import os

import json

import zipfile

from flask import Flask, jsonify, request

from groq import Groq

import pandas as pd

import glob

class ResumeProcessor:

    def \_\_init\_\_(self, groq\_api\_key):

        self.groq\_api\_key = groq\_api\_key

        self.allowed\_extensions = {'pdf', 'docx', 'zip'}

    def extract\_text\_from\_docx(self, docx\_file):

        try:

            import docx

            doc = docx.Document(docx\_file)

            text = [para.text for para in doc.paragraphs]

            return '\n'.join(text)

        except Exception as e:

            raise ValueError(f"Failed to extract text from .docx: {e}")

    def extract\_text\_from\_pdf(self, pdf\_file):

        try:

            from PyPDF2 import PdfReader

            pdf\_text = []

            reader = PdfReader(pdf\_file)

            for page in reader.pages:

                pdf\_text.append(page.extract\_text())

            return '\n'.join(pdf\_text)

        except Exception as e:

            raise ValueError(f"Failed to extract text from .pdf: {e}")

    def extract\_text\_from\_resume(self, file\_path):

        if file\_path.endswith('.docx'):

            return self.extract\_text\_from\_docx(file\_path)

        elif file\_path.endswith('.pdf'):

            return self.extract\_text\_from\_pdf(file\_path)

        else:

            raise ValueError(f"Unsupported file format: {file\_path}")

    def process\_resumes(self, directory\_path):

        resumes\_text = {}

        file\_pattern = os.path.join(directory\_path, '\*')

        files = glob.glob(file\_pattern)

        for file\_path in files:

            if os.path.isfile(file\_path) and self.allowed\_file(file\_path):

                print(f"Processing file: {file\_path}")

                if file\_path.endswith('.zip'):

                    with zipfile.ZipFile(file\_path, 'r') as zip\_ref:

                        for zip\_info in zip\_ref.infolist():

                            if zip\_info.filename.endswith(('.docx', '.pdf')):

                                with zip\_ref.open(zip\_info) as extracted\_file:

                                    text = self.extract\_text\_from\_resume(extracted\_file)

                                    resumes\_text[zip\_info.filename] = text

                else:

                    text = self.extract\_text\_from\_resume(file\_path)

                    resumes\_text[os.path.basename(file\_path)] = text

        extracted\_data = self.extract\_information(resumes\_text)

        self.save\_to\_json(extracted\_data, 'extracted\_data.json')

        self.save\_to\_csv(extracted\_data, 'extracted\_data.csv')

        return extracted\_data

    def allowed\_file(self, filename):

        return '.' in filename and filename.rsplit('.', 1)[1].lower() in self.allowed\_extensions

    def extract\_information(self, resumes\_text):

        extracted\_data = []

        client = Groq(api\_key=self.groq\_api\_key)

        for filename, text in resumes\_text.items():

            prompt\_template = f'''

            You are an AI bot designed to extract contextual information from resumes.

            Please extract the following information from this resume:

            1. Full\_Name

            2. Year\_of\_Experience

            2. Address

            3. Email\_ID

            4. Contact\_Number

            5. Employment\_Details

            6. Industry\_sector

            7. Technical\_Skills

            8. Education\_Degree

            9. College\_Name

            10. Profile\_Summary

            11. Project\_Summary

            12. Certifications

            Resume text:

            {text}

            Only provide the JSON without any explanatory text or notes. If any information is not available, include `null` for that field or possible to calculate it.

            Don't provide any notes or explaination on your assumptions.

            Calculate Year\_of\_Experience using total duration of work experiance. if it's not directly available

            Industry\_sector is not directly mentioned in resume, use your knowledge and decide it.

            Profile\_Summary must be based on overall work realted experience and it shoud not be more than one paragraph

            '''

            chat\_completion = client.chat.completions.create(

                messages=[

                    {

                        "role": "system",

                        "content": prompt\_template,

                    }

                ],

                temperature=0,

                model="llama-3.1-8b-instant"

            )

            response\_content = chat\_completion.choices[0].message.content

            # Debugging step: Print the raw response content

            print(f"Raw API response for file '{filename}': {response\_content}")

            try:

                data = json.loads(response\_content.strip())

                extracted\_data.append(data)

            except json.JSONDecodeError as e:

                print(f"Failed to parse JSON: {e}")

                print(f"Response content: {response\_content}")

        return extracted\_data

    def save\_to\_json(self, data, json\_path):

*"""Save extracted data to a JSON file."""*

        with open(json\_path, 'w', encoding='utf-8') as json\_file:

            json.dump(data, json\_file, ensure\_ascii=False, indent=4)

        print(f"Extracted data saved to {json\_path}")

    def save\_to\_csv(self, data, csv\_path):

*"""Save extracted data to a CSV file."""*

        if data:

            df = pd.DataFrame(data)

            df.to\_csv(csv\_path, index=False, encoding='utf-8')

            print(f"Extracted data saved to {csv\_path}")

        else:

            print("No data available to save.")

app = Flask(\_\_name\_\_)

app.config['GROQ\_API\_KEY'] = 'gsk\_4uZAXTFO3YJ8rD0cF2dDWGdyb3FYH6aaPnuY66ndq8d5IEy009QQ'

processor = ResumeProcessor(app.config['GROQ\_API\_KEY'])

@app.route('/process\_directory', methods=['POST'])

def process\_directory():

    data = request.get\_json()

    directory\_path = data.get('directory\_path')

    if not directory\_path or not os.path.isdir(directory\_path):

        return jsonify({"error": "Invalid directory path"}), 400

    try:

        processed\_data = processor.process\_resumes(directory\_path)

        return jsonify(processed\_data), 200

    except Exception as e:

        return jsonify({"error": str(e)}), 500

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0',port=5000,debug=True)