import os  
import sys  
  
current\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))  
print(current\_path)  
root\_path = os.path.abspath(os.path.join(current\_path,'..','..'))  
print(root\_path)  
sys.path.append(root\_path)  
  
import cv2  
import imutils  
from PIL import Image  
from skimage.metrics import structural\_similarity  
import numpy as np  
import pandas as pd  
from openpyxl.styles import Font, PatternFill  
from openpyxl import load\_workbook  
import openpyxl  
import time  
from Xcro.Utils.Escrow.read\_properties import Path\_Utils\_  
  
def imageDiff(baselinedImage, currentImage, flowname):  
 try:  
 if dimensionCompare(baselinedImage, currentImage) == True:  
 path = f'{Path\_Utils\_.COMPAREFOLDER}{flowname}.png'  
 # print(path)  
 original = cv2.imread(baselinedImage)  
 new = cv2.imread(currentImage)  
 original = imutils.resize(original, height=500)  
 new = imutils.resize(new, height=500)  
 diff = original.copy()  
 cv2.absdiff(original, new, diff)  
 gray = cv2.cvtColor(diff, cv2.COLOR\_BGR2GRAY)  
 for i in range(0, 3):  
 dilated = cv2.dilate(gray.copy(), None, iterations=i + 1)  
 (T, thresh) = cv2.threshold(dilated, 3, 255, cv2.THRESH\_BINARY)  
 cnts = cv2.findContours(thresh, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  
 cnts = imutils.grab\_contours(cnts)  
 for c in cnts:  
 (x, y, w, h) = cv2.boundingRect(c)  
 cv2.rectangle(new, (x, y), (x + w, y + h), (0, 255, 0), 2)  
 cv2.imwrite(path, new)  
 # return the image path  
 return path  
 except FileNotFoundError:  
 print("File not found !!")  
 except IsADirectoryError:  
 print("Directory not found!!!")  
  
  
def imageCompare(baselinedImage, currentImage):  
 try:  
 if dimensionCompare(baselinedImage, currentImage) == True:  
 baselinedImage = cv2.imread(baselinedImage)  
 currentImage = cv2.imread(currentImage)  
 first\_gray = cv2.cvtColor(baselinedImage, cv2.COLOR\_BGR2GRAY)  
 second\_gray = cv2.cvtColor(currentImage, cv2.COLOR\_BGR2GRAY)  
 score, diff = structural\_similarity(first\_gray, second\_gray, full=True)  
 perc1 = ("{:.4f}%".format(score \* 100))  
 # perc2 = score \* 100  
 # print(str(perc1))  
 # diff = (diff \* 255).astype("uint8")  
 # thresh = cv2.threshold(diff, 0, 255, cv2.THRESH\_BINARY\_INV | cv2.THRESH\_OTSU)[1]  
 # contours = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # contours = contours[0] if len(contours) == 2 else contours[1]  
 # mask = np.zeros(baselinedImage.shape, dtype='uint8')  
 # filled = currentImage.copy()  
 # for c in contours:  
 # area = cv2.contourArea(c)  
 # if area > 100:  
 # x, y, w, h = cv2.boundingRect(c)  
 # cv2.rectangle(baselinedImage, (x, y), (x + w, y + h), (36, 255, 12), 2)  
 # cv2.rectangle(currentImage, (x, y), (x + w, y + h), (36, 255, 12), 2)  
 # cv2.drawContours(mask, [c], 0, (0, 255, 0), -1)  
 # cv2.drawContours(filled, [c], 0, (0, 255, 0), -1)  
 return perc1  
 except FileNotFoundError:  
 print("File not found !!")  
 except IsADirectoryError:  
 print("Directory not found!!!")  
  
  
def mergeImages(baselinedImage, currentImage, flowname):  
 try:  
 if dimensionCompare(baselinedImage, currentImage) == True:  
 if imageCompare(baselinedImage, currentImage) != '100.0000%':  
 img\_01 = Image.open(baselinedImage)  
 img\_02 = Image.open(currentImage)  
 # print(imageDiff(baselinedImage, currentImage, flowname))  
 img\_03 = Image.open(imageDiff(baselinedImage, currentImage, flowname))  
  
 # Resizing the merged image to that of the original ones  
 img\_03 = img\_03.resize((img\_01.size[0], img\_01.size[1]))  
  
 img\_01\_size = img\_01.size  
 img\_02\_size = img\_02.size  
 img\_03\_size = img\_03.size  
  
 new\_im = Image.new('RGB', (max(img\_01\_size[0], img\_02\_size[0], img\_03\_size[0]),  
 img\_01\_size[1] + img\_02\_size[1] + img\_03\_size[1]), (250, 250, 250))  
 new\_im.paste(img\_01, (0, 0))  
 new\_im.paste(img\_02, (0, img\_01\_size[1]))  
 new\_im.paste(img\_03, (0, img\_01\_size[1] + img\_02\_size[1]))  
  
 # Saving the image  
 new\_im.save(f"{Path\_Utils\_.COMPAREFOLDER}" + flowname + ".png", "PNG")  
 print("Validation for : " + flowname + " with " + str(  
 imageCompare(baselinedImage, currentImage)) + " Image matched")  
 else:  
 img\_01 = Image.open(baselinedImage)  
 img\_02 = Image.open(currentImage)  
 img\_01\_size = img\_01.size  
 img\_02\_size = img\_02.size  
 new\_im = Image.new('RGB', (max(img\_01\_size[0], img\_02\_size[0]),  
 img\_01\_size[1] + img\_02\_size[1]), (250, 250, 250))  
 new\_im.paste(img\_01, (0, 0))  
 new\_im.paste(img\_02, (0, img\_01\_size[1]))  
 new\_im.save(f"{Path\_Utils\_.COMPAREFOLDER}" + flowname + ".png", "PNG")  
 print("Validation for : " + flowname + " with " + str(  
 imageCompare(baselinedImage, currentImage)) + " Image matched")  
 return str(imageCompare(baselinedImage,currentImage))  
 else:  
 print('Input images must have the same dimensions.')  
 except FileNotFoundError:  
 print("File not found !!")  
 except IsADirectoryError:  
 print("Directory not found!!!")  
 except Exception as e:  
 print(str(e))  
  
def appendResults(path):  
 try:  
 excel\_file = path  
 wb = load\_workbook(excel\_file)  
  
 # Read the file path from the Results Sheet in the Excel file  
 df = pd.read\_excel(excel\_file, sheet\_name='Results')  
  
 for index, row in df.iterrows():  
 file\_path = row['File\_path']  
 sheet\_name = row['Flowname']  
  
 # Create a new sheet with the given sheet\_name  
 ws = wb.create\_sheet(sheet\_name)  
  
 # Insert the image into the sheet using the file\_path  
 img = openpyxl.drawing.image.Image(f"./{Path\_Utils\_.COMPAREFOLDER}/{sheet\_name}.png")  
 ws.add\_image(img, 'A1')  
  
 # Styling  
 header\_font = Font(bold=True, color='FFFFFF', b=True)  
 header\_fill = PatternFill(start\_color='000080', end\_color='000080', fill\_type='solid')  
  
 sheet1 = wb['Results']  
  
 for cell in sheet1['1']:  
 cell.font = header\_font  
 cell.fill = header\_fill  
  
 # Save the modified workbook  
 wb.save(excel\_file)  
  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
def dimensionCompare(baselinedImage, currentImage):  
 baselinedImage = cv2.imread(baselinedImage)  
 currentImage = cv2.imread(currentImage)  
 if baselinedImage.size == currentImage.size:  
 return True  
 print(f"Baseline image size is {baselinedImage.size}")  
 print(f"Current Image size is {currentImage.size}")  
  
def reportGeneration():  
  
 fields = """<style>  
 table, td {  
 border-collapse: collapse;  
 border-bottom: 1px solid #ddd;  
 font-family: "Calibri";  
 }  
 tr:hover {background-color: coral;};  
 th {  
 background-color: #04AA6D;  
 color: white;  
 }  
 </style>"""  
  
 wb = pd.read\_excel(  
 'baselinedImages/baselinedImages\_details.xlsx') # This reads in your excel doc as a pandas DataFrame  
 wb.to\_html('baselinedImages/Report.html', index=False)  
  
 # Append-adds at last  
 file1 = open("baselinedImages/Report.html", "a") # append mode  
 file1.write(fields)  
 file1.close()  
  
from PIL import Image  
  
def compress\_image(image\_path, output\_path, quality):  
  
 with Image.open(image\_path) as img:  
  
 rgb\_img = img.convert('RGB')  
 rgb\_img.save(output\_path, "JPEG", quality=quality)  
 print(f"{image\_path} compressed successfully with {quality} Quality")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Start the timer  
 start\_time = time.time()  
  
 # Load the excel file  
 excel\_file = Path\_Utils\_.RESULTEXCEL  
 df = pd.read\_excel(excel\_file)  
 # Convert result columnt to string type  
 df['Result'] = df['Result'].astype(str)  
  
 # Read the excel file  
 for index, row in df.iterrows():  
 file\_path = row['File\_path']  
 flowname = row['Flowname']  
  
 baselinedImage = f"{Path\_Utils\_.BASEFOLDER}/{file\_path}"  
 currentImage = f"{Path\_Utils\_.CURRENTFOLDER}/{file\_path}"  
  
 result = mergeImages(baselinedImage, currentImage, flowname)  
 # reportGeneration()  
 df.at[index, 'Result'] = result  
  
 # Save the updated DataFrame with the "Result" column  
 df.to\_excel(excel\_file, index=False,engine='openpyxl',sheet\_name="Results")  
 appendResults(path=Path\_Utils\_.RESULTEXCEL)  
  
 # End time  
 end\_time = time.time()  
 print(f"{int(end\_time-start\_time)} seconds were taken for the validation")