**Explicación línea por línea del código:**

javascript

CopiarEditar

const axios = require('axios');

* Importa la librería axios para realizar solicitudes HTTP.

javascript

CopiarEditar

const bodyParser = require('body-parser');

const cors = require('cors');

const fs = require('fs');

const path = require('path');

const express = require('express');

* Importa librerías esenciales:
  + body-parser: Para procesar cuerpos de solicitudes JSON.
  + cors: Para permitir solicitudes de recursos desde dominios diferentes.
  + fs: Para manipular el sistema de archivos.
  + path: Para manejar rutas de archivos de manera segura.
  + express: Framework para construir servidores web.

javascript

CopiarEditar

const app = express();

const PORT = 3000;

* Inicializa una aplicación de Express y define el puerto del servidor (3000).

javascript

CopiarEditar

app.use(cors());

app.use(bodyParser.json());

* Configura middleware:
  + cors: Habilita CORS.
  + bodyParser.json(): Analiza cuerpos de solicitudes con formato JSON.

javascript

CopiarEditar

require('dotenv').config();

const API\_KEY = process.env.API\_KEY;

* Carga las variables de entorno desde un archivo .env y extrae API\_KEY.

javascript

CopiarEditar

if (!API\_KEY) {

console.error("⚠️ ERROR: No se encontró API\_KEY en las variables de entorno.");

process.exit(1);

}

* Verifica si API\_KEY existe; si no, muestra un error y detiene la ejecución del programa.

javascript

CopiarEditar

app.post('/consulta', async (req, res) => {

const pregunta = req.body.pregunta;

* Define una ruta POST /consulta. Extrae pregunta del cuerpo de la solicitud.

javascript

CopiarEditar

if (!pregunta) {

return res.status(400).json({

respuestaFinal: "```markdown\n⚠️ \*\*Error:\*\* La pregunta es requerida.\n```"

});

}

* Si pregunta está vacía, responde con un error 400 (solicitud incorrecta).

javascript

CopiarEditar

const indexPath = path.join(\_\_dirname, 'Documentos', 'Document\_Index.txt');

const documentIndex = fs.readFileSync(indexPath, 'utf8');

* Obtiene la ruta y el contenido del archivo Document\_Index.txt.

javascript

CopiarEditar

const primerRequerimiento = {

model: "gpt-3.5-turbo",

messages: [

{ role: "system", content: "Eres experto en indexación basada en Concept Curve Embeddings." },

{ role: "user", content: `Mi pregunta es la siguiente: ${pregunta}\n\nLee el Document\_Index.txt:\n${documentIndex}\n\nNo respondas la pregunta. Dime en qué chunks irías a buscar la información. Respóndeme solamente el nombre de dos archivos, dos palabras separadas por coma.` }

],

max\_tokens: 50,

temperature: 0

};

* Define un objeto para hacer una solicitud al modelo gpt-3.5-turbo, preguntando qué chunks del índice contienen información relevante.

javascript

CopiarEditar

const primerRespuesta = await axios.post('https://api.openai.com/v1/chat/completions', primerRequerimiento, {

headers: {

'Authorization': `Bearer ${API\_KEY}`,

'Content-Type': 'application/json'

}

});

* Envía la solicitud al modelo de OpenAI usando axios.

javascript

CopiarEditar

const archivosRelevantes = primerRespuesta.data.choices[0].message.content.trim();

const archivos = archivosRelevantes.split(',').map(file => file.trim());

const archivo1 = archivos[0];

const archivo2 = archivos[1];

* Procesa la respuesta, separando los nombres de los dos archivos relevantes.

javascript

CopiarEditar

const archivo1Path = path.join(\_\_dirname, 'Documentos', archivo1);

if (!fs.existsSync(archivo1Path)) {

return res.status(404).json({

respuestaFinal: `\`\`\`markdown\n⚠️ \*\*Error:\*\* El archivo \`${archivo1}\` no existe.\n\`\`\``

});

}

* Verifica si el primer archivo existe en el sistema; si no, responde con un error 404.

javascript

CopiarEditar

const contenidoArchivo1 = fs.readFileSync(archivo1Path, 'utf8');

const segundoRequerimiento1 = {

model: "gpt-3.5-turbo",

messages: [

{ role: "system", content: "Responde preguntas basadas en documentos cargados. Si la respuesta no está en el documento, responde 'no'." },

{ role: "user", content: `Mi pregunta es la siguiente: ${pregunta}\n\nDocumento relevante:\n${contenidoArchivo1.replace(/`/g, '\\`')}` }

],

max\_tokens: 500,

temperature: 0.1

};

* Lee el contenido del primer archivo y construye otra solicitud para buscar una respuesta basada en este documento.

javascript

CopiarEditar

const respuesta1 = await axios.post('https://api.openai.com/v1/chat/completions', segundoRequerimiento1, {

headers: {

'Authorization': `Bearer ${API\_KEY}`,

'Content-Type': 'application/json'

}

});

const respuestaFinal1 = respuesta1.data.choices[0].message.content.trim();

* Envía la solicitud al modelo y obtiene la respuesta.

javascript

CopiarEditar

if (respuestaFinal1.toLowerCase() !== 'no') {

return res.json({ respuestaFinal: `\`\`\`markdown\n${respuestaFinal1}\n\`\`\`` });

}

* Si la respuesta es válida y no es "no", la envía al cliente.

javascript

CopiarEditar

const archivo2Path = path.join(\_\_dirname, 'Documentos', archivo2);

if (!fs.existsSync(archivo2Path)) {

return res.status(404).json({

respuestaFinal: `\`\`\`markdown\n⚠️ \*\*Error:\*\* El archivo \`${archivo2}\` no existe.\n\`\`\``

});

}

* Si el primer archivo no contiene la respuesta, verifica la existencia del segundo archivo.

javascript

CopiarEditar

const contenidoArchivo2 = fs.readFileSync(archivo2Path, 'utf8');

const segundoRequerimiento2 = {

model: "gpt-3.5-turbo",

messages: [

{ role: "system", content: "Responde preguntas basadas en documentos cargados." },

{ role: "user", content: `Mi pregunta es la siguiente: ${pregunta}\n\nDocumento relevante:\n${contenidoArchivo2.replace(/`/g, '\\`')}` }

],

max\_tokens: 500,

temperature: 0.1

};

* Si el segundo archivo existe, genera otra solicitud con su contenido.

javascript

CopiarEditar

const respuesta2 = await axios.post('https://api.openai.com/v1/chat/completions', segundoRequerimiento2, {

headers: {

'Authorization': `Bearer ${API\_KEY}`,

'Content-Type': 'application/json'

}

});

const respuestaFinal2 = respuesta2.data.choices[0].message.content.trim();

* Envía la solicitud al modelo y obtiene la respuesta del segundo archivo.

javascript

CopiarEditar

return res.json({ respuestaFinal: `\`\`\`markdown\n${respuestaFinal2}\n\`\`\`` });

* Devuelve la respuesta obtenida del segundo archivo al cliente.

javascript

CopiarEditar

app.listen(PORT, () => {

console.log(`Servidor escuchando en http://localhost:${PORT}`);

});

* Inicia el servidor y lo pone a escuchar en el puerto definido (3000).