import io

from flask import Flask, request, jsonify, send\_file

import os

from reportlab.pdfgen import canvas

from reportlab.lib.colors import red, green, yellow, black

from PyPDF2 import PdfReader, PdfWriter, PageObject

import requests

def add\_colored\_text\_to\_pdf(input\_path, output\_path, topics):

    """

    Create a new PDF with the original text colored based on the topics and their scores.

    Text with:

    - 'excellent' score will be colored green.

    - 'partial' score will be colored yellow.

    - 'none' score will be colored red.

    """

    # קריאת הקובץ המקורי

    reader = PdfReader(input\_path)

    # יצירת קובץ PDF חדש

    packet = io.BytesIO()

    can = canvas.Canvas(packet, pagesize=(595.27, 841.89))  # גודל עמוד A4

    can.setFont("Helvetica", 12)

    for page\_num, page in enumerate(reader.pages):

        print(f"Processing page {page\_num + 1}...")  # לוג לעמוד הנוכחי

        # חילוץ הטקסט מהעמוד

        text = page.extract\_text()

        if not text:

            print(f"No text found on page {page\_num + 1}. Skipping...")

            continue

        y\_position = 800  # מיקום אנכי להתחלת הטקסט

        for line in text.split("\n"):

            # ברירת מחדל: שחור

            color = black

            # בדיקת התאמה לנושאים

            for topic in topics:

                if topic["text"] in line:

                    if topic["score"] == "excellent":

                        color = green

                    elif topic["score"] == "partial":

                        color = yellow

                    elif topic["score"] == "none":

                        color = red

                    break  # מצאנו התאמה, אין צורך לבדוק עוד

            # הגדרת הצבע וכתיבת השורה

            can.setFillColor(color)

            can.drawString(50, y\_position, line)

            y\_position -= 15  # מעבר לשורה הבאה

            # אם נגמר המקום בדף, הוסף דף חדש

            if y\_position < 50:

                can.showPage()

                can.setFont("Helvetica", 12)

                y\_position = 800

        # סיום עיבוד העמוד הנוכחי

        can.showPage()

    # שמירת הקובץ החדש

    can.save()

    packet.seek(0)

    # כתיבת התוכן לקובץ PDF חדש

    with open(output\_path, "wb") as output\_file:

        output\_file.write(packet.getvalue())

    print(f"File with colored text saved to {output\_path}")

from PyPDF2 import PdfReader

def extract\_text\_from\_pdf(pdf\_path):

    """

    Extract text from a PDF file.

    Args:

        pdf\_path (str): Path to the PDF file.

    Returns:

        str: Extracted text from the PDF.

    """

    reader = PdfReader(pdf\_path)

    text = ""

    for page in reader.pages:

        text += page.extract\_text()

    return text

import base64

import json  # ייבוא ספריית JSON

from doctest import debug

import os

from google import genai

from google.genai import types

from dotenv import load\_dotenv

from google import genai

from google.genai import types

gemini\_api\_key = os.getenv('GEMINI\_API\_KEY', 'AIzaSyA1\_-pRQQz89muAzUCFH1AFPDxyNkG5ctI')

client = genai.Client(api\_key=gemini\_api\_key)

model = "gemini-2.0-flash"

def analyze\_resume\_with\_genai(resume\_text, topics):

    """

    Analyze the resume text using GenAI and classify the user's knowledge level for each topic.

    """

    prompt = (

        "Analyze the content of the resume. And for each line/section classify the content in the resume into a list of topics\n"

        "The topics are:\n" f"{', '.join([topic['text'] for topic in topics])}\n\n"

        "According to this classification, return an array containing original text from the resume, its classification, and the score"

        "topic is like: 'excellent', 'partial', or 'bad'.\n\n"

        "Resume Text:\n"

        f"{resume\_text}\n\n"

        "Return the result as a JSON array in the following format only if it is related to any topic"

        '[{"text": "original\_text", "knowledge": "excellent"}, ...]'

    )

    # Prepare the content for the GenAI model

    contents = [

        types.Content(

            role="user",

            parts=[

                types.Part(text=prompt)

            ]

        )

    ]

    # Configuration for the model

    generate\_content\_config = types.GenerateContentConfig(

        temperature=0.7,

        top\_p=0.95,

        top\_k=40,

        max\_output\_tokens=1024,

        response\_mime\_type="application/json"

    )

    # Generate the response

    response\_text = ""

    for chunk in client.models.generate\_content\_stream(

        model=model, contents=contents, config=generate\_content\_config

    ):

        response\_text += chunk.text

    # Ensure the response is valid JSON

    try:

        # Replace single quotes with double quotes if necessary

        # response\_text = response\_text.replace("'", '"')

        result = json.loads(response\_text.strip())

        print("Parsed result:", result)

        return result

    except json.JSONDecodeError as e:

        print(f"Error parsing GenAI response: {e}")

        return []

import fitz  # PyMuPDF

def highlight\_text\_in\_pdf(input\_pdf, output\_pdf, topics):

    """

    Highlight text in a PDF using PyMuPDF and save the result as a new PDF.

    """

    # פתיחת קובץ ה-PDF

    doc = fitz.open(input\_pdf)

    for page\_num in range(len(doc)):

        page = doc[page\_num]

        # חילוץ הטקסט והמיקומים

        words = page.get\_text("words")  # רשימת מילים עם מיקומים

        for word in words:

            x0, y0, x1, y1, text = word[:5]  # קואורדינטות וטקסט

            # בדיקת התאמה לנושאים

            color = None

            for topic in topics:

                if topic["text"] in text:

                    if topic["score"] == "excellent":

                        color = (0, 1, 0)  # ירוק (RGB נורמלי)

                    elif topic["score"] == "partial":

                        color = (1, 1, 0)  # צהוב

                    elif topic["score"] == "none":

                        color = (1, 0, 0)  # אדום

                    break

            # אם יש צבע, הוסף מלבן צבעוני מעל הטקסט

            if color:

                print("color",color)

                rect = fitz.Rect(x0, y0, x1, y1)

                page.draw\_rect(rect, color=color, fill=color, overlay=True)

    # שמירת הקובץ החדש

    doc.save(output\_pdf)

    print(f"Highlighted PDF saved to {output\_pdf}")

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

@app.route("/highlight\_resume", methods=["POST"])

def highlight\_resume():

    """

    הדגשת קובץ קורות החיים על בסיס הידע של המשתמש והחזרת קובץ PDF מעודכן.

    """

    print("Received request to highlight resume")

    data = request.json

    resume\_url = data.get("resume\_url")

    topics = data.get("topics", [])

    if not resume\_url or not topics:

        return jsonify({"error": "Resume URL and topics must be provided"}), 400

    temp\_input\_path = "temp\_resume.pdf"

    temp\_output\_path = "highlighted\_resume.pdf"

    try:

        # הורדת קובץ ה-PDF

        response = requests.get(resume\_url)

        if response.status\_code != 200:

            return jsonify({"error": "Failed to download the resume file"}), 500

        with open(temp\_input\_path, "wb") as file:

            file.write(response.content)

        # קריאת הטקסט מקובץ קורות החיים

        resume\_text = extract\_text\_from\_pdf(temp\_input\_path)

        # ניתוח קובץ קורות החיים באמצעות GenAI

        analyzed\_topics = analyze\_resume\_with\_genai(resume\_text, topics)

        print("Analyzed topics:", analyzed\_topics)

        # הוספת שכבת צבעים לקובץ המקורי

        highlight\_text\_in\_pdf(temp\_input\_path, temp\_output\_path, analyzed\_topics)

        # החזרת הקובץ למשתמש

        return send\_file(temp\_output\_path, as\_attachment=True)

    except Exception as e:

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

    finally:

        # מחיקת קבצים זמניים

        try:

            if os.path.exists(temp\_input\_path):

                os.remove(temp\_input\_path)

            if os.path.exists(temp\_output\_path):

                os.remove(temp\_output\_path)

        except Exception as cleanup\_error:

            print(f"Error during cleanup: {cleanup\_error}")

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(host="0.0.0.0", port=5000)