NAME:CHITTETI KUSELA

REG NO:21BCE9158

ZOCKET ASSIGNMENT

AI Chat App – Documentation

Introduction

Project Overview

This project is an AI-powered chat application built with:

- Next.js (Frontend & Backend API handling)
- Google Gemini AI API (AI-generated responses)
- Tailwind CSS (User interface styling)
- GitHub (Version control)
- Vercel (Hosting and deployment)

Objective

The main goal of this project is to provide an AI chatbot that users can interact with in real-time, receiving responses generated by Google Gemini AI.

Problem Statement & Approach

Problem Statement

Many businesses and individuals need AI-powered chatbots for automation, customer support, or knowledge retrieval. However, integrating AI models with web applications can be complex.

Solution & Approach

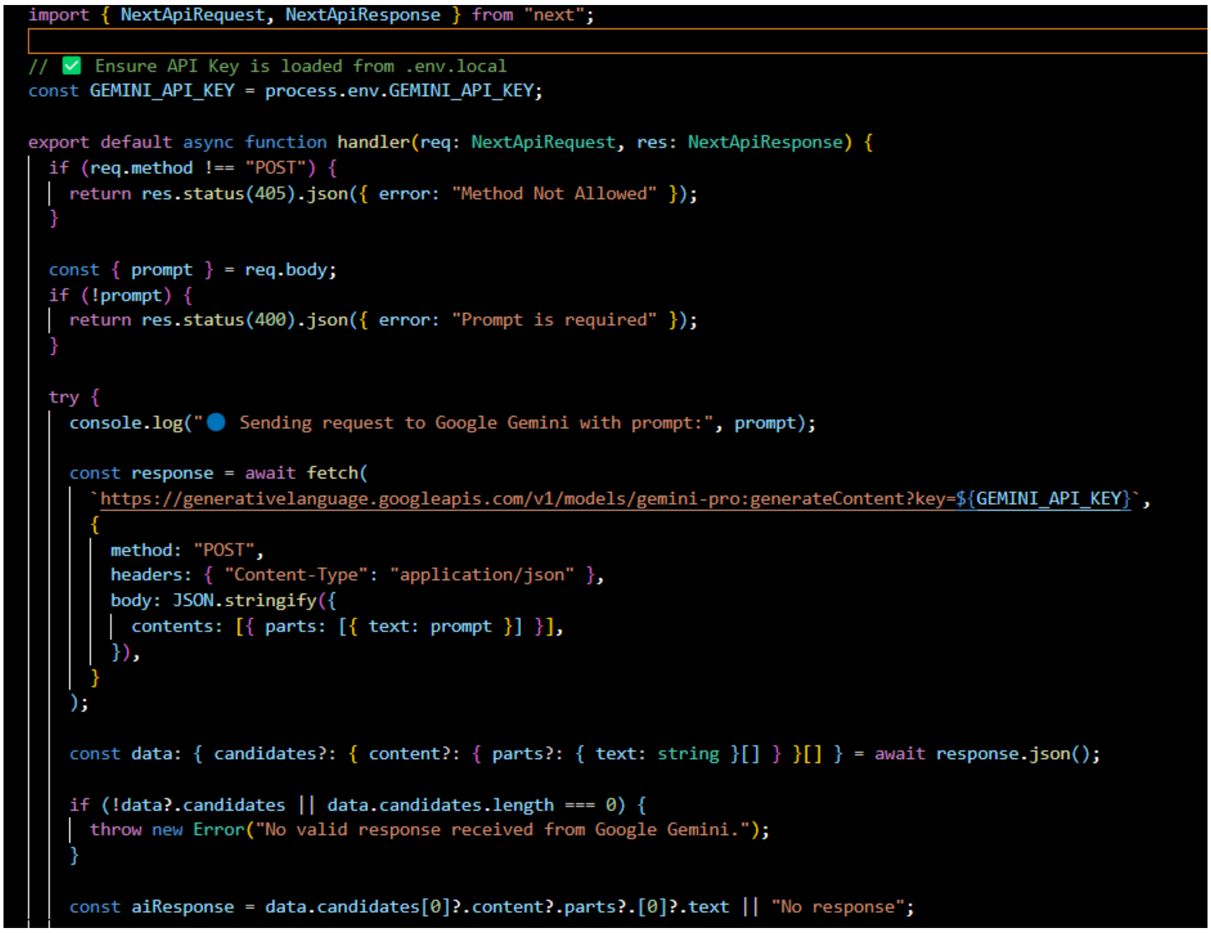
- Frontend: A simple chat interface where users can enter queries and receive AI responses.
- Backend: A Next.js API route that handles requests, communicates with the Google Gemini AI API, and returns AI-generated responses.

- Deployment: The entire application is hosted on Vercel, making it easily accessible.

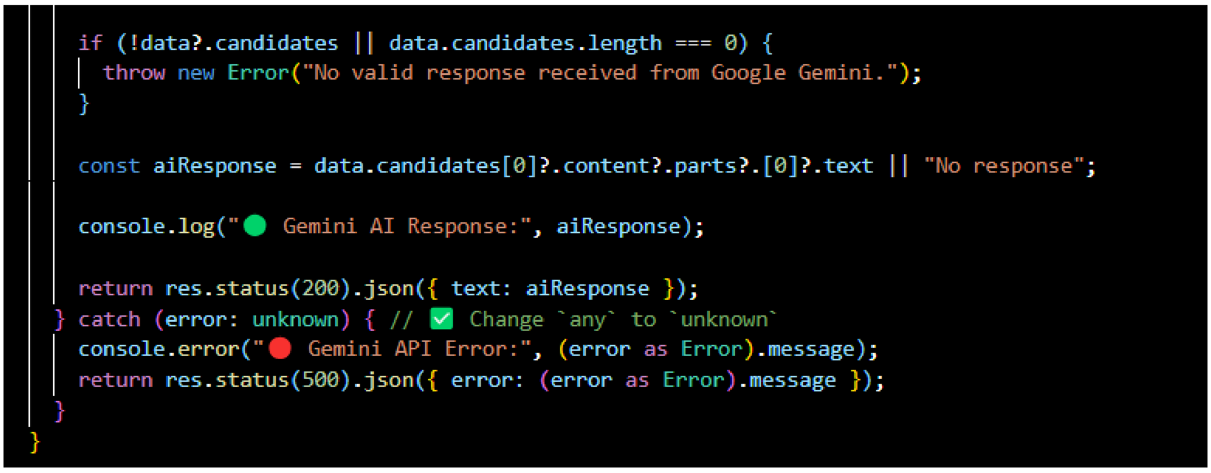
Backend - AI API Integration (`pages/api/ai.ts`)

- This file serves as the backend API route that handles user requests and communicates with the Google Gemini API.

Code Explanation



```
import { NextApiRequest, NextApiResponse } from "next";

// ✅ Ensure API Key is loaded from .env.local
const GEMINI_API_KEY = process.env.GEMINI_API_KEY;

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== "POST") {
    return res.status(405).json({ error: "Method Not Allowed" });
  }

  const { prompt } = req.body;
  if (!prompt) {
    return res.status(400).json({ error: "Prompt is required" });
  }

  try {
    console.log("🔵 Sending request to Google Gemini with prompt:", prompt);

    const response = await fetch(
      `https://generativelanguage.googleapis.com/v1/models/gemini-pro:generateContent?key=${GEMINI_API_KEY}`,
      {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          contents: [{ parts: [{ text: prompt }] }],
        }),
      }
    );

    const data: { candidates?: { content?: { parts?: { text: string }[] } }[] } = await response.json();

    if (!data?.candidates || data.candidates.length === 0) {
      throw new Error("No valid response received from Google Gemini.");
    }

    const aiResponse = data.candidates[0]?.content?.parts?.[0]?.text || "No response";
```



```
    if (!data?.candidates || data.candidates.length === 0) {
      throw new Error("No valid response received from Google Gemini.");
    }

    const aiResponse = data.candidates[0]?.content?.parts?.[0]?.text || "No response";

    console.log("🟢 Gemini AI Response:", aiResponse);

    return res.status(200).json({ text: aiResponse });
  } catch (error: unknown) { // ✅ Change `any` to `unknown`
    console.error("🔴 Gemini API Error:", (error as Error).message);
    return res.status(500).json({ error: (error as Error).message });
  }
}
```

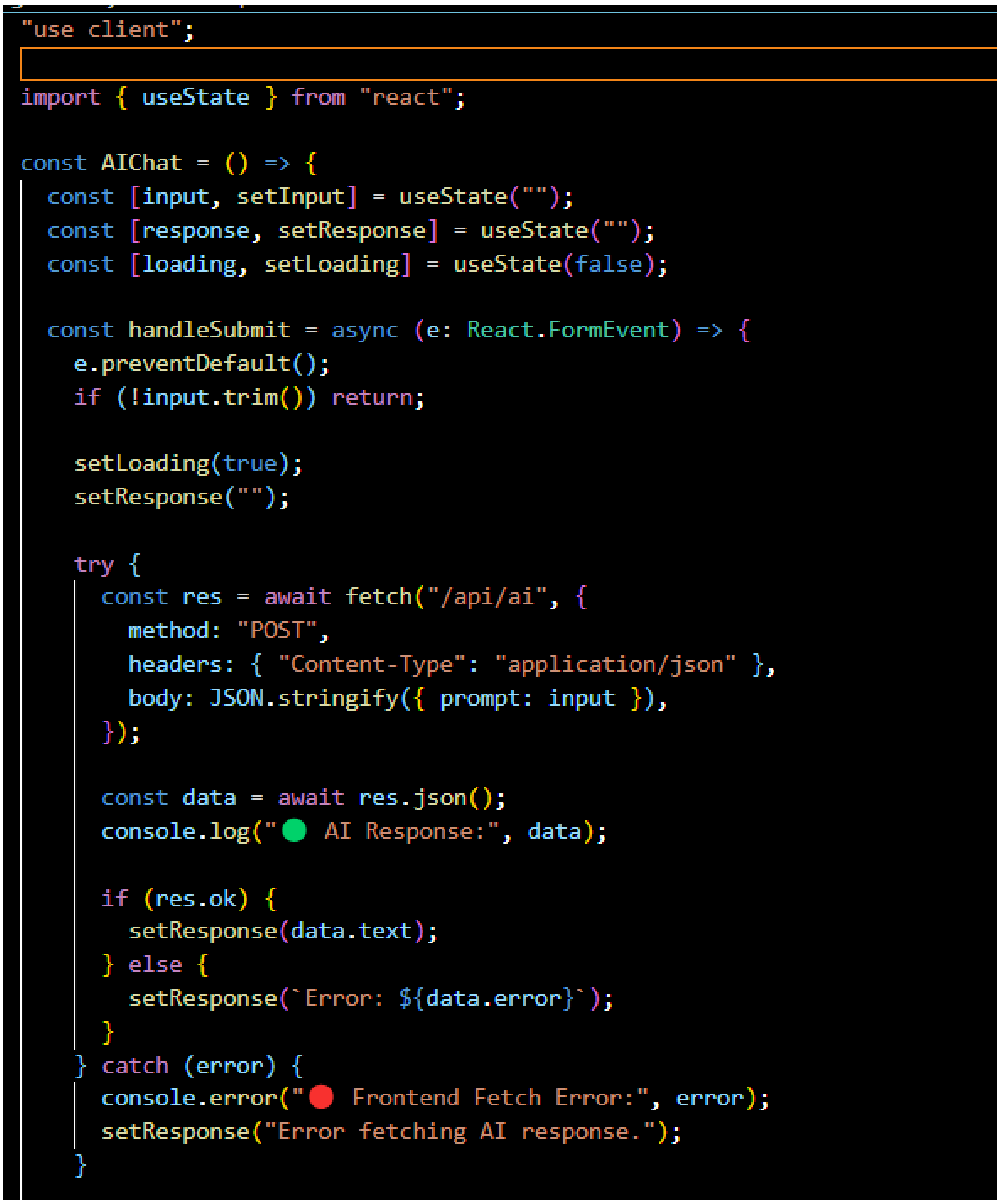
Functionality:

- Only accepts POST requests.
- Extracts the user’s input (prompt).
- Calls Google Gemini API for AI response.
- Returns the AI-generated text.
- Handles errors properly.

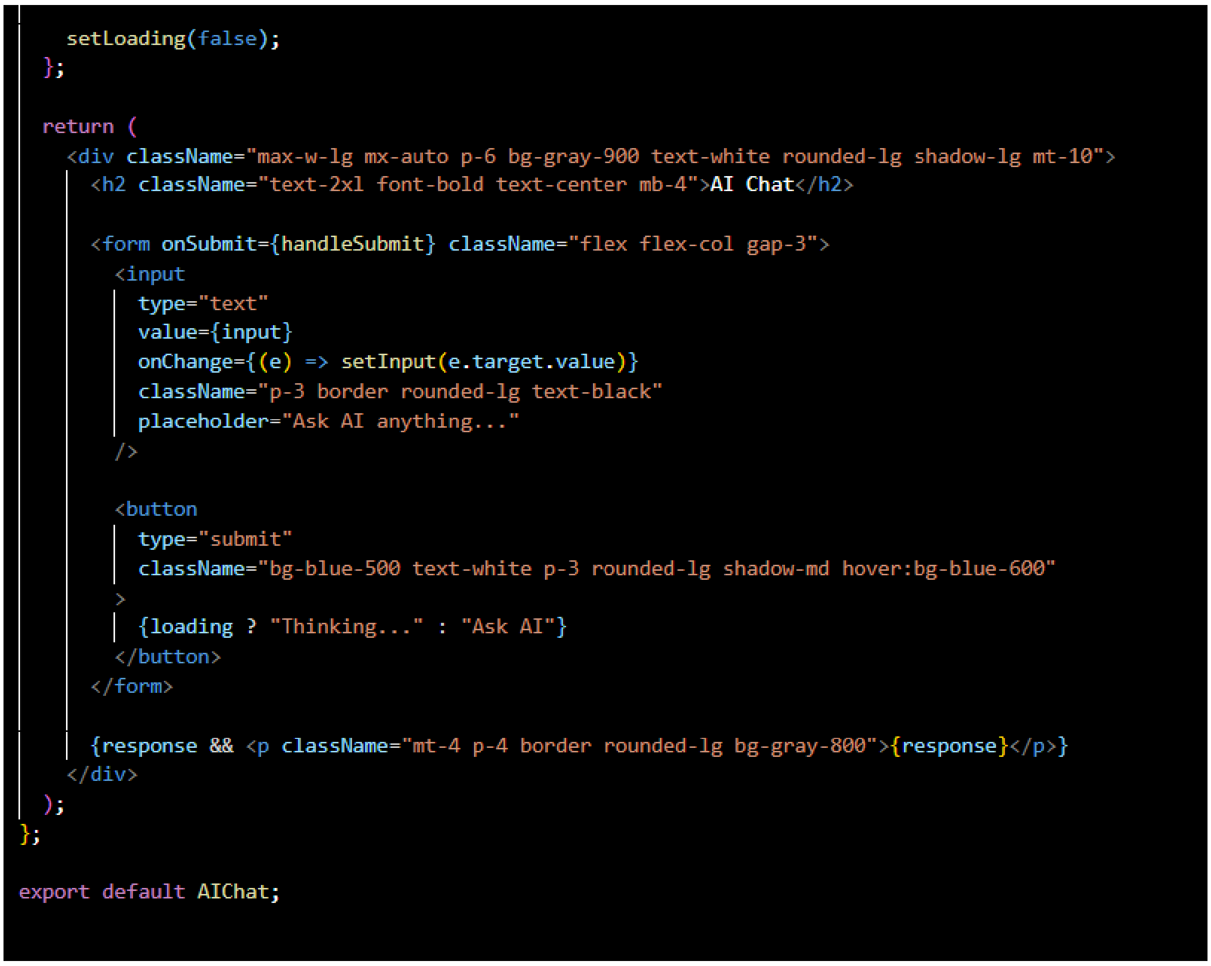
Frontend - Chat Interface (`components/AIChat.tsx`)

- This file contains the chat UI where users interact with the AI.

  The frontend of this AI chat application provides users with a simple and intuitive interface to interact with the AI. It is built using Next.js with React hooks for state management and Tailwind CSS for styling. The interface allows users to type in their queries, send them to the backend API, and display the AI-generated responses dynamically. The chat component also ensures a smooth user experience by handling loading states and error messages effectively.



```
"use client";

import { useState } from "react";

const AIChat = () => {
  const [input, setInput] = useState("");
  const [response, setResponse] = useState("");
  const [loading, setLoading] = useState(false);

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!input.trim()) return;

    setLoading(true);
    setResponse("");

    try {
      const res = await fetch("/api/ai", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ prompt: input }),
      });

      const data = await res.json();
      console.log("🟢 AI Response:", data);

      if (res.ok) {
        setResponse(data.text);
      } else {
        setResponse(`Error: ${data.error}`);
      }
    } catch (error) {
      console.error("🔴 Frontend Fetch Error:", error);
      setResponse("Error fetching AI response.");
    }
```



```
    setLoading(false);
  };

  return (
    <div className="max-w-lg mx-auto p-6 bg-gray-900 text-white rounded-lg shadow-lg mt-10">
      <h2 className="text-2xl font-bold text-center mb-4">AI Chat</h2>

      <form onSubmit={handleSubmit} className="flex flex-col gap-3">
        <input
          type="text"
          value={input}
          onChange={(e) => setInput(e.target.value)}
          className="p-3 border rounded-lg text-black"
          placeholder="Ask AI anything..."
        />

        <button
          type="submit"
          className="bg-blue-500 text-white p-3 rounded-lg shadow-md hover:bg-blue-600"
        >
          {loading ? "Thinking..." : "Ask AI"}
        </button>
      </form>

      {response && <p className="mt-4 p-4 border rounded-lg bg-gray-800">{response}</p>}
    </div>
  );
};

export default AIChat;
```

Functionality:

- Takes user input.
- Sends it to /api/ai.
- Displays the AI response.

Conclusion:

This project successfully demonstrates the integration of Google Gemini AI with a Next.js-based web application. By leveraging Next.js for both frontend and backend, we built a scalable and efficient AI chatbot. The use of Tailwind CSS ensures a visually appealing UI, while Vercel deployment makes the application easily accessible online. Future improvements include adding user authentication, improving error handling, and supporting voice input. This AI Chat App is a great step toward building more intelligent web applications.