import json

from io import BytesIO

from azure.core.credentials import AzureKeyCredential

from azure.ai.documentintelligence import DocumentIntelligenceClient

from azure.ai.documentintelligence.models import DocumentAnalysisFeature

from openai import AzureOpenAI

# --- Azure OCR and Document Intelligence ---

AZURE\_OCR\_ENDPOINT = "https://venka.cognitiveservices.azure.com/"

AZURE\_OCR\_KEY = "FmA8C0l5UHynz2tNCdA2gMXmLJAOsCpjp2eWncyiPvHZHgAPe3QbJQQJ99BEACYeBjFXJ3w3AAALACOGYitX"

ocr\_client = DocumentIntelligenceClient(

    endpoint=AZURE\_OCR\_ENDPOINT,

    credential=AzureKeyCredential(AZURE\_OCR\_KEY)

)

# --- Azure OpenAI ---

AZURE\_OPENAI\_KEY = "36BLOdl5AfrHTpOWCwMwUHRwVJePxq6JAgVosopBPo0pGCfoOFyEJQQJ99BFACYeBjFXJ3w3AAABACOGlMcW"

AZURE\_OPENAI\_ENDPOINT = "https://venka-openai.openai.azure.com/"

AZURE\_OPENAI\_DEPLOYMENT = "gpt-4o"

AZURE\_OPENAI\_API\_VERSION = "2024-02-15-preview"

openai\_client = AzureOpenAI(

    api\_key=AZURE\_OPENAI\_KEY,

    api\_version=AZURE\_OPENAI\_API\_VERSION,

    azure\_endpoint=AZURE\_OPENAI\_ENDPOINT

)

# --- LLM Prompt ---

EXTRACTION\_PROMPT = """

You are a payroll document expert. From the following pay stub text, extract these fields:

1. TotalHoursWorked — the total of all hours worked across categories like Regular, Overtime, Holiday, Sick, etc. Give only float value. If not listed, return null.

2. AveragePayRate — Use all available hour-rate pairs and compute weighted average as (rate × hours) summed and divided by total hours. Return float values rounded to 2 decimal places. If only one rate exists, just return it.ONLY extract the hourly pay rate. ignore if it is monthly, daily pay rate. Give only float value. If not listed, return null.

3. JobTitle — the job title of the employee (like "Maintenance", "Driver"). If not listed, return null.

Return only \*\*pure JSON\*\*, no extra formatting, no explanations, no markdown code blocks. Output \*\*exactly like this\*\*:

{{"TotalHoursWorked": "...", "AveragePayRate": "...", "JobTitle": "..."}}

Here is the pay stub text:

\"\"\"{text}\"\"\"

JSON:

"""

# --- OCR Text Extraction ---

def extract\_text\_from\_ocr(file\_bytes: bytes) -> str:

    poller = ocr\_client.begin\_analyze\_document(

        model\_id="prebuilt-read",

        body=BytesIO(file\_bytes)

    )

    result = poller.result()

    return "\n".join([line.content for page in result.pages for line in page.lines])

# --- LLM Field Extraction ---

def extract\_llm\_fields(text: str) -> dict:

    prompt = EXTRACTION\_PROMPT.format(text=text[:1500])

    try:

        response = openai\_client.chat.completions.create(

            model=AZURE\_OPENAI\_DEPLOYMENT,

            messages=[

                {"role": "system", "content": "You are a helpful assistant that extracts fields from text."},

                {"role": "user", "content": prompt}

            ],

            temperature=0.2

        )

        raw = response.choices[0].message.content.strip()

        parsed = json.loads(raw)

        return {

            "PayRate": {

                "value": parsed.get("AveragePayRate", None),

                "confidence": 100.0

            },

            "HoursWorked": {

                "value": parsed.get("TotalHoursWorked", None),

                "confidence": 100.0

            },

            "JobTitle": {

                "value": parsed.get("JobTitle", None),

                "confidence": 100.0

            }

        }

    except Exception as e:

        print("❌ LLM extraction failed:", e)

        return {

            "PayRate": {"value": None, "confidence": None},

            "HoursWorked": {"value": None, "confidence": None},

            "JobTitle": {"value": None, "confidence": None}

        }

# --- Main Function ---

def process\_pay\_stub(file\_bytes: bytes, filename: str) -> dict:

    # Step 1: Extract all regular fields using Azure payStub model

    doc\_poller = ocr\_client.begin\_analyze\_document(

        model\_id="prebuilt-payStub.us",

        body=BytesIO(file\_bytes),

        content\_type="application/octet-stream",

        features=[DocumentAnalysisFeature.QUERY\_FIELDS]

    )

    doc\_result = doc\_poller.result()

    extracted\_fields = {}

    doc = doc\_result.documents[0] if doc\_result.documents else None

    if not doc:

        return {"status": "error", "message": "No document returned by Azure."}

    for key, field in doc.fields.items():

        extracted\_fields[key] = {

            "value": field.content if field else None,

            "confidence": round(field.confidence \* 100, 2) if field and field.confidence else None

        }

    # Step 2: Extract 3 LLM-based fields (JobTitle, PayRate, HoursWorked)

    try:

        text = extract\_text\_from\_ocr(file\_bytes)

        llm\_fields = extract\_llm\_fields(text)

        extracted\_fields.update(llm\_fields)

    except Exception as e:

        print("LLM step failed:", e)

    return {

        "status": "success",

        "filename": filename,

        "extracted\_fields": extracted\_fields

    }

# Optional CLI test

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

    with open("sample\_pay\_stub.png", "rb") as f:

        file\_bytes = f.read()

        result = process\_pay\_stub(file\_bytes, "sample\_pay\_stub.png")

        print(json.dumps(result, indent=2))

from azure.core.credentials import AzureKeyCredential

from azure.ai.documentintelligence import DocumentIntelligenceClient

from azure.ai.documentintelligence.models import DocumentAnalysisFeature

from io import BytesIO

# 🔐 Azure credentials

AZURE\_ENDPOINT = "https://venka.cognitiveservices.azure.com/"

AZURE\_KEY = "FmA8C0l5UHynz2tNCdA2gMXmLJAOsCpjp2eWncyiPvHzHgAPe3QbJQQJ99BEACYeBjFXJ3w3AAALACOGYitX"

# 🔧 Azure client

client = DocumentIntelligenceClient(

    endpoint=AZURE\_ENDPOINT,

    credential=AzureKeyCredential(AZURE\_KEY)

)

def process\_pay\_stub(file\_bytes: bytes, filename: str) -> dict:

    # ✅ Define safe query field names

    query\_fields = ["PayRate", "HoursWorked", "JobTitle"]

    # 📤 Send document to Azure

    poller = client.begin\_analyze\_document(

        model\_id="prebuilt-payStub.us",

        body=BytesIO(file\_bytes),

        content\_type="application/octet-stream",

        features=[DocumentAnalysisFeature.QUERY\_FIELDS],

        query\_fields=query\_fields

    )

    result = poller.result()

    # 🧾 Extracted data

    extracted = {}

    doc = result.documents[0] if result.documents else None

    if not doc:

        return {

            "status": "error",

            "message": "No document returned by Azure."

        }

    # ✅ Regular + query fields all live in `doc.fields`

    for key, field in doc.fields.items():

        extracted[key] = {

            "value": field.content if field else None,

            "confidence": round(field.confidence \* 100, 2) if field and field.confidence else None

        }

    return {

        "status": "success",

        "filename": filename,

        "extracted\_fields": extracted

    }

