import cv2

import face\_recognition

import numpy as np

import pandas as pd

from datetime import datetime, timedelta

import time

import re

from threading import Lock

import os

import pickle

from multiprocessing import Pool, cpu\_count

from functools import partial

# ========================

# CONFIGURATION SETTINGS

# ========================

ENCODING\_FILE = 'Encoding File.p'

IMAGES\_FOLDER = 'images'

MAX\_IMAGE\_DIMENSION = 800

FACE\_DETECTION\_MODEL = 'hog'

NUM\_ENCODING\_JITTERS = 1

modeFolderPath = "Modes"

imagebackground = cv2.imread('Resources/background.png')

HOLIDAYS = ['23-03-2025', '01-05-2025', '14-08-2025']

FRAME\_WIDTH, FRAME\_HEIGHT = 640, 480

MODE\_DISPLAY\_DURATION = 5

FACE\_MATCH\_THRESHOLD = 0.4

ATTENDANCE\_START\_TIME = datetime.strptime("6:00:00 AM", "%I:%M:%S %p").time()

ATTENDANCE\_END\_TIME = datetime.strptime("11:00:00 AM", "%I:%M:%S %p").time()

COOLDOWN\_SECONDS = 30

EXCEL\_LOCK = Lock()

# ========================

# GLOBAL STATE VARIABLES

# ========================

last\_seen = {}

detected\_students = set()

cooldown\_tracker = {}

student\_data = {}

current\_mode = 0

mode\_start\_time = None

current\_student\_id = None

last\_date\_check = datetime.now()

founded\_encodings = []

stud\_ID = []

# ========================

# ENCODING GENERATION FUNCTIONS

# ========================

def process\_single\_image(images\_folder, filename):

try:

filepath = os.path.join(images\_folder, filename)

img = cv2.imread(filepath)

if img is None:

print(f"⚠️ Couldn't read: {filename}")

return None

img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

h, w = img\_rgb.shape[:2]

scale = MAX\_IMAGE\_DIMENSION / max(h, w)

if scale < 1:

img\_rgb = cv2.resize(img\_rgb, (int(w \* scale), int(h \* scale)), interpolation=cv2.INTER\_AREA)

face\_locations = face\_recognition.face\_locations(img\_rgb, model=FACE\_DETECTION\_MODEL)

if not face\_locations:

print(f"🚫 No face in {filename}")

return None

largest\_face = max(face\_locations, key=lambda loc: (loc[2]-loc[0])\*(loc[1]-loc[3]))

encoding = face\_recognition.face\_encodings(

img\_rgb,

known\_face\_locations=[largest\_face],

num\_jitters=NUM\_ENCODING\_JITTERS

)[0]

return (encoding, os.path.splitext(filename)[0])

except Exception as e:

print(f"Error processing {filename}: {str(e)}")

return None

def batch\_process\_images(images\_folder, filenames):

processor\_count = min(cpu\_count(), 8)

process\_task = partial(process\_single\_image, images\_folder)

with Pool(processes=processor\_count) as pool:

results = pool.map(process\_task, filenames)

successful\_results = [r for r in results if r is not None]

if not successful\_results:

return [], []

encodings, ids = zip(\*successful\_results)

return list(encodings), list(ids) # Convert tuples to lists

def check\_and\_update\_encodings():

if not os.path.exists(ENCODING\_FILE):

print("Encoding file not found. Generating from all images...")

image\_files = os.listdir(IMAGES\_FOLDER)

encodings, ids = batch\_process\_images(IMAGES\_FOLDER, image\_files)

data = {'encodings': encodings, 'ids': ids}

with open(ENCODING\_FILE, 'wb') as f:

pickle.dump(data, f)

print(f"Encodings saved to {ENCODING\_FILE}")

else:

with open(ENCODING\_FILE, 'rb') as f:

data = pickle.load(f)

# Ensure encodings and ids are lists (legacy files might have tuples)

if isinstance(data['encodings'], tuple):

data['encodings'] = list(data['encodings'])

if isinstance(data['ids'], tuple):

data['ids'] = list(data['ids'])

existing\_ids = set(data['ids'])

current\_images = os.listdir(IMAGES\_FOLDER)

current\_ids = {os.path.splitext(img)[0] for img in current\_images}

new\_images = [img for img in current\_images if os.path.splitext(img)[0] not in existing\_ids]

if new\_images:

print(f"Found {len(new\_images)} new images. Updating encoding file...")

new\_encodings, new\_ids = batch\_process\_images(IMAGES\_FOLDER, new\_images)

data['encodings'].extend(new\_encodings)

data['ids'].extend(new\_ids)

with open(ENCODING\_FILE, 'wb') as f:

pickle.dump(data, f)

print(f"Updated encoding file with {len(new\_ids)} new entries.")

# ========================

# INITIALIZATION FUNCTIONS

# ========================

def load\_encodings():

global founded\_encodings, stud\_ID

with open(ENCODING\_FILE, 'rb') as encode\_file:

encodings\_dict = pickle.load(encode\_file)

founded\_encodings, stud\_ID = encodings\_dict['encodings'], encodings\_dict['ids']

founded\_encodings = np.array(founded\_encodings, dtype=np.float64)

def load\_mode\_images():

global imgModeList

if not os.path.exists(modeFolderPath):

raise FileNotFoundError(f"Directory not found: {modeFolderPath}")

mode\_files = sorted([f for f in os.listdir(modeFolderPath) if re.match(r"^mode[0-3]\.png$", f)], key=lambda x: int(x[4:-4]))

if len(mode\_files) != 4:

raise ValueError(f"Need exactly 4 mode images. Found {len(mode\_files)}")

imgModeList = [cv2.imread(os.path.join(modeFolderPath, f)) for f in mode\_files]

# def load\_mode\_images():

# global imgModeList

# if not os.path.exists(modeFolderPath):

# raise FileNotFoundError(f"Directory not found: {modeFolderPath}")

# mode\_files = sorted([f for f in os.listdir(modeFolderPath) if re.match(r"^mode[0-3]\.png$", f)],

# key=lambda x: int(x[4:-4]))

# if len(mode\_files) != 4:

# raise ValueError(f"Need exactly 4 mode images. Found {len(mode\_files)}")

# imgModeList = [cv2.imread(os.path.join(modeFolderPath, f)) for f in mode\_files]

# ========================

# ATTENDANCE CORE FUNCTIONS

# ========================

def clean\_columns(df):

df.rename(columns=lambda x: x.strip(), inplace=True)

required\_columns = ['Student ID', 'Name', 'Total Attendance']

for col in required\_columns:

if col not in df.columns:

raise KeyError(f"Missing column: '{col}'")

return df

def mark\_off\_if\_needed(excel\_file\_path):

today\_date = datetime.now().strftime('%d-%m-%Y')

weekday = datetime.now().weekday()

if weekday >= 5 or today\_date in HOLIDAYS:

with EXCEL\_LOCK:

df = pd.read\_excel(excel\_file\_path)

df = clean\_columns(df)

status\_col = f'{today\_date} Status'

time\_col = f'{today\_date} Time'

if status\_col not in df.columns:

df[status\_col] = 'Off'

df[time\_col] = ''

df.to\_excel(excel\_file\_path, index=False)

print(f"Marked Off: {excel\_file\_path}")

return True

return False

def mark\_absentees\_before\_attendance(excel\_file\_path):

today\_date = datetime.now().strftime('%d-%m-%Y')

with EXCEL\_LOCK:

df = pd.read\_excel(excel\_file\_path)

df = clean\_columns(df)

status\_col = f'{today\_date} Status'

time\_col = f'{today\_date} Time'

if status\_col not in df.columns:

df[status\_col] = 'A'

df[time\_col] = '00:00:00'

updated = True

else:

updated = False

if updated:

df.to\_excel(excel\_file\_path, index=False)

print(f"Absentees marked: {excel\_file\_path}")

def preload\_detected\_students():

global detected\_students

today\_date = datetime.now().strftime('%d-%m-%Y')

for excel\_path in STEP\_FILES.values():

try:

with EXCEL\_LOCK:

df = pd.read\_excel(excel\_path)

df = clean\_columns(df)

status\_col = f'{today\_date} Status'

if status\_col in df.columns:

present\_students = df[df[status\_col] == 'P']['Student ID'].astype(str).tolist()

detected\_students.update(present\_students)

print(f"Preloaded detected students from {excel\_path}: {present\_students}")

except Exception as e:

print(f"Error preloading detected students from {excel\_path}: {e}")

def is\_already\_present(student\_id):

"""Return True if attendance for today is already marked as 'P'."""

try:

excel\_path = get\_excel\_path(student\_id)

current\_time = datetime.now()

today\_date = current\_time.strftime('%d-%m-%Y')

with EXCEL\_LOCK:

df = pd.read\_excel(excel\_path)

df = clean\_columns(df)

status\_col = f'{today\_date} Status'

if status\_col in df.columns:

student\_row = df[df['Student ID'] == student\_id]

if not student\_row.empty and student\_row.iloc[0][status\_col] == 'P':

return True

return False

except Exception as e:

print(f"Error checking attendance for {student\_id}: {e}")

return False

def MarkAttendance(student\_id):

global detected\_students

with EXCEL\_LOCK:

current\_time = datetime.now()

if not (ATTENDANCE\_START\_TIME <= current\_time.time() <= ATTENDANCE\_END\_TIME):

print(f"Attendance closed for {student\_id}")

return

excel\_path = get\_excel\_path(student\_id)

today\_date = current\_time.strftime('%d-%m-%Y')

try:

df = pd.read\_excel(excel\_path)

df = clean\_columns(df)

df['Student ID'] = df['Student ID'].astype(str)

status\_col = f'{today\_date} Status'

time\_col = f'{today\_date} Time'

if status\_col not in df.columns:

df[status\_col] = 'A'

df[time\_col] = '00:00:00'

student\_row = df[df['Student ID'] == student\_id]

if student\_row.empty:

print(f"Student {student\_id} not found")

return

idx = student\_row.index[0]

if df.at[idx, status\_col] == 'P':

return

df.at[idx, status\_col] = 'P'

df.at[idx, time\_col] = current\_time.strftime("%I:%M:%S %p")

df.at[idx, 'Total Attendance'] += 1

df.to\_excel(excel\_path, index=False)

detected\_students.add(student\_id)

print(f"Marked Present: {student\_id}")

except Exception as e:

print(f"Excel Error: {str(e)}")

def LoadStudentName(student\_id):

try:

excel\_path = get\_excel\_path(student\_id)

with EXCEL\_LOCK:

df = pd.read\_excel(excel\_path)

df = clean\_columns(df)

row = df[df['Student ID'] == student\_id].iloc[0]

return row['Name'], row['Total Attendance']

except:

return f"Student {student\_id}", 0

# ========================

# FILE PATH CONFIGURATION

# ========================

STEP\_FILES = {

"FY10": "C:/Face-Detection-System/stud\_data/FY1.xlsx",

"FY20": "C:/Face-Detection-System/stud\_data/FY2.xlsx",

"FY30": "C:/Face-Detection-System/stud\_data/FY3.xlsx",

"ST10": "C:/Face-Detection-System/stud\_data/ST1.xlsx",

"ST20": "C:/Face-Detection-System/stud\_data/ST2.xlsx",

"ST30": "C:/Face-Detection-System/stud\_data/ST3.xlsx",

"ST40": "C:/Face-Detection-System/stud\_data/ST4.xlsx",

"ST50": "C:/Face-Detection-System/stud\_data/ST5.xlsx",

"ST60": "C:/Face-Detection-System/stud\_data/ST6.xlsx",

}

def get\_excel\_path(student\_id):

"""Get full path to Excel file with proper validation"""

for prefix, path in STEP\_FILES.items():

if student\_id.startswith(prefix):

if not os.path.exists(path):

raise FileNotFoundError(

f"Excel file not found: {path}\n"

f"Required for student ID: {student\_id}\n"

"Check:\n"

"1. File exists at specified path\n"

"2. Student ID prefixes match configuration"

)

return path

raise ValueError(

f"No Excel mapping for ID: {student\_id}\n"

"Valid prefixes:\n" +

"\n".join([f"{k} -> {v}" for k, v in STEP\_FILES.items()])

)

def get\_major(student\_id):

return os.path.basename(get\_excel\_path(student\_id)).split('.')[0]

# ========================

# VIDEO PROCESSING FUNCTIONS

# ========================

def reset\_daily\_state():

global last\_seen, detected\_students, cooldown\_tracker, last\_date\_check

now = datetime.now()

if now.date() != last\_date\_check.date():

print("\n--- NEW DAY RESET ---")

detected\_students.clear()

cooldown\_tracker.clear()

last\_seen.clear()

last\_date\_check = now

step\_files = [

'C:/Face-Detection-System/stud\_data/FY1.xlsx',

'C:/Face-Detection-System/stud\_data/FY2.xlsx',

'C:/Face-Detection-System/stud\_data/FY3.xlsx',

'C:/Face-Detection-System/stud\_data/ST1.xlsx',

'C:/Face-Detection-System/stud\_data/ST2.xlsx',

'C:/Face-Detection-System/stud\_data/ST3.xlsx',

'C:/Face-Detection-System/stud\_data/ST4.xlsx',

'C:/Face-Detection-System/stud\_data/ST5.xlsx',

'C:/Face-Detection-System/stud\_data/ST6.xlsx',

]

for path in step\_files:

mark\_absentees\_before\_attendance(path)

def process\_frame(frame, frame\_bg):

global current\_mode, mode\_start\_time, current\_student\_id

small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

rgb\_frame = cv2.cvtColor(small\_frame, cv2.COLOR\_BGR2RGB)

face\_locations = face\_recognition.face\_locations(rgb\_frame)

face\_encodings = face\_recognition.face\_encodings(rgb\_frame, face\_locations)

for encoding, face\_location in zip(face\_encodings, face\_locations):

matches = face\_recognition.compare\_faces(founded\_encodings, encoding, FACE\_MATCH\_THRESHOLD)

face\_distances = face\_recognition.face\_distance(founded\_encodings, encoding)

if not matches[np.argmin(face\_distances)]:

continue

student\_id = stud\_ID[np.argmin(face\_distances)]

last\_seen[student\_id] = time.time()

top, right, bottom, left = [v \* 4 for v in face\_location]

cv2.rectangle(frame\_bg, (55 + left, 162 + top), (55 + right, 162 + bottom), (0, 255, 0), 2)

cv2.putText(frame\_bg, f"ID: {student\_id}", (55 + left, 162 + top - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 255, 0), 2)

if student\_id not in detected\_students:

if is\_already\_present(student\_id):

if time.time() - cooldown\_tracker.get(student\_id, 0) > COOLDOWN\_SECONDS:

current\_mode = 3

mode\_start\_time = time.time()

cooldown\_tracker[student\_id] = time.time()

detected\_students.add(student\_id)

else:

if current\_student\_id is None or current\_student\_id != student\_id:

MarkAttendance(student\_id)

detected\_students.add(student\_id)

current\_mode = 1

mode\_start\_time = time.time()

current\_student\_id = student\_id

if student\_id not in student\_data:

name, attendance = LoadStudentName(student\_id)

img\_path = f'C:/Face-Detection-System/resized\_images/{student\_id}.jpg'

student\_data[student\_id] = {

'name': name,

'attendance': attendance,

'image': cv2.imread(img\_path) if os.path.exists(img\_path) else None

}

else:

if current\_student\_id == student\_id and current\_mode in [1, 2]:

continue

if time.time() - cooldown\_tracker.get(student\_id, 0) > COOLDOWN\_SECONDS:

current\_mode = 3

mode\_start\_time = time.time()

cooldown\_tracker[student\_id] = time.time()

# ========================

# MAIN SYSTEM LOOP

# ========================

def main():

global current\_mode, mode\_start\_time, current\_student\_id

check\_and\_update\_encodings()

load\_encodings()

load\_mode\_images()

for path in STEP\_FILES.values():

if not os.path.exists(path):

raise SystemExit(f"Critical error: Required file not found\nMissing: {path}")

for path in STEP\_FILES.values():

if mark\_off\_if\_needed(path):

print("System exiting due to holiday/weekend")

return

mark\_absentees\_before\_attendance(path)

preload\_detected\_students()

cap = cv2.VideoCapture(0)

cap.set(3, FRAME\_WIDTH)

cap.set(4, FRAME\_HEIGHT)

time.sleep(2)

while True:

success, img = cap.read()

if not success:

continue

reset\_daily\_state()

frame\_bg = imagebackground.copy()

frame\_bg[162:162+480, 55:55+640] = img

process\_frame(img, frame\_bg)

if current\_mode in [1, 2, 3] and mode\_start\_time:

elapsed = time.time() - mode\_start\_time

if elapsed > MODE\_DISPLAY\_DURATION:

if current\_mode == 1:

current\_mode = 2

mode\_start\_time = time.time()

elif current\_mode == 2:

current\_mode = 0

if current\_student\_id:

cooldown\_tracker[current\_student\_id] = time.time()

current\_student\_id = None

mode\_start\_time = None

elif current\_mode == 3:

current\_mode = 0

mode\_start\_time = None

frame\_bg[44:44+633, 808:808+414] = imgModeList[current\_mode]

if current\_mode == 1 and current\_student\_id:

student = student\_data.get(current\_student\_id)

if student:

if student['image'] is not None:

frame\_bg[175:175+216, 909:909+216] = student['image']

(w, h), \_ = cv2.getTextSize(student['name'], cv2.FONT\_HERSHEY\_COMPLEX, 1, 1)

x\_pos = 808 + (414 - w) // 2

cv2.putText(frame\_bg, str(student['attendance']), (861, 125), cv2.FONT\_HERSHEY\_COMPLEX, 1, (255,255,255), 1)

cv2.putText(frame\_bg, student['name'], (x\_pos, 445), cv2.FONT\_HERSHEY\_COMPLEX, 1, (50,50,50), 1)

cv2.putText(frame\_bg, f"ID: {current\_student\_id}", (1006, 493), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (255,255,255), 1)

cv2.putText(frame\_bg, get\_major(current\_student\_id), (1006, 550), cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (255,255,255), 1)

cv2.imshow("STEP School Talagang Campus", frame\_bg)

key = cv2.waitKey(1) & 0xFF

if key in (27, ord('q')):

break

if cv2.getWindowProperty("STEP School Talagang Campus", cv2.WND\_PROP\_VISIBLE) < 1:

break

cap.release()

cv2.destroyAllWindows()

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

main()