This code is a **Next.js API route** that manages an **AI-powered interview process** by analyzing resumes and generating job-related questions using **OpenAI's GPT-4o-mini**. Let's break it down step by step.

Imports and Setup

```
import { NextResponse } from "next/server";
import mammoth from "mammoth";
import dotenv from "dotenv";
import { PdfReader } from "pdfreader";
import OpenAI from "openai";
// import axios from "axios";

dotenv.config(); // Load environment variables
```

- `NextResponse` → Used to send responses in Next.js API routes.
- `mammoth` → Extracts plain text from `.docx` (Word) files.
- `dotenv` → Loads environment variables from a `.env` file.
- `PdfReader` → Reads and extracts text from PDF files.
- `OpenAI` → Interacts with OpenAI's GPT API for AI-powered interview question generation.
- `dotenv.config()` → Ensures API keys and configurations are loaded from a `.env` file.

Global Variables

```
const openai = new OpenAI({ apiKey: process.env.OPEN_AI_API_KEY });
const sessions = new Map(); // Stores interview sessions
const MAX_QUESTIONS = 11; // Limit for the number of interview questions
```

- `openai` → Creates an OpenAI instance with the API key from environment variables.
- `sessions` → A `Map` to store interview sessions in memory.
- `MAX_QUESTIONS` → Limits each interview session to 11 questions.

POST Request: Handles File Upload and Starts the Interview

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

    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 });
    }

    const buffer = await file.arrayBuffer();
    const fileType = file.type;

    // Extract text from the PDF
    const extractedText = await extractTextFromFile(Buffer.from(buffer), fileType);

    if (!extractedText) {
      return NextResponse.json(
        { error: "Failed to extract text from PDF" },
        { status: 500 }
      );
    }
```

What Happens Here?

1. **Receives form data** with:
    - `"jobDescription"` (minimum 50 characters).
    - `"file"` (resume in PDF, DOCX, or TXT format).
2. **Validates inputs**:
    - If the job description is too short, returns a **400 error**.
    - If no file is uploaded, returns a **400 error**.
3. **Reads the file as a buffer** and extracts text from it.

Generating the First Interview Question

```
const sessionId = crypto.randomUUID();
const { Question, estimatedTime, ResponseTime } = await getOpenAIChatCompletion(
  jobDescription,
  extractedText,
  "easy",
  "technical",
  []
);

sessions.set(sessionId, {
  jobDescription,
  extractedText,
  questions: [{
    question: Question,
    userResponse: null,
    actualTime: estimatedTime,
    ResponseTime: ResponseTime
  }]
});

const encoder = new TextEncoder();
const stream = new ReadableStream({
  start(controller) {
```

```
      controller.enqueue(encoder.encode(JSON.stringify({ sessionId, question: Question }) +
"\n"));
      controller.close();
    }
  });

  return new Response(stream, {
    headers: { "Content-Type": "application/json" }
  });
} catch (error) {
  console.error("Error processing request:", error);
  return NextResponse.json({ error: "Too Many Request Token Limit Exit!" }, { status: 429 });
}
}
```

What Happens Here?

1. **Creates a new session ID** using `crypto.randomUUID()`.
2. Calls `getOpenAIChatCompletion()` to generate the **first question**.
3. **Stores the session** in `sessions`:
   - Saves `jobDescription`, `extractedText`, and the first question.
4. **Returns a JSON response** with:
   - `sessionId`
   - First interview `question`

PATCH Request: Handles User Responses and Generates the Next Question

```
export async function PATCH(req) {
  try {
    const { sessionId, answer } = await req.json();

    if (!sessionId || !sessions.has(sessionId)) {
```

```
    return NextResponse.json({ error: "Invalid session ID" }, { status: 400 });
  }

  if (!answer || answer.trim().length < 5) {
    return NextResponse.json({ error: "Answer must be at least 5 characters long" }, { status:
400 });
  }

  const sessionData = sessions.get(sessionId);
  let { jobDescription, extractedText, questions } = sessionData;
```

What Happens Here?

1. **Receives user input** (`sessionId` and `answer`).

2. **Validates inputs**:

    - Checks if `sessionId` exists in memory.

    - Ensures the answer is at least **5 characters long**.

Generating the Next Question

```
  if (questions.length >= MAX_QUESTIONS) {
    return NextResponse.json({ message: "Interview complete", sessionId, questions }, { status:
200 });
  }

  questions[questions.length - 1].userResponse = answer;
  questions[questions.length - 1]['CandidateAnsweredTime'] = new Date().toTimeString().split('
')[0];

  const nextDifficulty = getNextDifficulty(questions.length);
  const nextCategory = getNextCategory(questions.length);

  const { Question, estimatedTime, ResponseTime } = await getOpenAIChatCompletion(
    jobDescription,
    extractedText,
    nextDifficulty,
    nextCategory,
    questions
```

```
  );

  questions.push({
    question: Question,
    userResponse: null,
    actualTime: estimatedTime,
    ResponseTime: ResponseTime,
  });

  return NextResponse.json({ sessionId, questions });
```

- If the max number of questions (`MAX_QUESTIONS`) is reached, the interview **ends**.
- The last question's `userResponse` is updated.
- Determines the **next question's difficulty and category**.
- Calls `getOpenAIChatCompletion()` to get the **next question**.
- Stores the new question in the session.

Extracting Text from Resume Files

```
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) return reject(err);
          if (!item) return resolve(extractedText.trim());
          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 "";
  }
}
```

- **Extracts text from** `.docx`, `.txt`, and `.pdf` resumes.

Generating AI-Powered Interview Questions

```
async function getOpenAIChatCompletion(jobDescription, extractedText, difficulty, category, history) {
  const formattedHistory = history.map((q, i) => `Q${i + 1}: ${q.question}\nUser's Answer: ${q.userResponse || "(no answer)"}\n`).join("\n");

  const stream = await openai.chat.completions.create({
    model: "gpt-4o-mini",
    messages: [{ role: "system", content: "You are a structured AI interviewer." }, { role: "user", content: prompt }],
    stream: true,
  });

  let result = "";
  for await (const chunk of stream) {
    result += chunk.choices[0]?.delta?.content || "";
  }
```

- Uses OpenAI's API to generate **job-related** interview questions.
- Ensures questions **do not repeat**.

Conclusion

This API: ✅ **Extracts text** from resumes
✅ **Generates AI-powered interview questions**
✅ **Handles user responses** and adapts difficulty
✅ **Limits questions** to avoid overwhelming users