A PROJECT REPORT ON

REALTIME FACE ATTENDENCE SYSTEM USING PYTHON AND MONGODB

LITTLE FLOWER DEGREE , UPPAL

DEPARTMENT OF PHYSICAL SCIENCE

2024-2025

SUBMITTED BY

D.AKHIL(201022539002)

P.RAHUL(201022539009)

B.LOKESH(201022539001)

UNDER GUIDANCE OF

MR. P. VIJAY BHASKAR DEPARTMENT OF COMPUTER SCIENCE

SUPPORTED BY MR. VIJAY REDDY HEAD, DEPARTMENT OF PHYSICAL SCIENCE

DEPARTMENT OF PHYSICAL SCIENCE

LITTLE FLOWER DEGREE COLLEGE

UPPAL , HYDERABAD

CERTIFICATE

This is to certify that the project **REALTIME FACE ATTENDENCE SYSTEM USING PYTHON AND** MONGODB (vs code) being submitted by

D.AKHIL(201022539002)

P.RAHUL(201022539009)

B.LOKESH(201022539001)

in partial fulfillment of the requirement for the award of degree of Bachelor of Science is a bonafide record of the award of the project work done by department of Science, Little Flower Degree college , Uppal, Hyderabad.

HOD **INTERNAL** **EXTERNAL**

DECLARATION

We undersign, hereby declare that the project **" REALTIME FACE ATTENDENCE SYSTEM USING PYTHON AND MONGODB"** submitted in partial fulfillment of the requirement for the award of degree of Bachelor of Science of Osmania University is a bonafide record of the award of the project work done by us under the guidance of

MR. P. VIJAY BHASKAR, DEPARTMENT OF COMPUTER SCIENCE, Department of physical Science, Little Flower Degree & PG College, Uppal, Hyderabad.

This Project has not previously formalised the award of any degreeor similar title by any university .

DATE:

D.AKHIL(201022539002)

P.RAHUL(201022539009)

B.LOKESH(201022539001)

Acknowledgment

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this face attendance system.

First and foremost, I am deeply thankful to my mentors and instructors, whose guidance and insights have been invaluable in shaping my understanding of computer vision, database management, and software development. Their continuous support has greatly enhanced my technical knowledge and problem-solving skills.

I extend my appreciation to the open-source community and developers of libraries such as OpenCV, face recognition, NumPy, and MongoDB. Their contributions have been instrumental in building this project, enabling seamless facial recognition and efficient data management.

Special thanks to my peers and colleagues, whose constructive feedback and discussions have helped me refine the system, troubleshoot challenges, and enhance its performance. Their encouragement and collaboration made the development process smoother and more engaging.

I am also grateful to my educational institution, Omega Degree College for Women, for providing a conducive learning environment and access to resources that supported the research and development of this project.

Finally, I acknowledge the unwavering support of my family and friends, whose motivation and belief in my abilities have been a constant source of inspiration throughout this journey.

This project would not have been possible without the collective support and knowledge shared by these individuals and communities. I look forward to further improvements and innovations in this field.

Abstract

The **Face Attendance System** is an advanced, AI-powered solution designed to automate attendance tracking using facial recognition technology. This project leverages **OpenCV**, **face recognition**, **MongoDB**, and **Python** to efficiently detect, recognize, and log attendance in real-time.

The system captures live video input, processes facial features using deep learning-based encoding techniques, and compares them with pre-stored encodings to identify individuals. Once a match is found, the attendance is recorded in a MongoDB database, ensuring secure and efficient data management. The project implements a **multi-mode UI**, including **face detection, student verification, attendance logging, and warning notifications**, providing an interactive and user-friendly experience.

Key features of this system include:

✓ **Real-time face recognition** with high accuracy

✓ **Automatic attendance logging** in a cloud-based database

✓ **Multi-mode display system** for smooth user interaction

✓ **Prevention of duplicate attendance** within a set time frame

✓ **CSV export functionality** for attendance reports

✓ **Audio alerts** for successful attendance marking

This system eliminates the need for manual attendance-taking, reduces errors, and enhances security in educational institutions and workplaces. Future enhancements include integrating a **web dashboard** for real-time attendance monitoring and expanding the system's scalability for larger environments.

Introduction

Traditional attendance systems, such as manual roll-calling and RFID-based methods, are time-consuming, prone to errors, and vulnerable to proxy attendance. To address these challenges, **Face Recognition-based Attendance Systems** have emerged as an efficient and automated solution.

This project implements a **Face Attendance System** using **Python, OpenCV, face_recognition, and MongoDB** to provide a seamless and secure method for tracking attendance. The system uses a live camera feed to capture images, detect faces, and compare them with stored face encodings in a database. If a match is found, the system automatically logs the attendance, preventing fraudulent practices and ensuring an efficient attendance-taking process.

The system operates in **four modes**:

1. **Detection Mode (Mode 0)** – Captures a live feed and detects faces.
2. **Verification Mode (Mode 1)** – Identifies the individual and fetches their details from the database.
3. **Logging Mode (Mode 2)** – Updates the attendance record if the face is recognized and attendance criteria are met.
4. **Warning Mode (Mode 3)** – Displays a warning if attendance is attempted within a restricted timeframe.

The project provides several **key features**, including **real-time face recognition**, **automated attendance marking**, **secure database storage**, **CSV export functionality**, and **audio alerts** for successful attendance. It is designed for educational institutions and workplaces, where accurate and efficient attendance tracking is essential.

By integrating **computer vision and machine learning**, this system minimizes human intervention, enhances accuracy, and provides a scalable solution for attendance management. Future developments include a **web-based dashboard** for real-time monitoring and analytics.

Benefits of Using the Face Recognition-Based Attendance System

Implementing a **Face Recognition-Based Attendance System** offers numerous advantages over traditional attendance methods. Below are some key benefits:

1. Automation and Efficiency

- Eliminates the need for manual roll calls, reducing time and effort.
- Speeds up the attendance process, allowing seamless entry for students/employees.
- Reduces paperwork and administrative workload.

2. Accuracy and Reliability

- Uses **AI-based face recognition** to ensure precise identification.
- Prevents errors caused by manual data entry or misidentification.
- Avoids proxy attendance (buddy punching), ensuring genuine attendance records.

3. Security and Fraud Prevention

- Ensures **tamper-proof** attendance records by eliminating human manipulation.
- Face recognition technology prevents unauthorized users from marking attendance.
- Data is securely stored in **MongoDB**, reducing risks of data loss or unauthorized access.

4. Real-time Monitoring and Insights

- Attendance logs can be stored and analyzed for **performance tracking**.
- Integration with a **web-based dashboard** allows **administrators to view attendance history** remotely.
- Attendance patterns can be analyzed to improve class/workplace engagement.

5. Contactless and Hygienic

- Reduces the need for fingerprint scanners, ID cards, or manual sign-ins.
- Ideal for **post-pandemic scenarios**, where contactless systems are preferred for safety.

6. Easy Integration and Scalability

- Can be **integrated into existing school, college, or office management systems**.
- Scalable to support **large organizations and multiple locations**.
- Can be enhanced with additional features like **voice notifications, multi-camera setups, and cloud storage**.

7. Cost-Effective in the Long Run

- Reduces costs associated with maintaining **ID cards, RFID scanners, and manual record-keeping**.
- Requires minimal human supervision, cutting down administrative costs.
- Improves productivity by reducing the time spent on attendance management.

By leveraging **computer vision and machine learning**, this **Face Attendance System** provides a **modern, efficient, and secure** solution for tracking attendance, ensuring a **smooth and fraud-free** experience for both students and employee

About Face Recognition-Based Attendance System

The **Face Recognition-Based Attendance System** is an advanced biometric solution that automates attendance tracking using artificial intelligence (AI) and computer vision. Instead of traditional methods like manual roll calls, RFID cards, or fingerprint scanning, this system leverages **facial recognition technology** to identify and record attendance seamlessly.

How It Works

1. **Face Detection:** The system captures an image from a live camera feed.
2. **Face Encoding & Matching:** It compares the detected face with a pre-stored database of known faces.
3. **Attendance Logging:** If a match is found, the system logs the individual's attendance in a database (MongoDB).
4. **Automated Updates:** Attendance records are stored with timestamps and can be exported for further analysis.

Core Technologies Used

- **OpenCV** – For image processing and real-time video capture.
- **face_recognition Library** – For encoding and recognizing faces.
- **MongoDB** – For storing and managing attendance data.
- **Python & NumPy** – For efficient computation and data handling.

Key Features

✅ **Contactless & Secure** – No need for physical cards or fingerprint scanners.
✅ **Fast & Automated** – Attendance is marked within seconds.
✅ **Fraud Prevention** – Eliminates buddy punching or proxy attendance.

✅ **Scalability** – Can be used in schools, offices, or large organizations.

✅ **Real-time Monitoring** – Provides instant attendance reports.

This system is ideal for educational institutions, corporate offices, and any organization looking to enhance efficiency, security, and convenience in attendance tracking.

Role of Python in the Face Recognition-Based Attendance System

Python plays a **crucial role** in the development and execution of the Face Recognition-Based Attendance System. It serves as the **backbone** of the project, integrating various libraries and frameworks to handle image processing, machine learning, and database management efficiently.

---

1 . Image Processing & Face Recognition

Python provides powerful libraries to **capture, process, and recognize faces** in real-time.

- **OpenCV**: Used for capturing video from the webcam and processing images.
- **face_recognition**: Helps detect, encode, and compare faces for authentication.
- **NumPy**: Handles numerical operations, such as calculating face distances for matching.

💡 *Python enables seamless face detection and recognition with minimal computation time.*

---

2 . Data Handling & Storage

The attendance system requires a **database** to store student/employee details and attendance records.

- **MongoDB (via PyMongo)**: A NoSQL database that stores attendance logs, timestamps, and user details.

- **Pickle**: Used to serialize and save pre-encoded face data, reducing processing time.

💡 *Python's database libraries allow real-time attendance logging and data retrieval.*

---

3. Real-time Decision Making & Mode Switching

The system switches between different operational **modes** based on face recognition results:

- **Mode 0** → Ready to scan
- **Mode 1** → Attendance recorded
- **Mode 2** → Attendance successful message
- **Mode 3** → Too soon warning

Python's **control structures** (if-else conditions, loops, and threading) help manage these **dynamic transitions** smoothly.

💡 *Python makes real-time decision-making fast and reliable!*

---

4 . Automation & Alerts

Python is used to **automate actions and provide feedback**:

- **Playsound**: Generates sound alerts when attendance is marked.
- **Time & Date Handling**: Ensures attendance is recorded with accurate timestamps.

💡 *Python helps automate notifications, making the system interactive and user-friendly.*

Why Python?

✅ **Easy to Learn & Use** – Simple syntax and vast libraries.

✅ **Powerful & Scalable** – Can handle small classrooms to large corporations.

✅ **Extensive Library Support** – Face recognition, image processing, and database handling are all supported.

✅ **Cross-Platform Compatibility** – Works on Windows, Linux, and macOS.

Python is the perfect choice for developing **efficient, scalable, and real-time** biometric attendance systems! 🚀

Libraries Used in the Face Recognition-Based Attendance System :

This project relies on several Python libraries to perform **face recognition, image processing, database handling, and automation**. Below is a detailed explanation of each library used in the system.

---

1 . OpenCV (cv2)

Purpose:

- Captures video from the webcam.
- Reads and processes images.
- Performs image manipulation (resizing, overlaying UI elements, etc.).

Usage in the Project:

- cap = cv2.VideoCapture(0): Captures frames from the webcam.
- cv2.imread(): Reads background and mode images.
- cv2.imshow(): Displays the attendance system interface.

💡 *OpenCV is essential for handling real-time video and image processing.*

---

2 . face_recognition

Purpose:

- Detects faces in images and video.
- Encodes face data for efficient comparison.
- Matches live faces with stored encodings.

Usage in the Project:

- face_recognition.face_locations(imgS): Identifies face positions.
- face_recognition.face_encodings(imgS, faceCurFrame): Encodes detected faces.
- face_recognition.compare_faces(encodeListKnown, encodeFace): Compares detected faces with stored data.

💡 *This library enables accurate and fast face recognition using deep learning models.*

---

3 . NumPy (numpy)

Purpose:

- Performs mathematical operations on arrays.
- Calculates distances between face encodings.

Usage in the Project:

- np.argmin(faceDis): Finds the closest match among stored face encodings.
- faceDis = face_recognition.face_distance(encodeListKnown, encodeFace): Computes similarity scores.

💡 *NumPy speeds up numerical calculations, making face recognition efficient.*

---

4 . PyMongo (pymongo)

Purpose:

- Connects Python to MongoDB.
- Stores and retrieves student attendance records.

Usage in the Project:

- client = MongoClient("<MongoDB Connection String>"): Connects to the database.
- collection.find_one({"_id": Id}): Fetches student details.
- collection.update_one({"_id": Id}, {"$set": {"total_attendance": student_data['total_attendance']}}): Updates attendance records.

💡 *PyMongo allows seamless database integration for real-time attendance tracking.*

---

5 . Pickle (pickle)

Purpose:

- Saves and loads encoded face data for faster processing.

Usage in the Project:

- with open("EncodeFile.p", "rb") as file: Loads stored face encodings.
- pickle.dump((encodeListKnown, studentIds), file): Saves encoded face data.

💡 *Pickle helps avoid re-encoding faces every time the system starts, reducing processing time.*

---

6 . Datetime (datetime)

Purpose:

- Retrieves and formats timestamps for attendance records.

Usage in the Project:

- current_time = datetime.now(timezone.utc): Gets the current time.
- last_att_time = datetime.fromisoformat(last_att_time_str): Converts stored timestamps for comparison.

💡 *Ensures accurate date and time tracking for attendance logs.*

---

7 . OS (os)

Purpose:

- Interacts with the file system (access images, encode files, etc.).

Usage in the Project:

- os.listdir(folderModePath): Lists mode images in a directory.
- os.path.join(folderModePath, path): Constructs paths for images.

💡 *Used for handling file and directory operations efficiently.*

---

8 . Playsound (playsound)

Purpose:

- Plays sound alerts when attendance is recorded.

Usage in the Project:

- playsound.playsound("attendance_success.mp3"): Plays a notification sound.

💡 *Enhances user experience by providing audible feedback for attendance confirmation.*

Camera :

```
import cv2

# Open the webcam

video = cv2.VideoCapture(0)

while True:

    ret, frame = video.read()  # Read frame from the camera

    if not ret:

        print("❌ Failed to capture frame")

        break

    cv2.imshow("Webcam Feed", frame)  # Show the frame

    # Press 'q' to exit

    if cv2.waitKey(1) & 0xFF == ord("q"):

        break

# Release the camera and close the window

video.release()

cv2.destroyAllWindows()
```

1 . Opening the Webcam

```
python

CopyEdit

video = cv2.VideoCapture(0)
```

- Opens the **default camera** (device index 0).
- If using an **external webcam**, try cv2.VideoCapture(1) or cv2.VideoCapture(2).

---

2 . Capturing Frames in a Loop

python

CopyEdit

while True:

ret, frame = video.read() # Read frame from the camera

- **video.read()** captures a frame from the webcam.
- **ret (Boolean)**: True if a frame is successfully captured, False otherwise.
- **frame (numpy array)**: The actual image data.

---

3 . Checking if Frame Capture is Successful

python

CopyEdit

if not ret:

print("❌ Failed to capture frame")

break

- If ret is False, it means the camera couldn't capture a frame. The script **prints an error message** and exits the loop.

---

4 . Displaying the Video Feed

python

CopyEdit

```
cv2.imshow("Webcam Feed", frame)  # Show the frame
```

- **Displays the live webcam feed** in a window named "Webcam Feed".

---

5 . Exit Condition (Press 'q' to Quit)

python

CopyEdit

```
if cv2.waitKey(1) & 0xFF == ord("q"):
    break
```

- **cv2.waitKey(1)**: Waits for **1 millisecond** for a key press.
- **ord("q")**: If the 'q' key is pressed, the loop exits.

---

6 Releasing the Camera & Closing Windows

python

CopyEdit

```
video.release()
cv2.destroyAllWindows()
```

- **Releases the webcam**, freeing it for other applications.
- **Closes all OpenCV windows** to prevent crashes.

🚀 **How This Works**

✅ Opens the webcam.
✅ Continuously captures frames & displays them in real-time.
✅ Stops when **'q'** is pressed.
✅ Closes the webcam and window safely.

Background image :

while True:

success, img = cap.read()

imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)

imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

imgBackground[162:162 + 480, 55:55 + 640] = img

1 . Capturing Frames Continuously

python

CopyEdit

while True:

success, img = cap.read()

- **cap.read()** captures a frame from the webcam.
- **success (Boolean)**: True if the frame is successfully captured, False otherwise.
- **img (numpy array)**: The captured frame (image).

---

2 . Resizing the Captured Frame

python

CopyEdit

```
imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
```

- The image (img) is **downscaled to 25% of its original size**.
- This is done to **increase processing speed** for face recognition.

---

3 .Converting to RGB Format

python

CopyEdit

```
imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
```

- OpenCV captures images in **BGR format**, but the face_recognition library requires **RGB format**.
- This line **converts the color format** from **BGR to RGB**.

---

4 . Overlaying the Image on a Background

python

CopyEdit

```
imgBackground[162:162 + 480, 55:55 + 640] = img
```

- **imgBackground** is a pre-loaded **background image** (like a UI template).
- The captured frame (img) is placed inside imgBackground at coordinates **(162, 55)**.
- The frame is pasted with a size of **480 × 640 pixels**.

🚀 **What This Code Does**

✅ **Reads a frame** from the webcam.
✅ **Resizes** it for fast processing.
✅ **Converts** it to RGB for face recognition.
✅ **Overlays** it onto a background image.

Encodings :

Face recognition systems process an image and generate a **128-dimensional encoding** (a vector of 128 floating-point numbers). This vector contains **key facial features** extracted from the image.

Step-by-Step Encoding Process:

1 . **Face Detection** → The system identifies faces in an image.
2 . **Feature Extraction** → It extracts facial landmarks (eyes, nose, mouth, jawline, etc.).
3 . **Encoding Generation** → The system converts facial features into a **128-dimensional numerical vector**.
4 . **Comparison & Matching** → When a new face appears, its encoding is compared to known face encodings to determine a match.

MongoDB Connection

client = MongoClient("mongodb+srv://dasariakhil:201022539002@cluster0.lonxb.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")

```
db = client['LFDCAttendence']
fs = gridfs.GridFS(db)

# Folder containing student images
folder_path = "C:\\Users\\Akhil Dasari\\facerecognition\\images"
path_list = os.listdir(folder_path)

image_list.append(image)
    # Extract student ID from filename
(without extension)
    student_id = os.path.splitext(path)[0]
    student_ids.append(student_id)
    # Upload image to GridFS
    with open(image_path, "rb") as img_file:
        file_id = fs.put(img_file, filename=path)
        print(f" 📤 Image '{path}' uploaded successfully to GridFS with ID: {file_id}")

print(" ✅ All student IDs collected:", student_ids)
```

1. **Appending Image to a List**

python

CopyEdit

image_list.append(image)

- You are storing image (probably an OpenCV or PIL image) in image_list.

2. **Extracting Student ID from Filename**

python

CopyEdit

student_id = os.path.splitext(path)[0]

student_ids.append(student_id)

- os.path.splitext(path)[0] removes the file extension (e.g., .jpg, .png) and returns only the filename.
- However, path should be the actual filename, not the full path. If path contains a directory ("folder/image.jpg"), os.path.splitext(path)[0] will return "folder/image", which is incorrect.

3. **Uploading Image to GridFS**

python

CopyEdit

with open(image_path, "rb") as img_file:

file_id = fs.put(img_file, filename=path)

- The image is opened in **binary mode ("rb")** and uploaded to GridFS.

4. **Printing Confirmation**

python

CopyEdit

print(f"📤 Image '{path}' uploaded successfully to GridFS with ID: {file_id}")

- Shows a success message after uploading.

```
def find_encodings(images, student_ids):
    encode_list = []
    valid_ids = []

    for img, sid in zip(images, student_ids):
            img_rgb = cv2.cvtColor(img,
  cv2.COLOR_BGR2RGB)
            encodes =
  face_recognition.face_encodings(img_rgb)

            if encodes:
                encode_list.append(encodes[0])
                valid_ids.append(sd)
            else:
                print(f"⚠ No face found in image of
  student ID: {sid}, skipping...")

        return encode_list, valid_ids

```

1. **Initialize Lists**

encode_list = [] # Stores valid face encodings

valid_ids = [] # Stores corresponding student IDs

2. **Loop Through Images and IDs**

python

CopyEdit

for img, sid in zip(images, student_ids):

- Iterates over **both images and their corresponding student IDs** using zip().
- Ensures the same **index** in images and student_ids corresponds to the same student.

3. **Convert Image to RGB**

python

CopyEdit

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

- OpenCV loads images in **BGR** format, but face_recognition expects **RGB**.

4. **Extract Face Encodings**

python

CopyEdit

```
encodes = face_recognition.face_encodings(img_rgb)
```

- The face_encodings() function returns a **list** of face encodings.
- If the list is **not empty**, it means a face was found.

5. **Store Encoding & ID if a Face is Found**

python

CopyEdit

```
if encodes:
  encode_list.append(encodes[0])
  valid_ids.append(sid)
```

- Takes only the **first detected face encoding** (encodes[0]).
- Adds the corresponding **student ID** to valid_ids.

6. **Handle Missing Faces**

python

CopyEdit

else:

print(f" ⚠️ No face found in image of student ID: {sid}, skipping...")

- If no face is detected, it prints a warning and **skips the student**.

7. **Return Encodings & Valid IDs**

python

CopyEdit

return encode_list, valid_ids

- Returns **only valid encodings** and **corresponding student IDs**.

Save encodings and corresponding student IDs

data = [encode_list_known, valid_ids]

with open("EncodeFile.p", "wb") as file:

pickle.dump(data, file)

print(" 💾 EncodeFile.p saved successfully.")

□ Prepare Data for Saving

python

CopyEdit

data = [encode_list_known, valid_ids]

- encode_list_known: List of **face encodings** (NumPy arrays).
- valid_ids: List of **student IDs** corresponding to each encoding.

□ Save Data to a File (EncodeFile.p)

python

CopyEdit

with open("EncodeFile.p", "wb") as file:

pickle.dump(data, file)

- "wb" mode means **write binary** since encodings are NumPy arrays.
- pickle.dump(data, file) writes the data to **EncodeFile.p**.

Confirmation Message

python

CopyEdit

print("💾 EncodeFile.p saved successfully.")

- Confirms that the data has been store

Encoding.py :

import cv2

import pickle

import os

import face_recognition

from pymongo import MongoClient

import gridfs

MongoDB Connection

client = MongoClient("mongodb+srv://dasariakhil:201022539002@cluster0.lonxb.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")

```
db = client['LFDCAttendence']
fs = gridfs.GridFS(db)
# Folder containing student images
folder_path = "C:\\Users\\Akhil Dasari\\facerecognition\\images"
path_list = os.listdir(folder_path)
image_list = []
student_ids = []
for path in path_list:
    image_path = os.path.join(folder_path, path)
    image = cv2.imread(image_path)
    if image is None:
        print(f"⚠️ Could not read image: {image_path}")
        continue
    # Add image to list
    image_list.append(image)
    # Extract student ID from filename (without extension)
    student_id = os.path.splitext(path)[0]
    student_ids.append(student_id)
    # Upload image to GridFS
    with open(image_path, "rb") as img_file:
        file_id = fs.put(img_file, filename=path)
```

```
        print(f"📤 Image '{path}' uploaded successfully to GridFS with ID: {file_id}")
print("✅ All student IDs collected:", student_ids)
# Encode faces
def find_encodings(images, student_ids):
    encode_list = []
    valid_ids = []
    for img, sid in zip(images, student_ids):
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encodes = face_recognition.face_encodings(img_rgb)
        if encodes:
            encode_list.append(encodes[0])
            valid_ids.append(sid)
        else:
            print(f"⚠️ No face found in image of student ID: {sid}, skipping...")

    return encode_list, valid_ids

print("🔍 Encoding started...")
encode_list_known, valid_ids = find_encodings(image_list, student_ids)
```

```
print("✅ Encoding completed.")

# Save encodings and corresponding student IDs

data = [encode_list_known, valid_ids]

with open("EncodeFile.p", "wb") as file:

    pickle.dump(data, file)

print("💾 EncodeFile.p saved successfully.")
```

Encoding summary :

1. Connects to MongoDB & GridFS

Establishes a connection to a MongoDB database (LFDCAttendence) using GridFS for storing student images.

- Ensures a reliable connection for **handling large image files**.

2 .Loads & Processes Student Images

- Reads all image files from a specified folder.
- Extracts **student IDs** from filenames.
- Checks if an image **already exists in GridFS** before uploading, avoiding duplicate uploads.
- Uploads **new images** to MongoDB using **GridFS**.

3 . Face Encoding using face_recognition

- Converts images to **RGB format** (required for face detection).
- Uses face_recognition.face_encodings() to extract **facial features**.
- Handles cases where:
    - **No face is detected** → Skips the image.

- **Multiple faces are detected** → Warns the user and selects the first face.

4 . Encodings & Student IDs

- Stores the **face encodings** and **corresponding student IDs** in a list.
- Saves this data to a **Pickle (.p) file** for quick reloading.
- Includes **error handling** to prevent failures when saving the file.

Connecting python with Mongodb

```
from pymongo import MongoClient

client =
MongoClient("mongodb+srv://dasariakhil:201022539002@cluster0.lonxb.mongo
db.net/?retryWrites=true&w=majority&appName=Cluster0")

db = client["LFDCAttendence"]

collection = db["Datascience"]

# Load and resize mode images

folderModePath = "resources/modes"

modePathList = os.listdir(folderModePath)

imageModeList = []

for path in modePathList:

    img = cv2.imread(os.path.join(folderModePath, path))

    if img is not None:

        img_resized = cv2.resize(img, (414, 633))  # Resize to match the expected
shape

        imageModeList.append(img_resized)



folderModePath = "resources/modes"

modePathList = os.listdir(folderModePath)
```

Defines the folder path where the mode images are stored.

Gets a list of all file names in the resources/modes directory.

imageModeList = []

Initializes an empty list to store processed mode images.

for path in modePathList:

Loops through each image file found in modePathList.

    img = cv2.imread(os.path.join(folderModePath, path))

Reads each image using OpenCV (cv2.imread()).

    if img is not None:

Ensures the image is successfully loaded before processing it.

        img_resized = cv2.resize(img, (414, 633)) # Resize to match the expected shape

Resizes the image to 414x633 pixels for consistency.

        imageModeList.append(img_resized)

Adds the processed image to the imageModeList.

✅ Purpose & Functionality

Loads mode images stored in "resources/modes".

Resizes each image to 414x633 pixels.

Stores them in imageModeList for later use in your face attendance system.

Load encodings

print("🔍 Loading encode file...")

with open("EncodeFile.p", "rb") as file:

encodeListKnown, studentIds = pickle.load(file)

print("✅ Encode file loaded")

with open("EncodeFile.p", "rb") as file:

encodeListKnown, studentIds = pickle.load(file)

Opens the file "EncodeFile.p" in read-binary (rb) mode.

Unpickles (loads) the previously stored face encodings and student IDs.

encodeListKnown → A list of 128-dimensional face encodings (numeric vectors).

studentIds → A list of student IDs corresponding to these encodings.

Edit

print("✅ Encode file loaded")

Prints a success message after loading is complete.

✅ **Purpose & Functionality**

- **Saves time** by avoiding re-encoding faces every time the system runs.
- **Loads face encodings and student IDs** for quick recognition.
- Uses **pickle**, which is efficient for storing and retrieving Python objects.

Running infinite loop :

```
if modeType == 0:
    imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[2]  # Ready Mode
    for encodeFace, faceLoc in zip(encodeCurFrame, faceCurFrame):
        matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
        faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
        matchIndex = np.argmin(faceDis)
        if matches[matchIndex]:
            Id = str(studentIds[matchIndex])
            print("✅ Face Matched. ID:", Id)
            student_data = collection.find_one({"_id": Id})
            current_time = datetime.now(timezone.utc)  # Ensure UTC format
            current_attendance_time = current_time.isoformat()
            if student_data:
                last_att_time_str = student_data.get('last_attendance_time', '').strip()
                # Convert stored time to datetime object with timezone awareness
                if last_att_time_str:
                    try:
                        last_att_time = datetime.fromisoformat(last_att_time_str)
```

```
            if last_att_time.tzinfo is None:
                last_att_time = last_att_time.replace(tzinfo=timezone.utc)
            secondsDiff = (current_time - last_att_time).total_seconds()
        except ValueError:
            print(f"⚠️ Invalid time format in DB: {last_att_time_str}")
            secondsDiff = None
    else:
        secondsDiff = None

    # Check time difference for re-attendance
    if secondsDiff is None or secondsDiff > 120:
        student_data['total_attendance'] += 1
        # Update MongoDB
        collection.update_one({"_id": Id}, {
            "$set": {
                "total_attendance": student_data['total_attendance'],
                "last_attendance_time": current_attendance_time
            },
            "$push": {
                "attendance_time_log": current_attendance_time
            }
```

```
                })
                print("✅ Attendance updated at:", current_attendance_time)
                modeType = 1  # Show student info
            else:
                print("⏳ Too soon. Last attendance:",
student_data['last_attendance_time'])
                modeType = 3  # Too Soon Warning
                imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[1]
        else:
            print("❌ Student data not found.")
        counter = 0
        break
```

explaination :

- When the system is in **mode 0** (Ready Mode), it displays a specific mode image (imageModeList[2]).

---

1 . Matching Detected Faces with Known Encodings

python

CopyEdit

```
for encodeFace, faceLoc in zip(encodeCurFrame, faceCurFrame):
    matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
```

```
faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
matchIndex = np.argmin(faceDis)
```

- **encodeCurFrame** → List of encodings detected in the current frame.
- **faceCurFrame** → List of face locations in the current frame.
- **compare_faces()** → Compares detected encodings with stored encodings.
- **face_distance()** → Computes the similarity distance (lower is better).
- **argmin()** → Finds the index of the best match.

python

CopyEdit

```
if matches[matchIndex]:
    Id = str(studentIds[matchIndex])
    print(" ✅ Face Matched. ID:", Id)
```

- If a face **matches**, the corresponding student **ID is retrieved**.

---

2 . Fetching Student Data from MongoDB

python

CopyEdit

```
    student_data = collection.find_one({"_id": Id})
    current_time = datetime.now(timezone.utc)  # Ensure UTC format
    current_attendance_time = current_time.isoformat()
```

- **Finds the student record in MongoDB** using their ID.

- **Gets the current timestamp** in UTC format for logging attendance.

---

3 . Checking Last Attendance Time

python

CopyEdit

```
if student_data:
    last_att_time_str = student_data.get('last_attendance_time', '').strip()
```

- **Retrieves the last recorded attendance time** from the database.

python

CopyEdit

```
if last_att_time_str:
    try:
        last_att_time = datetime.fromisoformat(last_att_time_str)
        if last_att_time.tzinfo is None:
            last_att_time = last_att_time.replace(tzinfo=timezone.utc)
        secondsDiff = (current_time - last_att_time).total_seconds()
    except ValueError:
        print(f"⚠️ Invalid time format in DB: {last_att_time_str}")
        secondsDiff = None
else:
    secondsDiff = None
```

- Converts the stored timestamp into a **Python datetime object**.
- **Handles invalid time formats gracefully**.
- **Calculates the time difference** between the last and current attendance records.

---

4 . Updating Attendance in MongoDB

python

CopyEdit

```
    if secondsDiff is None or secondsDiff > 120:
        student_data['total_attendance'] += 1


        # Update MongoDB
        collection.update_one({"_id": Id}, {
            "$set": {
                "total_attendance": student_data['total_attendance'],
                "last_attendance_time": current_attendance_time
            },
            "$push": {
                "attendance_time_log": current_attendance_time
            }
        })
```

print(" ✅ Attendance updated at:", current_attendance_time)

modeType = 1 # Show student info

- **If 120 seconds have passed** since the last attendance:
    - Increments total_attendance.
    - **Updates MongoDB**:
        - Sets the new total_attendance and last_attendance_time.
        - **Logs the attendance time** inside an array (attendance_time_log).
    - **Switches to modeType = 1** (Display student info).

---

5 . Handling Duplicate Attendance Attempts

python

CopyEdit

else:

print(" ⏳ Too soon. Last attendance:", student_data['last_attendance_time'])

modeType = 3 # Too Soon Warning

imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[1]

- If the last attendance was **less than 120 seconds ago**, it **prevents duplicate attendance**.
- **Displays a warning UI** (modeType = 3).

---

6 . Handling Missing Student Records

python

CopyEdit

```
else:
    print("❌ Student data not found.")
```

- If the student ID is **not found in MongoDB**, it prints an error.

✅ **Summary of Functionality**

1 . **Detects and encodes a face from the camera**.
2 . **Compares the encoding with stored student data**.
3 . **If matched** → Fetches student details from MongoDB.
4 . **Checks the last attendance time** to avoid duplicate entries.
5 . **Updates attendance in MongoDB** (if the time condition is met).
6 . **Prevents duplicate entries within 120 seconds**.
7 . **Displays different UI modes** based on the attendance status.

Redirecting to modes and Adding texts

```
# --- Mode 1: Show Student Info ---

   elif modeType == 1:

      imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[1]  # Show Info Mode



      if student_data:

         # Display student info

         cv2.putText(imgBackground, str(student_data['total_attendance']), (861, 125),

                     cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 1)

         cv2.putText(imgBackground, str(student_data['group']), (1006, 550),

                     cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)

         cv2.putText(imgBackground, str(student_data['_id']), (1006, 493),

                     cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)

         cv2.putText(imgBackground, str(student_data['starting_year']), (1125, 625),

                     cv2.FONT_HERSHEY_COMPLEX, 0.6, (100, 100, 100), 1)

         # Display student name

         (w, h), _ = cv2.getTextSize(student_data['name'], cv2.FONT_HERSHEY_COMPLEX, 1, 1)

         offset = (414 - w) // 2
```

```
            cv2.putText(imgBackground, str(student_data['name']), (808 + offset, 445),
                        cv2.FONT_HERSHEY_COMPLEX, 1, (50, 50, 50), 1)
            # ✅ Show Attendance Time
            cv2.putText(imgBackground, current_attendance_time, (860, 200),
                        cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 255, 255), 1)
            cv2.waitKey(10)
        counter += 1
        if counter >= 20:
            modeType = 2
            counter = 0
    # --- Mode 2: End Message ---
    elif modeType == 2:
        imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[1]  # End Message Mode
        counter += 1
        if counter >= 20:
            modeType = 0
            counter = 0
            Id = -1
            student_data = None
    # --- Mode 3: Too Soon Warning ---
```

```
    elif modeType == 3:
        imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[3]  # Too Soon Mode
        counter += 1
        if counter >= 20:
            modeType = 0
            counter = 0
            Id = -1
            student_data = None
    cv2.imshow("Face Attendance", imgBackground)
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

1️⃣ Mode 1: Display Student Info

```
python
CopyEdit
elif modeType == 1:
    imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[1]  # Show Info Mode
```

- **Switches to Mode 1 (Student Info Mode)**.

- Loads **imageModeList[1]**, which is the UI for displaying student details.

💻 Displaying Student Information on the Screen

python

CopyEdit

```
if student_data:
    # Display total attendance
    cv2.putText(imgBackground, str(student_data['total_attendance']), (861, 125),
                cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 1)

    # Display student group
    cv2.putText(imgBackground, str(student_data['group']), (1006, 550),
                cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)

    # Display student ID
    cv2.putText(imgBackground, str(student_data['_id']), (1006, 493),
                cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)

    # Display starting year
    cv2.putText(imgBackground, str(student_data['starting_year']), (1125, 625),
                cv2.FONT_HERSHEY_COMPLEX, 0.6, (100, 100, 100), 1)
```

- Displays **total attendance count** at (861, 125).
- Displays the **group name** at (1006, 550).
- Displays **student ID** at (1006, 493).
- Displays the **starting year** at (1125, 625).

🖥️ Centering and Displaying Student Name

python

CopyEdit

```
(w, h), _ = cv2.getTextSize(student_data['name'], cv2.FONT_HERSHEY_COMPLEX, 1, 1)
offset = (414 - w) // 2
cv2.putText(imgBackground, str(student_data['name']), (808 + offset, 445),
            cv2.FONT_HERSHEY_COMPLEX, 1, (50, 50, 50), 1)
```

- Uses **cv2.getTextSize()** to get the width of the student's name.
- Centers the name horizontally within the **414-pixel wide** display area.

✅ Displaying Attendance Timestamp

python

CopyEdit

```
cv2.putText(imgBackground, current_attendance_time, (860, 200),
            cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 255, 255), 1)
```

- **Displays the timestamp** of when the attendance was recorded.

⏳ **Holding Mode 1 for 20 Frames**

python

CopyEdit

```
    cv2.waitKey(10)
    counter += 1
    if counter >= 20:
        modeType = 2
        counter = 0
```

- Holds **Mode 1** for **20 frames** before switching to **Mode 2**.
- **Resets the counter** once it reaches 20.

---

2 . Mode 2: Display "Attendance Recorded" Message

python

CopyEdit

```
elif modeType == 2:
    imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[1]  # End Message Mode
```

- **Switches to Mode 2** after showing the student info.
- Uses imageModeList[1], likely showing a **confirmation message** ("Attendance Marked Successfully").

⏳ Holding Mode 2 for 20 Frames

python

CopyEdit

```
counter += 1
if counter >= 20:
    modeType = 0  # Reset to Ready Mode
    counter = 0
    Id = -1
    student_data = None
```

- Holds Mode 2 for **20 frames**, then **resets everything**:
    - Sets modeType = 0 (Ready Mode).
    - Clears the Id and student_data.

---

3 . Mode 3: Too Soon Warning

python

CopyEdit

```
elif modeType == 3:
    imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[3]  # Too Soon Mode
```

- Displays **"Too Soon" Warning UI** if attendance was attempted too quickly.

⏳ Holding Mode 3 for 20 Frames

python

CopyEdit

```
    counter += 1
    if counter >= 20:
        modeType = 0  # Reset to Ready Mode
        counter = 0
        Id = -1
        student_data = None
```

- **Holds Mode 3 for 20 frames** before resetting to **Mode 0 (Ready Mode)**.

---

4 . Displaying UI and Waiting for Key Press

python

CopyEdit

```
cv2.imshow("Face Attendance", imgBackground)
if cv2.waitKey(1) == ord('q'):
    break
```

- **Displays the attendance UI**.
- **Breaks the loop if 'q' is pressed** to close the application.

---

5 . Releasing Camera and Closing Windows

python

CopyEdit

cap.release()

cv2.destroyAllWindows()

- **Releases the camera** (cap.release()) to free system resources.
- **Closes all OpenCV windows** (cv2.destroyAllWindows()).

✅ **Summary of Functionality**

Mode	Description
Mode 0	**Ready Mode** (Waiting for a face)
Mode 1	**Displays Student Info** (Name, ID, Attendance)
Mode 2	**Shows "Attendance Marked Successfully" message**
Mode 3	**Displays "Too Soon" warning** (if attendance is taken too quickly)

CODE :

```
import cv2

import os

import pickle

import face_recognition
```

```
import numpy as np
from pymongo import MongoClient
from datetime import datetime, timezone
import threading
import playsound
import pytz


# MongoDB Connection
client =
MongoClient("mongodb+srv://dasariakhil:201022539002@cluster0.lonxb.mongo
db.net/?retryWrites=true&w=majority&appName=Cluster0")
db = client['LFDCAttendence']
collection = db['Datascience']


# Timezone Setup (Asia/Kolkata)
TIMEZONE = pytz.timezone("Asia/Kolkata")


# Load camera
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)
```

```
# Load background image
bg_path = "C:\\Users\\Akhil Dasari\\facerecognition\\resources\\background\\background.jpg"
imgBackground = cv2.imread(bg_path)

if imgBackground is None:
    print(f"❌ ERROR: Background image not found at {bg_path}. Check the path.")
    exit()

def play_sound(sound_path):
    threading.Thread(target=playsound.playsound, args=(sound_path,), daemon=True).start()

# Load mode images
folderModePath = "resources/modes"
modePathList = os.listdir(folderModePath)

imageModeList = []
for path in modePathList:
    img = cv2.imread(os.path.join(folderModePath, path))
    if img is not None:
        img_resized = cv2.resize(img, (414, 633))
```

```
        imageModeList.append(img_resized)


if len(imageModeList) < 4:
    print(f" ⚠️ Warning: Expected 4 mode images, but found {len(imageModeList)}.")


print(" ✅ Mode images loaded:", len(imageModeList))


# Load encodings
print(" 🔍 Loading encode file...")
with open("EncodeFile.p", "rb") as file:
    encodeListKnown, studentIds = pickle.load(file)
print(" ✅ Encode file loaded")


# Initialize variables
modeType = 0
counter = 0
Id = -1
student_data = None
sound_played = False
last_detected_id = None
```

```
# Play initial sound
play_sound("C:\\Users\\Akhil Dasari\\Downloads\\zapsplat_science_fiction_computer_tone_beep_positive_accepted_83987.mp3")


# Function to get current time in ISO 8601 format (Asia/Kolkata)
def get_current_time():
    # Get the current time in UTC
    current_time_utc = datetime.now(timezone.utc)
    # Convert it to Asia/Kolkata time zone
    current_time_local = current_time_utc.astimezone(TIMEZONE)
    # Return the time in ISO 8601 format (this includes the timezone info)
    return current_time_local.isoformat()


while True:
    success, img = cap.read()
    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

    faceCurFrame = face_recognition.face_locations(imgS)
    encodeCurFrame = face_recognition.face_encodings(imgS, faceCurFrame)
```

```
    imgBackground[162:162 + 480, 55:55 + 640] = img

    # --- Mode 0: Ready to Scan ---
    if modeType == 0:
        imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[2]

        for encodeFace, faceLoc in zip(encodeCurFrame, faceCurFrame):
            matches = face_recognition.compare_faces(encodeListKnown,
encodeFace)
            faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
            matchIndex = np.argmin(faceDis)

            if matches[matchIndex]:
                Id = str(studentIds[matchIndex])
                print("✅ Face Matched. ID:", Id)

                student_data = collection.find_one({"_id": Id})
                current_time = get_current_time()  # Get the current time in ISO 8601
format
                if student_data:
                    last_att_time_str = student_data.get('last_attendance_time', '')
```

```
            print(f"Last attendance time: {last_att_time_str}")  # Debugging: Check if time string is valid
            if last_att_time_str:
                try:
                    # Attempt to convert the time string to a datetime object in ISO 8601 format
                    last_att_time = datetime.fromisoformat(last_att_time_str).astimezone(TIMEZONE)
                    print(f"Last attendance time (converted): {last_att_time}")  # Debugging: Check converted time
                    # Calculate time difference in seconds
                    current_time_obj = datetime.fromisoformat(current_time).astimezone(TIMEZONE)
                    secondsDiff = (current_time_obj - last_att_time).total_seconds()
                    print(f"Seconds difference: {secondsDiff}")  # Debugging: Check if seconds are being calculated
                except ValueError as e:
                    print(f" ⚠️ Error converting time: {e}")  # Print error if there's an issue with the conversion
                    secondsDiff = None
            else:
                secondsDiff = None
            if secondsDiff is None or secondsDiff > 10:  # Only allow attendance if more than 10 seconds have passed
```

```
                student_data['total_attendance'] += 1
                # Update MongoDB
                collection.update_one({"_id": Id}, {
                    "$set": {
                        "total_attendance": student_data['total_attendance'],
                        "last_attendance_time": current_time  # Update last attendance time in ISO 8601 format
                    },
                    "$push": {
                        "attendance_time_log": current_time  # Add new attendance to log
                    }
                })
                modeType = 1  # Move to Mode 1 (Show Student Info)
            else:
                print(f"⏳ Too soon. Last attendance: {student_data['last_attendance_time']}")
                modeType = 3  # Too soon warning (Mode 3)
            counter = 0  # Reset counter after checking attendance
            break
    # --- Mode 1: Show Student Info ---
    elif modeType == 1:
```

```
        imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[3]
        face_path = f"images/{Id}.jpg"
        if os.path.exists(face_path):
            imgStudent = cv2.imread(face_path)
            imgStudent = cv2.resize(imgStudent, (250, 250))
            imgBackground[180:430, 900:1150] = imgStudent
        else:
            print("⚠️ No face image found for student:", Id)
        if student_data:
            text_x = 950
            image_top = 430
            line_spacing = 40
            # Display student's name at the given coordinates
            cv2.putText(imgBackground, student_data['name'], (text_x, image_top +
line_spacing),
                        cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 255, 255), 1)
            # Display student's group at the given coordinates
            cv2.putText(imgBackground, f"Group: {student_data['group']}", (text_x,
image_top + 2 * line_spacing),
                        cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 255, 255), 1)
            # Display student's ID at the given coordinates
```

```
            cv2.putText(imgBackground, f"ID: {student_data['_id']}", (text_x,
image_top + 3 * line_spacing),
                        cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 255, 255), 1)

            # Display student's year exactly below the ID
            cv2.putText(imgBackground, f"Year: {student_data['year']}", (text_x,
image_top + 4 * line_spacing),
                        cv2.FONT_HERSHEY_COMPLEX, 0.6, (225, 225, 225), 1)

            # Display Total Attendance (TP) at the new coordinates
            cv2.putText(imgBackground, f"TP: {student_data['total_attendance']}",
(861, 125),
                        cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 255, 255), 1)

            formatted_time = datetime.now().strftime("%I:%M:%S %p")
            cv2.putText(imgBackground, formatted_time, (1050, 150),
                        cv2.FONT_HERSHEY_COMPLEX, 0.6, (255, 255, 255), 1)
        counter += 1
        if counter >= 6:
            modeType = 2
            counter = 0
    # --- Mode 2: End Message ---
```

```
elif modeType == 2:
    imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[0]
    if Id != last_detected_id or last_detected_id is None:
        last_detected_id = Id
        sound_played = False
    # Play sound only if it hasn't been played for this person
    if not sound_played and last_detected_id == Id:
        play_sound("c:\\Users\\Akhil Dasari\\Downloads\\p_42214563_35.mp3")
        sound_played = True  # Prevent repeated sounds for the same person

    counter += 1
    if counter >= 6:
        modeType = 0
        counter = 0
        Id = -1
        student_data = None
# --- Mode 3: Too Soon Warning ---
elif modeType == 3:
    imgBackground[44:44 + 633, 808:808 + 414] = imageModeList[1]
    if Id != last_detected_id or last_detected_id is None:
        last_detected_id = Id
```

```
            sound_played = False
        # Play sound only if it hasn't been played for this person
        if not sound_played and last_detected_id == Id:
            play_sound("c:\\Users\\Akhil Dasari\\Downloads\\p_42214563_35.mp3")
            sound_played = True  # Prevent repeated sounds for the same person
        counter += 1
        if counter >= 6:
            modeType = 0
            counter = 0
            Id = -1
            student_data = None
    cv2.imshow("Face Attendance", imgBackground)
    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

MONGODB CODE :

```
from pymongo import MongoClient

from datetime import datetime

import pytz

# 🎯 Connect to MongoDB

client =
MongoClient("mongodb+srv://dasariakhil:201022539002@cluster0.lonxb.mongo
db.net/?retryWrites=true&w=majority&appName=Cluster0")

db = client["LFDCAttendence"]

collection = db["Datascience"]

# ✅ Fixed Time Zone for All Students (Asia/Kolkata)

TIMEZONE = pytz.timezone("Asia/Kolkata")

# ✅ Function to get current local time in ISO 8601 format (with timezone)

def get_local_time():

    # Get current UTC time

    utc_now = datetime.now(pytz.utc)

    # Convert UTC time to local time (Asia/Kolkata)

    local_time = utc_now.astimezone(TIMEZONE)

    return local_time

# ✅ Student Data (Initial)

ourdata = [
```

```
{
    "_id": "201022539002",
    "name": "Dasari Akhil",
    "group": "MSDS-3",
    "total_attendance": 0,
    "year": "2022-2025",
    "last_attendance_time": None,
    "attendance_time_log": []
},
{
    "_id": "201022539003",
    "name": "JA",
    "group": "MSDS",
    "total_attendance": 7,
    "year": "3",
    "last_attendance_time": None,
    "attendance_time_log": []
},
{
    "_id": "201022539004",
    "name": "VK",
```

```
        "group": "MSDS",
        "total_attendance": 7,
        "year": "3",
        "last_attendance_time": None,
        "attendance_time_log": []
    },
    {
        "_id": "201022539005",
        "name": "Asad",
        "group": "MSDS",
        "total_attendance": 8,
        "year": "3",
        "last_attendance_time": None,
        "attendance_time_log": []
    }
]
# 🔄 Insert/Update Data in MongoDB
for student in ourdata:
    collection.update_one({"_id": student["_id"]}, {"$set": student}, upsert=True)

print("✅ Student data inserted/updated successfully.")
```

```
# 📌 Function to Mark Attendance (With ISO 8601 Format)
def mark_attendance(student_id):
    student = collection.find_one({"_id": student_id})

    if student:
        # Get current UTC time and convert it to local time
        current_time = get_local_time()

        # Convert the local time to ISO 8601 format for storage
        current_time_iso = current_time.isoformat()  # ISO 8601 format

        # Get last attendance time from database
        last_att_time_str = student.get('last_attendance_time', None)

        if last_att_time_str:
            try:
                # Parse last attendance time stored in ISO 8601 format
                last_att_time = datetime.fromisoformat(last_att_time_str)
                current_time_obj = get_local_time()
                time_diff = (current_time_obj - last_att_time).total_seconds()
```

```
            if time_diff < 60:
                print(f" ⏳ Too soon! {student['name']}'s last attendance: {last_att_time_str}")
                return
        except Exception as e:
            print(f" ⚠️ Error processing {student['name']}'s attendance: {e}")
            last_att_time = None

    # Increase attendance count
    new_attendance_count = student["total_attendance"] + 1

    # Convert existing datetime objects in attendance_time_log to ISO 8601 format
    attendance_time_log = student.get("attendance_time_log", [])
    updated_log = []
    for entry in attendance_time_log:
        if isinstance(entry, datetime):
            updated_log.append(entry.isoformat())  # Convert to ISO format
        else:
            updated_log.append(entry)
    # Add the new attendance log entry (in ISO 8601 format)
    updated_log.append(current_time_iso)
```

```
        # Update MongoDB with the new attendance count and log
        collection.update_one(
            {"_id": student_id},
            {
                "$set": {
                    "total_attendance": new_attendance_count,
                    "last_attendance_time": current_time_iso  # Store in ISO format
                },
                "$push": {"attendance_time_log": current_time_iso}  # Store in ISO
format
            }
        )
        # Print success message
        print(f"✅ Attendance marked for {student['name']} at {current_time_iso}")
    else:
        print(f"❌ Student with ID {student_id} not found!")
# Example: Mark attendance for a student
mark_attendance("201022539002")
# Example: Retrieve and display student data after attendance
vk_data = collection.find_one({"_id": "201022539004"})
```

```
print(vk_data)


# ✅ Close MongoDB Connection

client.close()
```

EXCEL CONVERTION :

```
import os

import pandas as pd

from pymongo import MongoClient

import openpyxl

from datetime import datetime

# MongoDB Connection

client =
MongoClient("mongodb+srv://dasariakhil:201022539002@cluster0.lonxb.mongo
db.net/?retryWrites=true&w=majority&appName=Cluster0")

db = client['LFDCAttendence']

collection = db['Datascience']

# Fetch attendance data

data = list(collection.find({}, { "name": 1, "_id": 1, "group": 1, "year": 1,
"total_attendance": 1, "attendance_time_log": 1}))

# Convert MongoDB data to Pandas DataFrame

df = pd.DataFrame(data)

# Rename '_id' to 'Student ID' for better readability
```

```
df.rename(columns={'_id': 'Student ID', 'total_attendance': 'Total Attendance',
'attendance_time_log': 'Attendance Log'}, inplace=True)

# Ensure the correct column order

df = df[['name', 'Student ID', 'group', 'year', 'Total Attendance', 'Attendance Log']]

# Get current date in 'YYYY-MM-DD' format

current_date = datetime.now().strftime('%Y-%m-%d')

# Specify custom folder path (ensure folder exists)

folder_path = "C:\\Users\\Akhil Dasari\\OneDrive\\Desktop"

# Create file name with current date

file_path = f"{folder_path}Attendance_Report_{current_date}.xlsx"


# Save to Excel file

df.to_excel(file_path, index=True, engine="openpyxl")

print(f"✅ Attendance data saved successfully to {file_path}")
```

Conclusion

This Face Attendance System is a robust solution for automated student attendance management. The system utilizes advanced face recognition technology to identify students and log their attendance in real-time, making it efficient and convenient for both instructors and students.

Key features of the system include:

1. **Face Recognition Integration**: The system recognizes students using their facial features, which are matched against a pre-existing set of known encodings stored in a pickle file.
2. **MongoDB Integration**: The system stores student data, attendance logs, and other relevant information in a MongoDB database, ensuring data consistency and security. Each student's attendance is recorded with the current time in ISO 8601 format (Asia/Kolkata timezone), allowing easy tracking and retrieval.
3. **Mode-based User Interface**: The user interface is designed around different modes that guide the system's operation:
    - **Mode 0**: Ready to scan for student faces.
    - **Mode 1**: Display student information once a match is found.
    - **Mode 2**: Display an end message and reset after a few seconds.
    - **Mode 3**: Warns the user if the attendance is too soon (i.e., less than 10 seconds from the last recorded attendance).
4. **Sound Alerts**: The system uses sound notifications to alert users when a student's attendance is successfully recorded or when there's an issue such as attempting to mark attendance too soon.
5. **Real-Time Updates**: Once a student is recognized, the system checks the last attendance time and ensures a minimum 10-second gap between

consecutive attendances. The student's attendance is updated accordingly, and the last attendance time is logged for future reference.

This system can be customized further to fit different needs, such as integrating it with a larger student management system, refining the user interface, or adding additional features like web dashboards for attendance monitoring.

Overall, this project serves as a modern, efficient, and user-friendly solution for automating attendance tracking, eliminating the need for manual roll-calls and reducing administrative workload.