL files in the project.\n\n${reactBasePrompt}\n\nHere is a list of files that exist on the file system but are not being shown to you:\n\n  - .gitignore\n  - package-lock.json\n`,

        ],

        uiPrompts: [reactBasePrompt],

      });

      return;

    }

    if (projectType === "node") {

      res.json({

        prompts: [

          `Here is a node artifact that contains all files of the project visible to you.\nConsider the contents of ALL files in the project.\n\n${nodeBasePrompt}\n\nHere is a list of files that exist on the file system but are not being shown to you:\n\n  - .gitignore\n  - package-lock.json\n`,

        ],

        uiPrompts: [nodeBasePrompt],

      });

      return;

    }

    res.status(403).json({ message: "❌ You can't access this" });

  } catch (error: any) {

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

    res.status(500).json({ message: "❌ Internal Server Error" });

  }

});

// creating chat endpoint

// 🔹 Creating chat endpoint

app.post("/chat", async (req: Request, res: Response): Promise<void> => {

    try {

        const { messages }: { messages: Array<{ role: "user" | "system" | "assistant"; content: string }> } = req.body;

        if (!messages || !Array.isArray(messages) || messages.length === 0) {

            res.status(400).json({ message: "❌ Messages array is required and must be non-empty" });

            return;

        }

        // 🔹 System Prompt for Additional Control

        const systemPrompt = getSystemPrompt();

        // 🔹 Ensure messages conform to OpenAI API format

        const chatMessages = [

            { role: "system", content: systemPrompt },

            ...messages

        ] as { role: "user" | "system" | "assistant"; content: string }[]; // ✅ Explicitly set the type

        // 🔹 Send Request to GPT-4

        const response = await client.chat.completions.create({

            model: "gpt-4o",

            messages: chatMessages, // ✅ Ensure proper type

            max\_tokens: 8000,

        });

        // 🔹 Validate AI Response

        const messageContent = response.choices?.[0]?.message?.content;

        if (!messageContent) {

            res.status(400).json({ message: "❌ AI response is empty" });

            return;

        }

        // 🔹 Send AI Response

        res.json({ message: messageContent });

    } catch (error: any) {

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

        res.status(500).json({ message: "❌ Internal Server Error" });

    }

});

// ✅ Ensure app listens properly

app.listen(3000, () => console.log("🚀 Server running on port 3000"));