import csv

import os

import asyncio

import aiohttp

import sqlite3

from PIL import Image

from io import BytesIO

import uuid

import requests

# Step 1: Define the directory to store images

output\_dir = "processed\_images"

os.makedirs(output\_dir, exist\_ok=True)

# Step 2: Set up the SQLite database

def setup\_database(db\_name="image\_data.db"):

conn = sqlite3.connect(db\_name)

cursor = conn.cursor()

# Create table for storing image data

cursor.execute('''

CREATE TABLE IF NOT EXISTS images (

id INTEGER PRIMARY KEY AUTOINCREMENT,

request\_id TEXT,

serial\_number INTEGER,

product\_name TEXT,

input\_image\_urls TEXT,

output\_image\_urls TEXT,

status TEXT

)

''')

# Create table for storing request status

cursor.execute('''

CREATE TABLE IF NOT EXISTS requests (

request\_id TEXT PRIMARY KEY,

file\_name TEXT,

status TEXT

)

''')

conn.commit()

return conn

# Step 3: Function to validate the CSV row data

def validate\_row(row):

required\_columns = ['S. No.', 'Product Name', 'Input Image Urls']

for col in required\_columns:

if col not in row or not row[col].strip():

print(f"Validation error: Missing or empty field '{col}' in row {row}")

return False

if not row['S. No.'].isdigit():

print(f"Validation error: 'S. No.' must be an integer in row {row}")

return False

if not row['Product Name'].strip():

print(f"Validation error: 'Product Name' cannot be empty in row {row}")

return False

image\_urls = row['Input Image Urls'].split(',')

for url in image\_urls:

if not url.strip().startswith(('http://', 'https://')):

print(f"Validation error: Invalid URL '{url}' in row {row}")

return False

return True

# Step 4: Function to download and process a single image asynchronously

async def download\_and\_process\_image(session, url, product\_name, index, output\_dir):

try:

async with session.get(url.strip()) as response:

if response.status == 200:

img\_data = await response.read()

img = Image.open(BytesIO(img\_data))

img = img.resize((256, 256)) # Resizing the image

output\_path = os.path.join(output\_dir, f"{product\_name}\_{index+1}.jpg")

img.save(output\_path, quality=50) # Compressing and saving the image

return output\_path

else:

print(f"Failed to download image: {url}")

return None

except Exception as e:

print(f"Error processing image {url}: {e}")

return None

# Step 5: Function to save processed image data to the database

def save\_to\_database(conn, request\_id, serial\_number, product\_name, input\_image\_urls, output\_image\_urls):

cursor = conn.cursor()

cursor.execute('''

INSERT INTO images (request\_id, serial\_number, product\_name, input\_image\_urls, output\_image\_urls, status)

VALUES (?, ?, ?, ?, ?, ?)

''', (request\_id, serial\_number, product\_name, input\_image\_urls, output\_image\_urls, "processed"))

conn.commit()

# Step 6: Function to update request status in the database

def update\_request\_status(conn, request\_id, status):

cursor = conn.cursor()

cursor.execute('''

UPDATE requests SET status = ? WHERE request\_id = ?

''', (status, request\_id))

conn.commit()

# Step 7: Function to trigger webhook

def trigger\_webhook(request\_id, webhook\_url, output\_file):

with open(output\_file, 'rb') as file:

response = requests.post(webhook\_url, files={'file': file}, data={'request\_id': request\_id})

print(f"Webhook triggered with response: {response.status\_code}")

# Step 8: Function to generate output CSV

def generate\_output\_csv(conn, request\_id, output\_file):

cursor = conn.cursor()

cursor.execute('''

SELECT serial\_number, product\_name, input\_image\_urls, output\_image\_urls

FROM images WHERE request\_id = ?

''', (request\_id,))

rows = cursor.fetchall()

with open(output\_file, 'w', newline='', encoding='utf-8') as csvfile:

writer = csv.writer(csvfile)

writer.writerow(['S. No.', 'Product Name', 'Input Image Urls', 'Output Image Urls'])

for row in rows:

writer.writerow(row)

# Step 9: Function to process images for a single row asynchronously

async def process\_images\_for\_row(row, output\_dir, conn, request\_id):

if not validate\_row(row):

print(f"Skipping row due to validation errors: {row}")

return

serial\_number = int(row['S. No.'])

product\_name = row['Product Name']

image\_urls = row['Input Image Urls'].split(',')

async with aiohttp.ClientSession() as session:

tasks = [

download\_and\_process\_image(session, url, product\_name, i, output\_dir)

for i, url in enumerate(image\_urls)

]

processed\_image\_paths = await asyncio.gather(\*tasks)

processed\_image\_paths = [path for path in processed\_image\_paths if path]

print(f"Processed images for {product\_name}: {processed\_image\_paths}")

# Save the processed images' data to the database

save\_to\_database(conn, request\_id, serial\_number, product\_name, ','.join(image\_urls), ','.join(processed\_image\_paths))

# Step 10: Process CSV and handle images asynchronously

async def process\_csv(file\_path, conn, request\_id, webhook\_url):

with open(file\_path, newline='', encoding='utf-8') as csvfile:

reader = csv.DictReader(csvfile)

tasks = [process\_images\_for\_row(row, output\_dir, conn, request\_id) for row in reader]

await asyncio.gather(\*tasks)

# Mark the request as completed

update\_request\_status(conn, request\_id, "completed")

# Generate the output CSV

output\_file = os.path.join(output\_dir, f"{request\_id}\_output.csv")

generate\_output\_csv(conn, request\_id, output\_file)

# Trigger the webhook

if webhook\_url:

trigger\_webhook(request\_id, webhook\_url, output\_file)

# Step 11: API to submit CSV file for processing and return a request ID

def submit\_csv(file\_path, webhook\_url=None):

conn = setup\_database()

request\_id = str(uuid.uuid4()) # Generate a unique request ID

cursor = conn.cursor()

cursor.execute('''

INSERT INTO requests (request\_id, file\_name, status)

VALUES (?, ?, ?)

''', (request\_id, os.path.basename(file\_path), "processing"))

conn.commit()

print(f"Request ID: {request\_id} - Processing started")

asyncio.run(process\_csv(file\_path, conn, request\_id, webhook\_url))

conn.close()

return request\_id

# Step 12: API to check processing status using request ID

def check\_status(request\_id):

conn = setup\_database()

cursor = conn.cursor()

cursor.execute('''

SELECT status FROM requests WHERE request\_id = ?

''', (request\_id,))

result = cursor.fetchone()

conn.close()

if result:

return f"Request ID: {request\_id} - Status: {result[0]}"

else:

return f"Request ID: {request\_id} not found."

# Example usage:

# Submit a CSV file and get a request ID

csv\_file\_path = 'input.csv' # Replace with the path to your CSV file

webhook\_url = 'https://your-webhook-url.com/endpoint' # Replace with your webhook URL

request\_id = submit\_csv(csv\_file\_path, webhook\_url)

# Check the status of the processing using the request ID

status = check\_status(request\_id)

print(status)