NILE UNIVERSITY

AI face recognition attendance system with SQL

Project Report

By

Mostafa Hossam

231000560

Zeyad Abdelhaleem

231000483

Amr Khaled

231000168

Salah Abdelrahman

231000210

Project source code on GitHub repository

link : https://github.com/Not-Mostafa/Face_recognition

Project video link :

https://drive.google.com/file/d/12t0ztGhGIPuFZhdsSOMOLbM6D_gAb-UD/view?usp=sharing

Abstract:

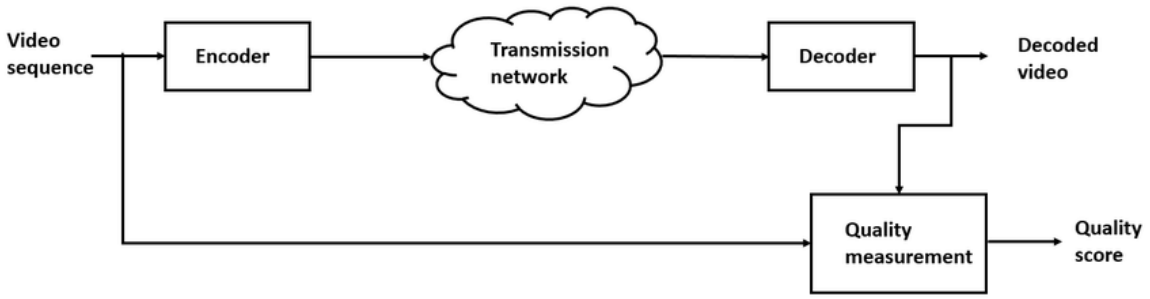
This project presents a Face Recognition Attendance System designed to automate and streamline attendance tracking in educational institutions or workplaces. Traditional manual attendance methods are time-consuming and prone to errors, leading to inefficiencies in record-keeping. To address these challenges, this system leverages computer vision and machine learning for real-time face detection and recognition, coupled with a database management system for secure and efficient data storage.
The system captures live video feeds or images, processes them using OpenCV and deep learning models (such as ***opencv*** or ***Dlib***), and matches detected faces against a pre-registered database of individuals. Upon successful recognition, attendance is automatically logged into a relational database ***SQL Server*** recording timestamps, student/employee details, and attendance status. Administrators can generate reports, manage user profiles, and monitor attendance trends through a user-friendly dashboard.

Image encoding:

Intro:

Attendance tracking is a critical process in educational institutions and workplaces, but traditional methods such as manual roll calls or paper-based systems are inefficient, time-consuming, and susceptible to errors or proxy attendance. With advancements in artificial intelligence (AI) and computer vision, automated attendance systems using face recognition technology have emerged as a reliable and secure alternative.

This project proposes a Face Recognition-Based Attendance System that leverages deep learning algorithms and database management to accurately identify individuals and record attendance in real time. The system eliminates the need for physical interaction, reduces administrative overhead, and ensures tamper-proof records. By integrating a structured database, the system efficiently stores and retrieves student/employee data, attendance logs, and analytics for better decision-



2. Types of Image Encoding:

A. Lossless Encoding

Preserves all original data – no quality loss.

Used where exact reproduction is critical (medical imaging, archival).

Examples:

PNG (Portable Network Graphics) – Uses DEFLATE compression.

GIF (Graphics Interchange Format) – Limited to 256 colors.

TIFF (Tagged Image File Format) – Supports layers and metadata.

B. Lossy Encoding

Discards some data to achieve higher compression.

Used in web images, video streaming, and digital photography.

Examples:

JPEG (Joint Photographic Experts Group) – Uses Discrete Cosine Transform (DCT).

WebP – Google's format, 30% smaller than JPEG.

HEIC/HEIF – Apple's high-efficiency format.

Methodology

System Architecture

The system follows a **client-server architecture** with three main components:

Front end:

Face recognition Gui: tkinter

Login and database webpage : html/java

Backend:

Database backend for the Gui – python fast Api for the html

Database :

MySQL database using Microsoft SQL server .

Setup & Configuration

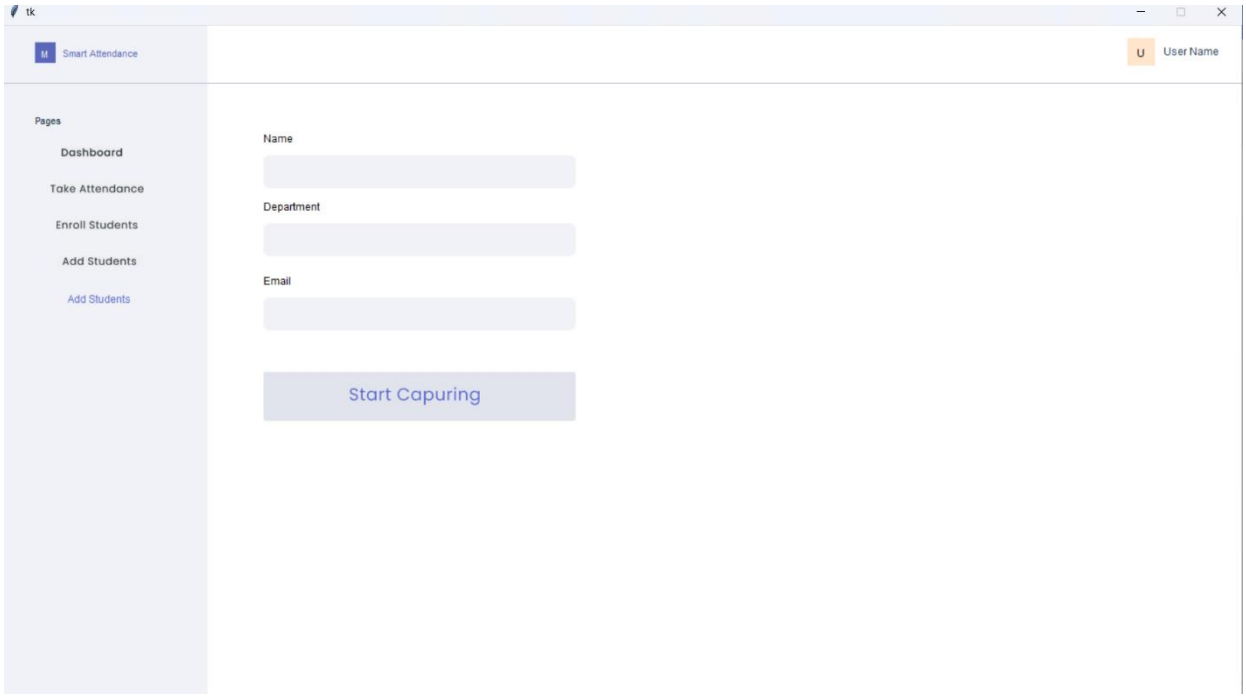
1. **Environment Setup**
   - Install Python 3.9 (required for dlib compatibility)
   - Create virtual environment
   - Install dependencies (OpenCV, face_recognition, Flask, MySQL connector)
2. **Database Configuration**
   - Install MySQL Server 8.0+
   - Create face_attendance database
   - Configure in python code

Project Features:

1. **Image Capture Module (Photo_Capture.py)**
   - Uses OpenCV to access webcam
   - Implements face detection with Haar Cascades
   - Saves images in standardized format (userID_timestamp.jpg)
2. **Encoding Module (Image_Encoder.py)**
   - Uses face recognition library to generate 128D embeddings
   - Implements duplicate detection by comparing new encodings with database
   - Stores encodings in MySQL as BLOB data
3. **Recognition Module (Image_Comparison.py)**
   - Real-time face matching using cosine similarity
   - Configurable threshold (default: 0.5)
   - Logs recognized faces to attendance table

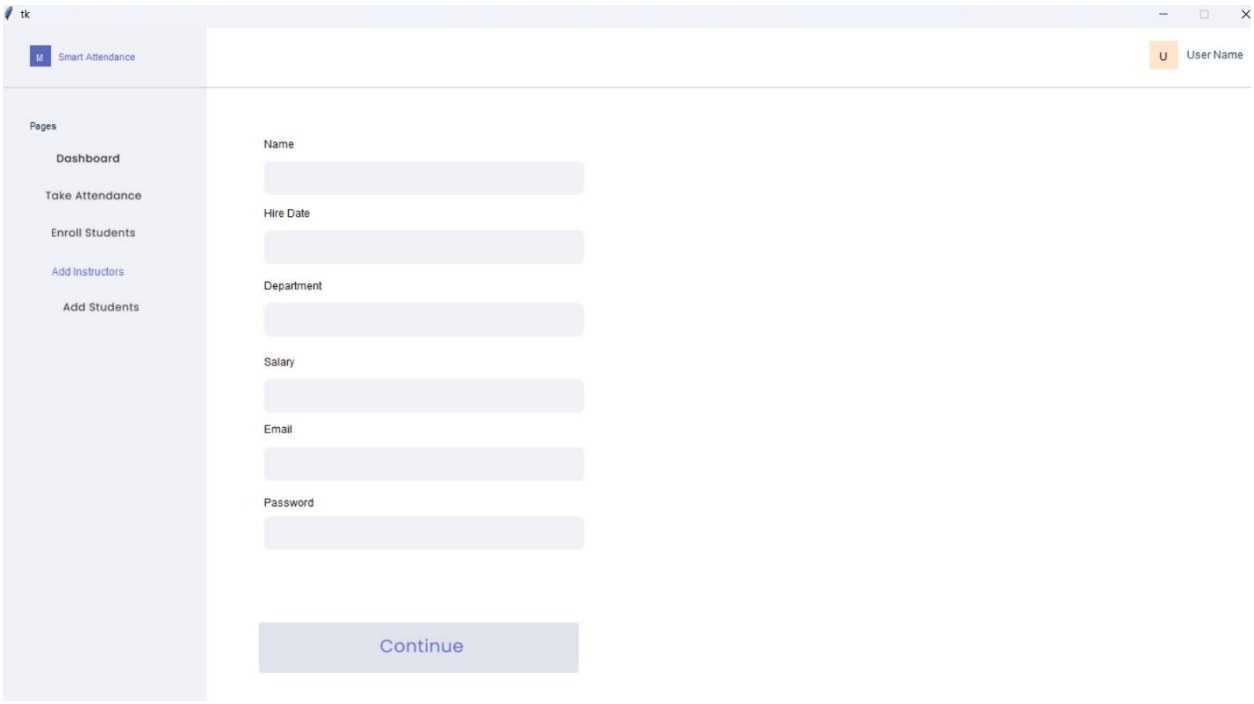
4. Add student to db (add_student_gui.py)

- Takes student input (student name – email – department name)
- Capture student face
- Encodes the student face and check if he was matching with a previous encoding or not if it's matching it doesn't insert the student into the sql database if not matching with a previous encoded image it inserts the taken student entry from the gui (name , email , department) using the query **Insert into students values (','?','?','?')**



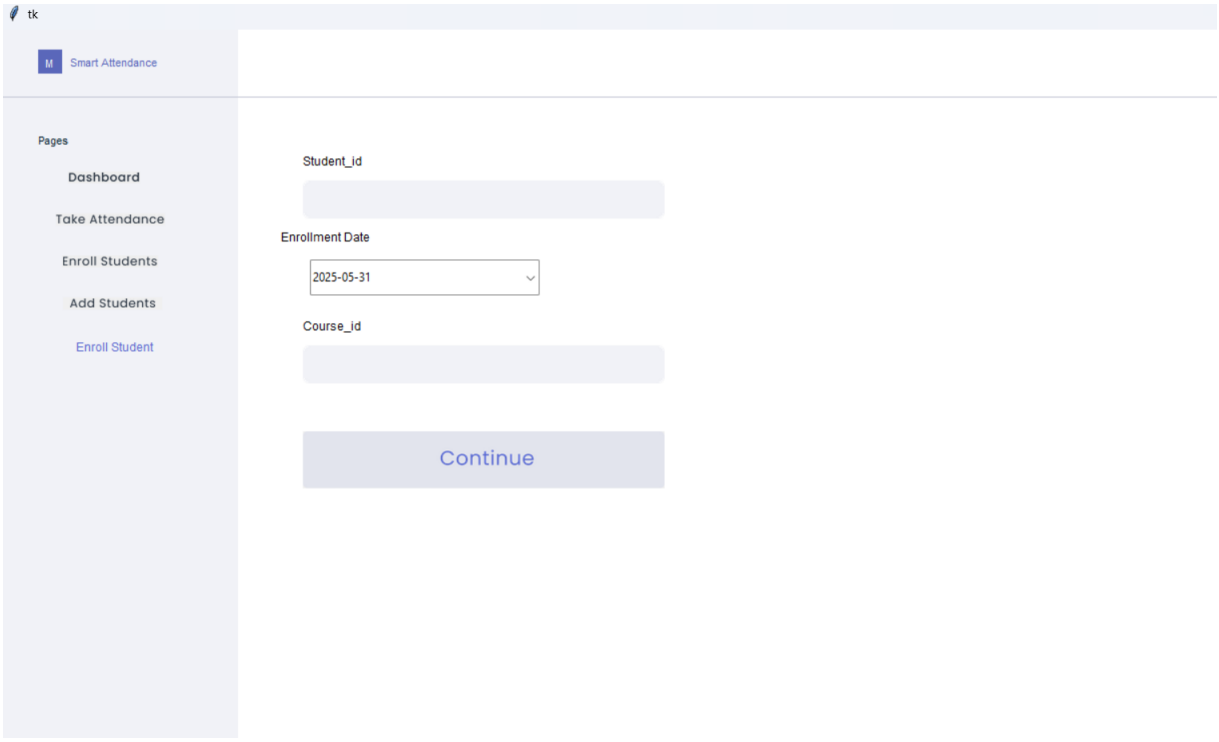
5. **<u>Add to instructor (add to instructor gui.py)</u>**

- Takes instructor input (name – email – department – salary – hire date – password) .
- Capture instructor face
- Encodes the instructor face and check if he was matching with a previous encoding or not if it's matching it doesn't insert the student into the sql database if not matching with a previous encoded image it inserts the taken insturctor entry from the gui (name , email , department , etc …) using the query **Insert into students values (','?','?','?')**



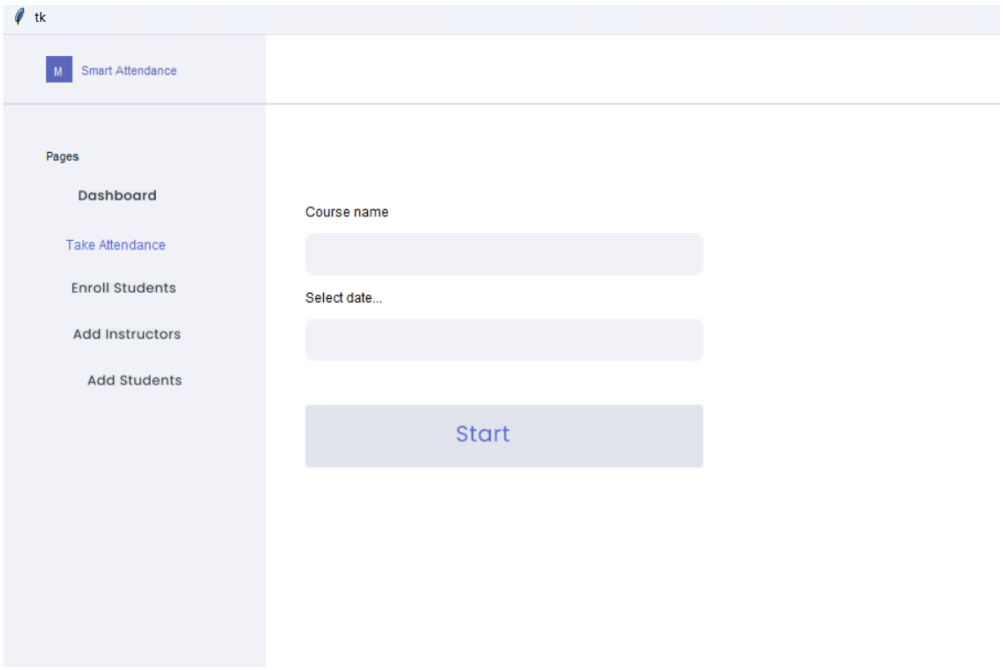
6. Enroll students (enroll student gui.py):

- Takes student entries (student id , enrollment date , course id)
- Makes SQL query to insert into the Database



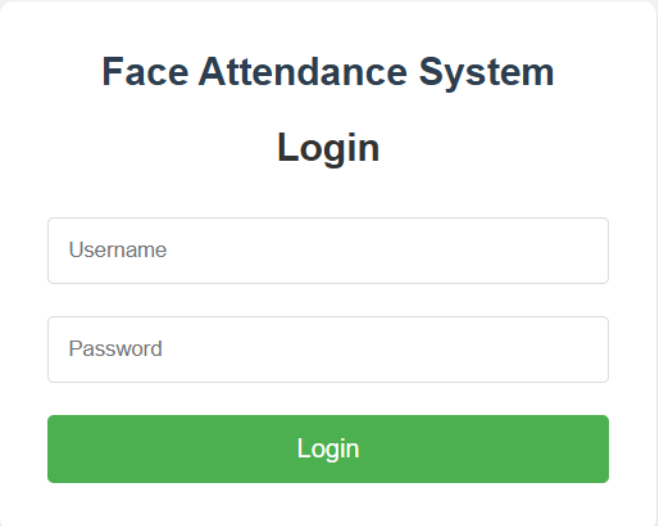
7. **Take attendance (take attendance gui.py);**

- Students select the course and attendance date then presses capture
- A photo of the student is captured
- If it matches a previous encoded image it takes the matching id number and insert into the attendance table the data and set the attendance status intro present .
- If It doesn't match it sets the student status into absent .



	attendance_id	student_id	course_id	attendance_date	timestamp_in	status
1	30	1	40	2025-05-31	NULL	Present
2	31	1	40	2025-05-31	NULL	Present

Html login page :

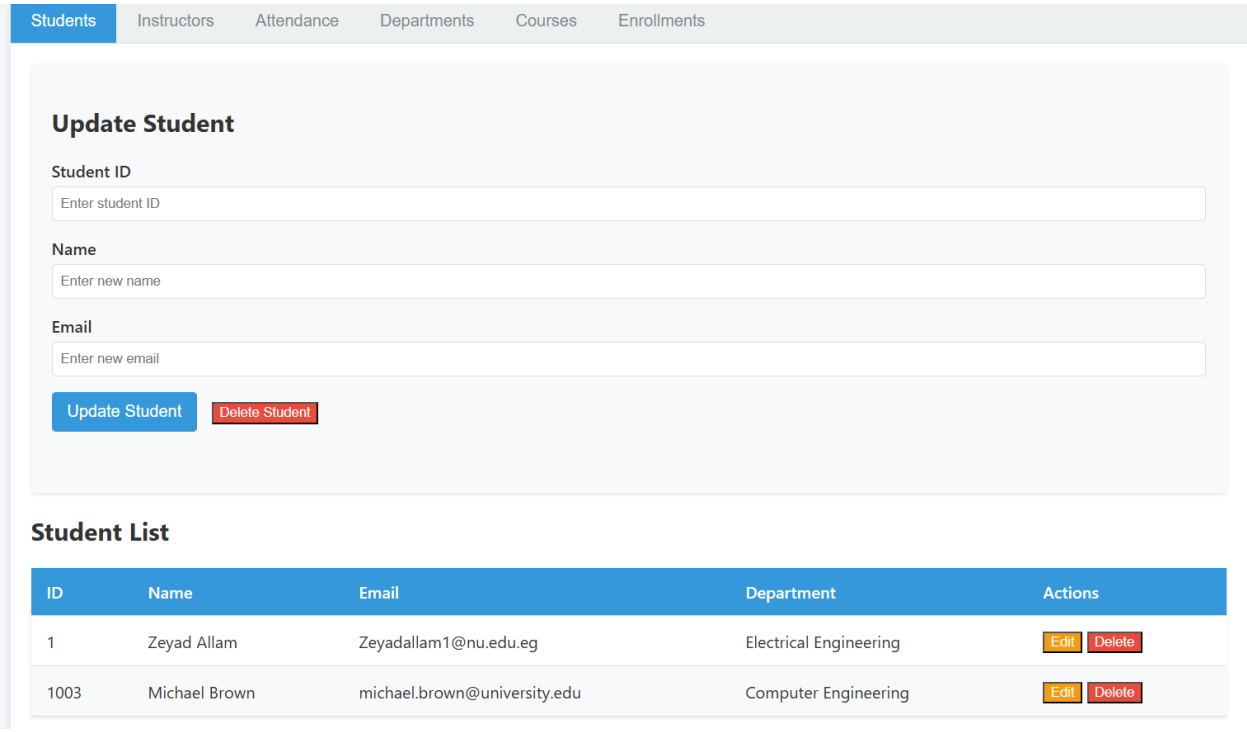


Dashboard (html + java + python backend server):

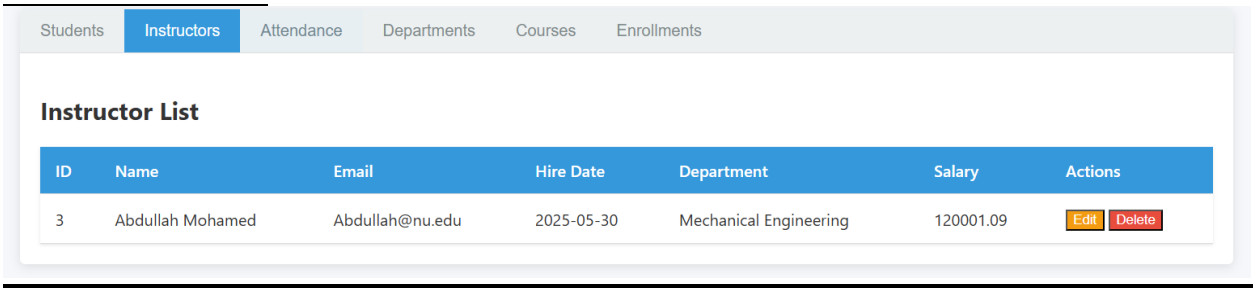
Crude functions:

Select all (students , instructors , attendance , enrollments , departments , courses)
Insert into (courses , departments)
Delete (students , instructors , courses , departments)
Update (courses , students , departments , instructors)

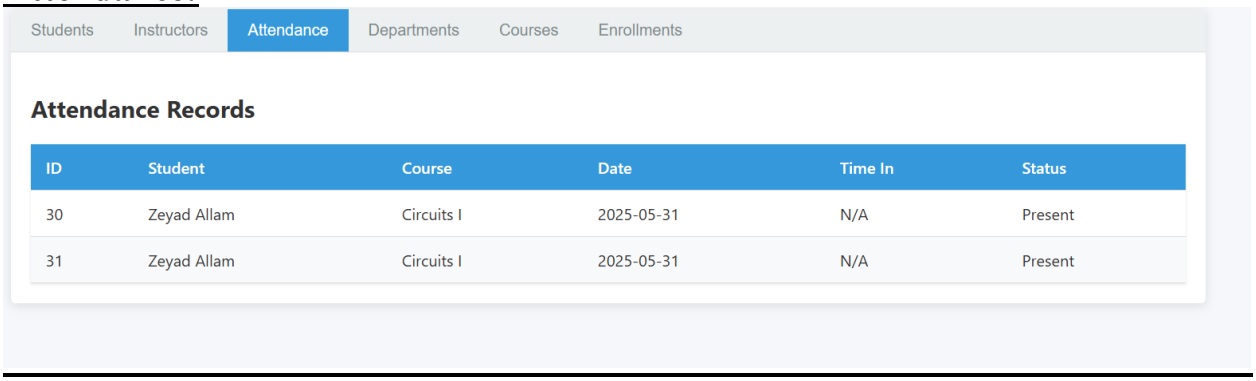
Students table:



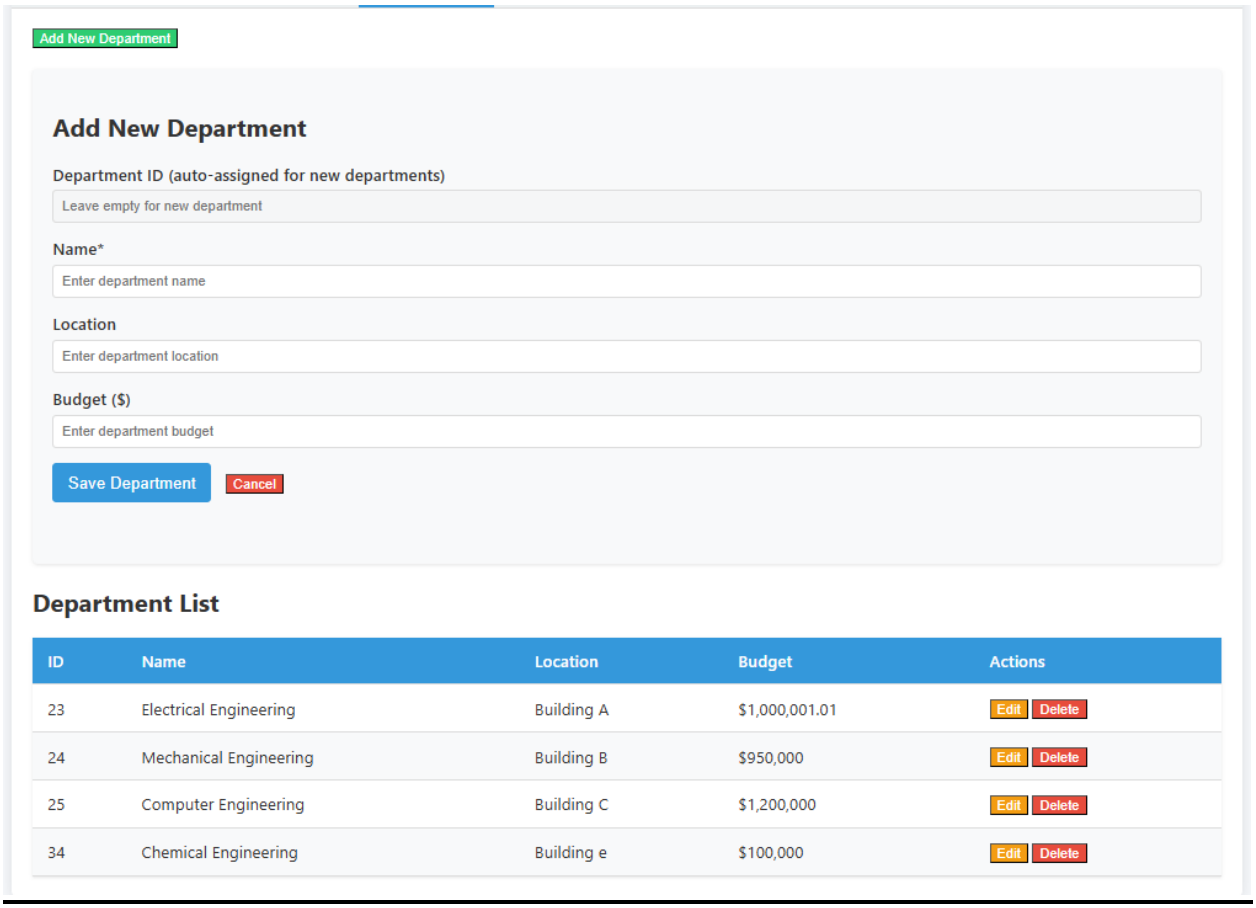
Instructor table:



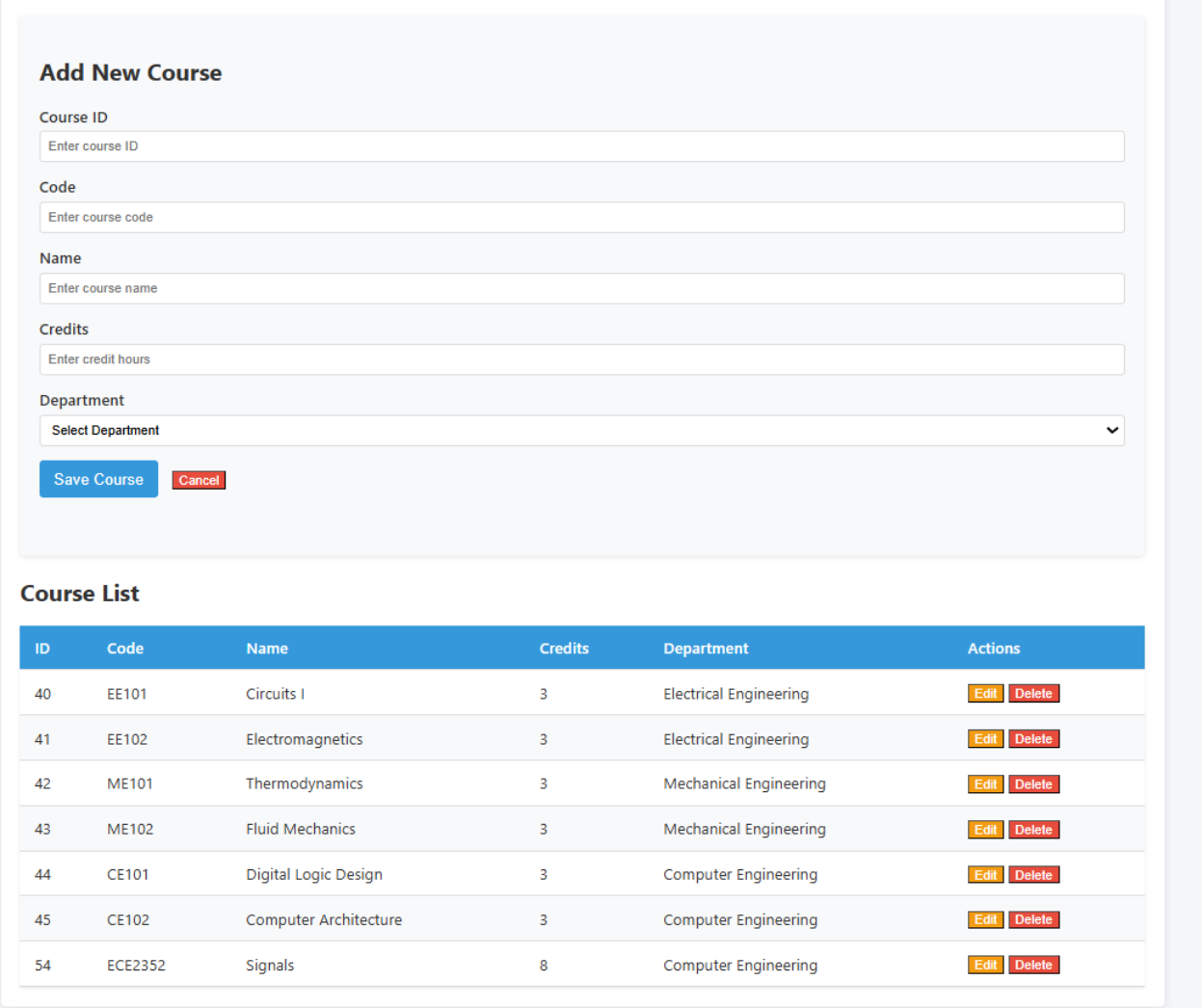
Attendance:



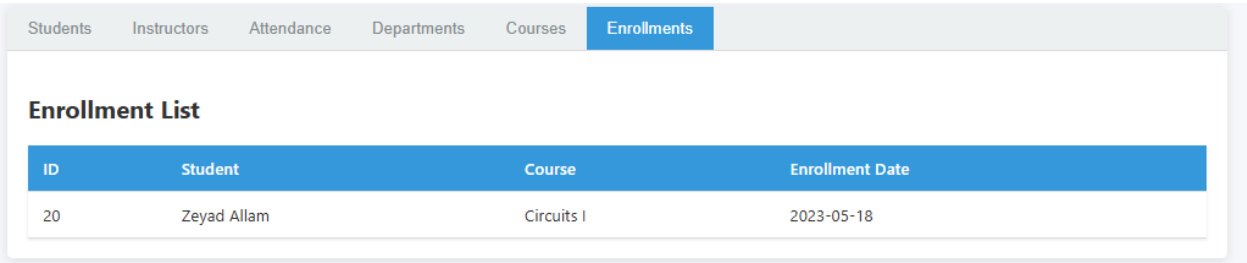
Department:


Add New Department
Add New Department
Department ID (auto-assigned for new departments)
Leave empty for new department
Name*
Enter department name
Location
Enter department location
Budget ($)
Enter department budget
Save Department
Cancel
Department List
ID
Name
Location
Budget
Actions
23
Electrical Engineering
Building A
$1,000,001.01
Edit
Delete
24
Mechanical Engineering
Building B
$950,000
Edit
Delete
25
Computer Engineering
Building C
$1,200,000
Edit
Delete
34
Chemical Engineering
Building e
$100,000
Edit
Delete

Courses:



Enrollments:



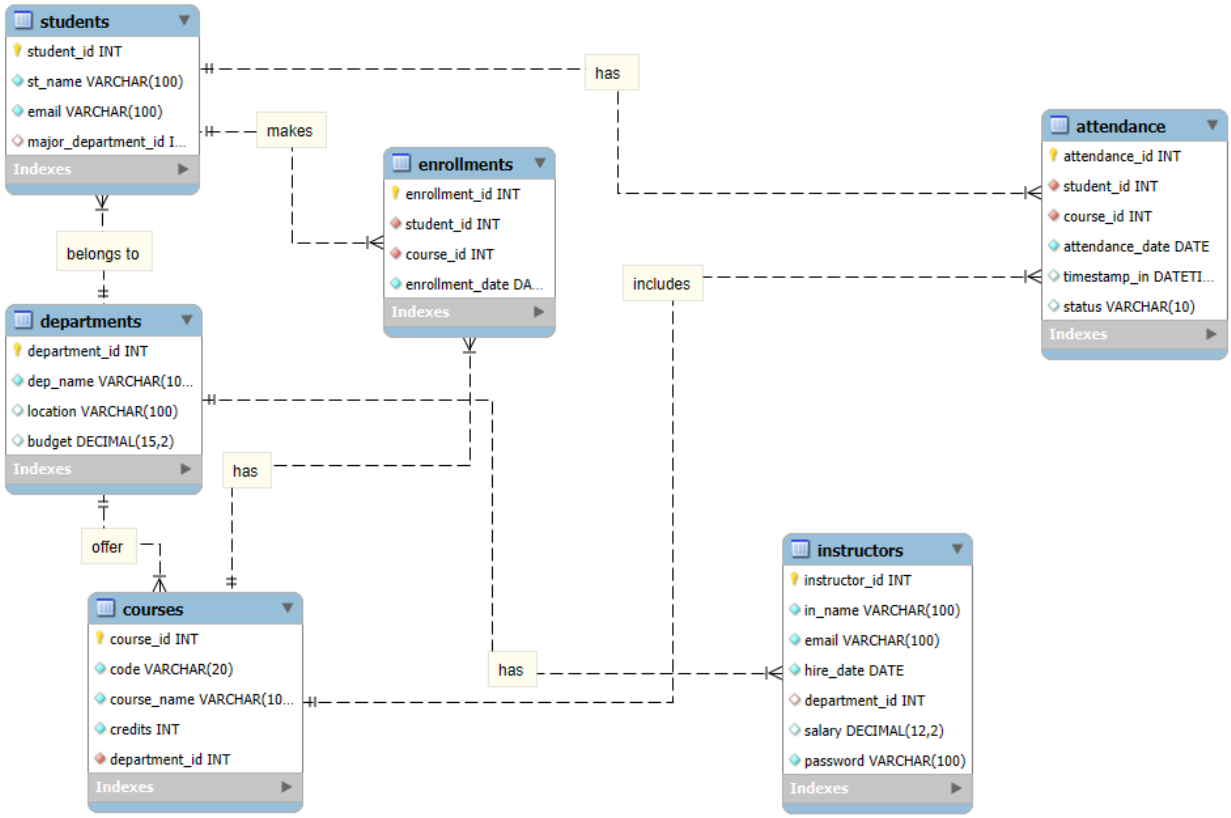
Database code:

```
CREATE DATABASE face_attendance;
USE face_attendance;

-- Create Departments table
```

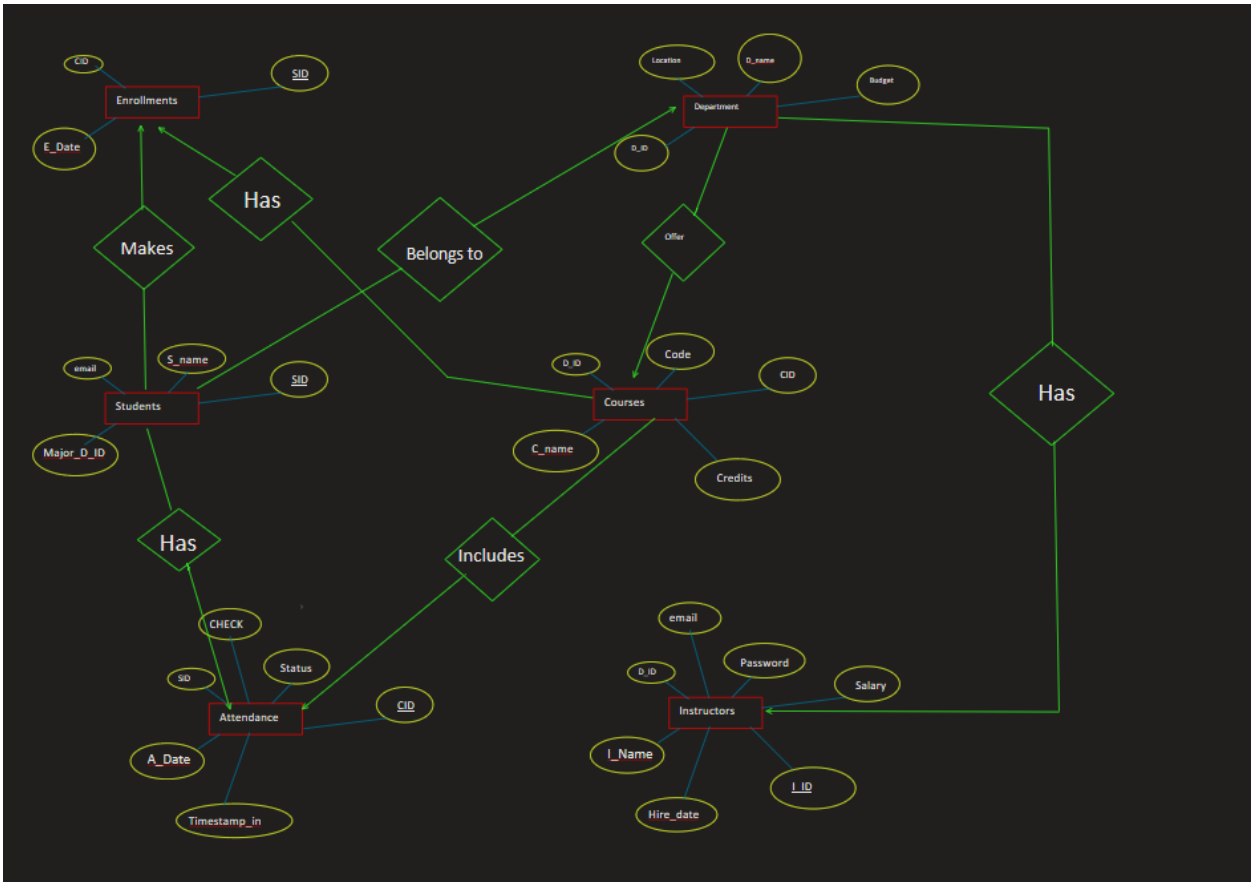
```
CREATE TABLE Departments (
    department_id INT PRIMARY KEY IDENTITY(1,1),
    dep_name VARCHAR(100) NOT NULL,
    location VARCHAR(100),
    budget DECIMAL(15,2)
);

-- Create Instructors table
CREATE TABLE Instructors (
    instructor_id INT PRIMARY KEY IDENTITY(1,1),
    in_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    hire_date DATE NOT NULL,
    department_id INT,
    salary DECIMAL(12,2),
    password VARCHAR(100) NOT NULL DEFAULT 'default123',
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);

-- Create Courses table
CREATE TABLE Courses (
    course_id INT PRIMARY KEY IDENTITY(1,1),
    code VARCHAR(20) UNIQUE NOT NULL,
    course_name VARCHAR(100) NOT NULL,
    credits INT NOT NULL,
    department_id INT NOT NULL,
    FOREIGN KEY (department_id) REFERENCES Departments(department_id)
);

-- Create Students table
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    st_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    major_department_id INT,
    FOREIGN KEY (major_department_id) REFERENCES
Departments(department_id)
);

-- Create Enrollments table (no sections, just direct course enrollments)
```

```
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY IDENTITY(1,1),
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    enrollment_date DATE NOT NULL,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

-- Create Attendance table (no sections, directly related to courses)
CREATE TABLE Attendance (
    attendance_id INT PRIMARY KEY IDENTITY(1,1),
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    attendance_date DATE NOT NULL,
    timestamp_in DATETIME,
    status VARCHAR(10) DEFAULT 'Absent',
    CHECK (status IN ('Present', 'Absent')),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

Database ERD 1nf :



Database ERD 2nf and 3nf:



Sql Queries used in gui :

Add student gui:

SELECT department_id FROM Departments WHERE dep_name = ?
INSERT INTO students (student_ID, st_name, Email, major_department_id)
VALUES (?, ?, ?, ?)

Add instructor gui

SET IDENTITY_INSERT Instructors ON;
INSERT INTO Instructors
(instructor_id, in_name, hire_date, department_id, salary, email, password)
VALUES (?, ?, ?, ?, ?, ?, ?);
SET IDENTITY_INSERT Instructors OFF;

Take_attendance gui:

SELECT course_id FROM Courses WHERE course_name = ?
SELECT E.student_id FROM Enrollments E WHERE E.course_id = ?
INSERT INTO Attendance (student_id, course_id, attendance_date, status)
VALUES (?, ?, ?, 'Absent') – if not matching an encoded image
UPDATE Attendance
SET status = 'Present'
WHERE student_id = ? AND course_id = ? AND attendance_date = ? – if matching an encoded image

Enroll students gui:

SELECT 1 FROM Students WHERE student_id = ?
SELECT 1 FROM Courses WHERE course_id = ?
SELECT 1 FROM Enrollments WHERE student_id = ? AND course_id = ?
INSERT INTO Enrollments (student_id, course_id, enrollment_date)
VALUES (?, ?, ?)

Notes:

- All queries use parameterized inputs (?) for security
- The Cdb_query() function reuses the same connection for transaction batches (like attendance marking)
- Date handling uses Python's datetime.now() formatted as YYYY-MM-DD
- The IDENTITY_INSERT is explicitly managed for instructor insertion

Html/dashboard Queries:

Students:

SELECT s.student_id, s.st_name as name, s.email, d.dep_name as department_name
FROM Students s
LEFT JOIN Departments d ON s.major_department_id = d.department_id

Instructors:

SELECT i.instructor_id, i.in_name as name, i.email, i.hire_date, i.salary, d.dep_name as department_name
FROM Instructors i
LEFT JOIN Departments d ON i.department_id = d.department_id

Enrollments:

```
SELECT e.enrollment_id, e.student_id, s.st_name as student_name,
    e.course_id, c.course_name, e.enrollment_date
FROM Enrollments e
JOIN Students s ON e.student_id = s.student_id
JOIN Courses c ON e.course_id = c.course_id
```

Attendance:

```
SELECT a.attendance_id, a.student_id, s.st_name as student_name,
  a.course_id, c.course_name, a.attendance_date, a.timestamp_in, a.status
FROM Attendance a
 JOIN Students s ON a.student_id = s.student_id
 JOIN Courses c ON a.course_id = c.course_id
```

Courses:

```
SELECT c.course_id, c.course_name
FROM Courses c
```

3. Student Endpoints

Get All Students:

```
SELECT s.student_id, s.st_name as name, s.email, d.dep_name as
department_name
 FROM Students s
 LEFT JOIN Departments d ON s.major_department_id = d.department_id
```

Update Student (Dynamic):

```
UPDATE Students SET st_name = ?, email = ?, major_department_id = ?
WHERE student_id = ?
```

Delete Student:

```
DELETE FROM Enrollments WHERE student_id = ?
DELETE FROM Attendance WHERE student_id = ?
DELETE FROM Students WHERE student_id = ?
```

4. Instructor Endpoints

Get All Instructors:

```
SELECT i.instructor_id, i.in_name as name, i.email, i.hire_date, i.salary,
d.dep_name as department_name, i.department_id
FROM Instructors i
LEFT JOIN Departments d ON i.department_id = d.department_id
```

Update Instructor (Dynamic):
UPDATE Instructors SET in_name = ?, email = ?, hire_date = ?,
salary = ?, department_id = ? WHERE instructor_id = ?

Delete Instructor:
DELETE FROM Instructors WHERE instructor_id = ?

5. Department Endpoints

Get All Departments:
SELECT department_id, dep_name as name, location, budget
FROM Departments;

Create Department:
INSERT INTO Departments (dep_name, location, budget) VALUES (?, ?, ?)

Update Department (Dynamic):
UPDATE Departments SET dep_name = ?, location = ?, budget = ?
WHERE department_id = ?

Delete Department (With Checks):
SELECT (SELECT COUNT(*) FROM Courses WHERE department_id = ?) as course_count,
(SELECT COUNT(*) FROM Instructors WHERE department_id = ?) as instructor_count,
(SELECT COUNT(*) FROM Students WHERE major_department_id = ?) as student_count
DELETE FROM Departments WHERE department_id = ?

6. Course Endpoints

Get All Courses:
SELECT c.course_id, c.code, c.course_name as name, c.credits,
c.department_id, d.dep_name as department_name
FROM Courses c
LEFT JOIN Departments d ON c.department_id = d.department_id

Create Course:
INSERT INTO Courses (code, course_name, credits, department_id) VALUES (?, ?, ?, ?);

Update Course:
UPDATE Courses SET code = ?, course_name = ?, credits = ?, department_id = ?
WHERE course_id = ?

Delete Course:

```
DELETE FROM Enrollments WHERE course_id = ?
DELETE FROM Attendance WHERE course_id = ?
DELETE FROM Courses WHERE course_id = ?
```

7. Enrollment Endpoints

Get All Enrollments:

```
SELECT e.enrollment_id, e.student_id, s.st_name as student_name,
  e.course_id, c.course_name, e.enrollment_date
 FROM Enrollments e
OIN Students s ON e.student_id = s.student_id
 JOIN Courses c ON e.course_id = c.course_id
```

Update Enrollment:

```
UPDATE Enrollments SET course_id = ? WHERE enrollment_id = ?
```

8. Attendance Endpoints

Get All Attendance Records:

```
SELECT a.attendance_id, a.student_id, s.st_name as student_name,
 a.course_id, c.course_name, a.attendance_date, a.timestamp_in, a.status
FROM Attendance a
JOIN Students s ON a.student_id = s.student_id
JOIN Courses c ON a.course_id = c.course_i
```

Conclusion :

The Face Recognition Attendance System successfully automates attendance tracking using modern computer vision and database technologies. By integrating OpenCV, face_recognition, and MySQL, the system provides:

- Accurate, real-time attendance marking without manual intervention
- Efficient user management through a Tkinter GUI and web dashboard
- Secure data storage with duplicate prevention and configurable thresholds
- Scalable architecture suitable for schools, offices, and events

Future improvements could include mobile app support, liveness detection, and cloud deployment for wider accessibility. This project demonstrates how AI-driven solutions can streamline administrative tasks while maintaining reliability and security.

Final Outcome: A fully functional attendance system that replaces traditional methods with faster, error-free face recognition technology.