Detailed Overview of the Code

This code is a **Next.js API route** that processes a **job description** and an uploaded **resume file** to generate **interview questions** using OpenAI's GPT-4o-mini model.

---

1. Overview of Dependencies

```
import OpenAI from "openai";

import { NextResponse } from "next/server";

import { PdfReader } from "pdfreader";

import mammoth from "mammoth";

import dotenv from "dotenv";
```

- `OpenAI`: Used to communicate with OpenAI's API.
- `NextResponse`: Utility for sending responses in Next.js API routes.
- `PdfReader`: Extracts text from PDF files.
- `mammoth`: Extracts plain text from `.docx` files.
- `dotenv`: Loads environment variables (API keys) from a `.env` file.

```
dotenv.config();
```

- Loads **environment variables** so that the OpenAI API key can be accessed.

---

2. Setting Up OpenAI

```
const openai = new OpenAI({ apiKey: process.env.OPEN_AI_API_KEY });
```

- **Initializes** the OpenAI client with an API key stored in `.env`.

---

3. Handling Incoming Requests

```
export async function POST(req) {
  try {
    const formData = await req.formData();
    const jobDescription = formData.get("jobDescription");
    const file = formData.get("file");
```

- Receives an HTTP `POST` request.
- Extracts:
  - `jobDescription` (text input from the user)
  - `file` (uploaded resume in `.docx`, `.txt`, or `.pdf` format)

3.1 Validating Inputs

```
    if (!jobDescription || jobDescription.trim().length < 50) {
      return NextResponse.json({ error: "Job description must be at least 50 characters long" }, { status: 400 });
    }

    if (!file) {
```

```
    return NextResponse.json({ error: "No file uploaded" }, { status: 400 });
  }
```

- **Ensures** that the job description is at least 50 characters long.
- **Checks** if a file is uploaded.

---

4. Extracting Text from the Resume

```
  const buffer = await file.arrayBuffer();
   const fileType = file.type;
   const extractedText = await extractTextFromFile(Buffer.from(buffer), fileType);
```

- **Converts** the file into a `Buffer` for processing.
- **Identifies** the file type.
- **Calls** `extractTextFromFile()` to extract text.

---

5. Extracting Text Based on File Type

```
async function extractTextFromFile(buffer, fileType) {
  try {
    if (fileType ===
"application/vnd.openxmlformats-officedocument.wordprocessingml.document") {
      const { value } = await mammoth.extractRawText({ buffer });
      return value.trim();
```

```
  } else if (fileType === "text/plain") {
    return buffer.toString("utf-8").trim();
  } else if (fileType === "application/pdf") {
    return new Promise((resolve, reject) => {
      let extractedText = "";
      new PdfReader().parseBuffer(buffer, (err, item) => {
        if (err) {
          console.error("Error extracting text from PDF:", err);
          return reject(err);
        }
        if (!item) {
          return resolve(extractedText.trim()); // End of file
        }
        if (item.text) {
          extractedText += item.text + " ";
        }
      });
    });
  } else {
    throw new Error("Unsupported file type. Only .docx, .txt, and .pdf are supported.");
  }
} catch (error) {
  console.error("Error extracting text from file:", error);
  return "";
```

```
  }
}
```

File Processing

- **DOCX (`.docx`)** → Uses `mammoth.extractRawText()` for extraction.
- **Plain Text (`.txt`)** → Directly converts `buffer` to a UTF-8 string.
- **PDF (`.pdf`)** → Uses `PdfReader()` to extract text.
- **Unsupported Formats** → Throws an error.

---

6. Generating AI-Powered Interview Questions

```
async function getOpenAIChatCompletion(jobDescription, extractedText) {
  return await openai.chat.completions.create({
    model: "gpt-4o-mini",
    messages: [
      {
        role: "system",
        content: `AI-Driven Interview Question Generation


        Objective:
        - Generate a structured set of interview questions based on the provided job description and candidate's resume.


        Methodology:
```

- Utilize AI to analyze both data sources.

- Ensure the AI considers both inputs to create highly relevant questions.

Output Format:

- Return an **array of objects** containing Two sets of interview questions.

- Each set should include three categories: **Technical, Behavioral, Situational**.

- Each category should contain an object with three difficulty levels: **easy, medium, hard**.

IMPORTANT:

TWO sets of Interview questions.

[{ "set": 1, "categories": [...] }, { "set": 2, "categories": [...] }]

- Ensure clarity and relevance of each question to the candidate’s skills and job role.`

},

{

role: "user",

content: `Job Description: ${jobDescription}\n\nExtracted Resume Text: ${extractedText}`

}

],

});

}

How It Works

- Calls OpenAI's `GPT-4o-mini` model to generate **interview questions**.
- Uses **job description + resume text** to ensure **relevant** questions.
- Requests **two sets of questions**, each categorized into:
    - **Technical**
    - **Behavioral**
    - **Situational**
- Each category has **three difficulty levels**:
    - Easy
    - Medium
    - Hard

---

7. Handling AI Response

```
  const aiResponse = await getOpenAIChatCompletion(jobDescription, extractedText);
   let rawJsonString = aiResponse.choices[0].message.content;

   // Remove Markdown code block markers if they exist
   rawJsonString = rawJsonString.replace(/```json|```/g, "").trim();

   try {
      const parsedJson = JSON.parse(rawJsonString);
      return NextResponse.json({ message: "Processing complete", parsedJson }, { status: 200
});
   } catch (error) {
```

```
    console.error("JSON Parsing Error:", error);

    return NextResponse.json({ error: "Invalid JSON format received from AI response" }, { status: 500 });

  }
```

Processing AI Output

- Extracts `JSON` from OpenAI's response.
- **Removes** any Markdown formatting (e.g., ```` ```json ```` or ```` ``` ````).
- **Attempts** to parse it into a JSON object.
- **Handles errors** if the JSON format is invalid.

---

8. Handling Errors

```
} catch (error) {

  console.error("Error processing request:", error);

  return NextResponse.json({ error: "Too Many Request Token Limit Exit!" }, { status: 429 });

 }
```

- **Catches unexpected errors**.
- **Returns HTTP 429** if the OpenAI request limit is exceeded.

Key Takeaways

1. **Accepts a job description & a resume file** via a `POST` request.
2. **Extracts text** from `.docx`, `.txt`, or `.pdf` resumes.
3. **Generates AI-powered interview questions** using OpenAI.
4. **Formats the response** into two structured question sets.
5. **Handles errors gracefully** (invalid input, API issues, JSON parsing errors).