Automated Attendance System

Chapter 1: Introduction

1.1 Project Overview

This project focuses on the design, development, and implementation of an Automated Attendance System using facial recognition technology. The system aims to streamline the process of recording attendance in environments such as educational institutions or workplaces, offering a more efficient, accurate, and secure alternative to traditional manual methods. It involves capturing facial images via a camera, identifying individuals against a registered database, and logging attendance records automatically.

1.2 Problem Statement

Manual attendance systems, including sign-in sheets and roll calls, suffer from significant drawbacks. They are labor-intensive, susceptible to human errors (e.g., incorrect marking, illegible entries), time-consuming, and vulnerable to fraudulent practices like proxy attendance . These inaccuracies can lead to incorrect payroll calculations, unfair assessments, and inefficient resource management. Furthermore, manual systems lack the capability for real-time monitoring and instant report generation.

1.3 Objectives

The primary objectives of this project were:

1. To research and analyze different automated attendance technologies, selecting the most suitable one (Facial Recognition).
2. To design a robust system architecture incorporating software components (backend, frontend, database).
3. To develop core modules for user registration (including face capture and encoding), real-time facial recognition, and attendance logging.
4. To implement a secure database for storing user information and attendance records.
5. To create a user-friendly web interface for system administration, user management, and report generation.
6. To test the system thoroughly for accuracy, performance, security, and usability.
7. To document the entire process, including design, implementation, testing, and potential deployment strategies.

1.4 Scope and Limitations

In Scope:

- Development of a web-based application for administration.
- User registration module with facial data capture.
- Real-time facial detection and recognition using a connected camera.
- Automated attendance logging upon successful recognition.
- Storage of user profiles and attendance logs in a database.
- Generation of basic attendance reports (e.g., daily, user-specific).

Limitations:

- Recognition accuracy may be affected by significant variations in lighting, facial expressions, pose angles, and occlusions (e.g., masks, glasses).
- The system performance depends on the quality of the camera and the processing power of the host machine.
- The current version assumes a relatively controlled environment for optimal performance.

Chapter 2: Literature Review

2.1 DeepFace by Facebook (2014)

2.2 Viola-Jones Face Detection Framework (2001)

2.3 A Survey on Face Recognition Systems (2020)

Chapter 3: Technology Stack

3.1 Core Technology: Deep face and MTCNN

- **DeepFace:** A deep learning-based face recognition library that supports multiple pre-trained models for accurate facial recognition and classification.
- **MTCNN:** A popular open-source library for real-time face detection and image processing. It includes tools for image enhancement, edge detection, and feature extraction.And we will use it to run DeepFace.

3.2 Software Components

3.2.1 Backend

- **Language:** Python 3.8+
- **Framework:** FastAPI- Chosen for its simplicity and flexibility for building the API.
- **Core Libraries:**
  - **OpenCV** : For image acquisition from the camera, image preprocessing

(grayscale conversion, resizing), and face detection (using Haar Cascades or MTCNN).
    - **Deepface:** For facial landmark detection and generating face embeddings (128-d vectors) using its deep metric learning model.
    - **VGG-Face**: A deep learning model based on the VGG architecture that converts facial images into distinctive embeddings, enabling effective face recognition and verification. Its primary purpose is to identify and verify faces once they are detected and extracted, commonly using models like MTCNN.
    - **MTCNN** : A deep learning based face detection model that efficiently detects and aligns faces in images and videos, improving recognition accuracy. Main purpose to detect faces in the group photo
    - **NumPy**: For numerical operations, especially on image arrays and embeddings.
- **Database ORM:** SQLAlchemy - For interacting with the PostgreSQL database.

3.2.2 Frontend

- **Language:** JavaScript
- **HTTP Client:** Fetch API - For making requests to the backend API.
- **UI Components :** CSS.

3.2.3 Database

- **Database System:** PostgreSQL - Chosen for its robustness, reliability, and support for various data types.

Chapter 4: System Design and Architecture

4.1 System Architecture Overview

*The system operates on a local-first design. The user captures classroom images and upload them directly by the teacher dashboard , which processes and identifies students on-device. The recognized data is then verified with the **local database**. Attendance records are securely stored and made accessible via the **web dashboard**, with a **security layer** ensuring safe and role-based access to all modules.*

4.2 Component Description

1. ***Facial Recognition Module:***
    - *Uses AI algorithms to detect and recognize student faces.*
    - *Compares detected faces with pre-registered profiles in the database.*
    - *Extract classroom image from cloud and with the help of the unique course*

id
attached with the photo it finds all enrolled students .

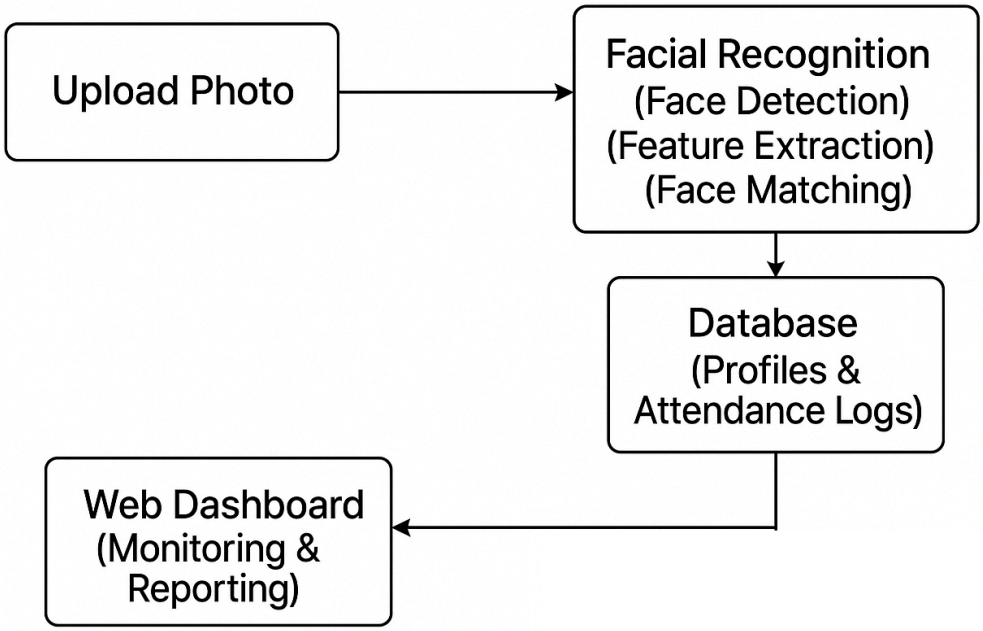
2. ***Database Module:***
    - *Stores student profiles, attendance logs, and administrator data.*
    - *Ensures data integrity and secure access.*
3. ***Web Dashboard:***
    - *Offers a graphical user interface (GUI) for students, teachers and administrators.*
    - *Displays real-time attendance record.*
4. ***Security Layer:***
    - *Implements data encryption, secure APIs, and role-based access control.*
    - *Ensures compliance with data protection regulations.*
    - *Password hashing to protect uses detail*

4.3 Data Flow Diagram

Upload Photo → Facial Recognition Module →

Database → Web Dashboard

4.4 UML Diagrams



4.6 Security Considerations

- ***Authentication & Authorization:****Secure API endpoints*
- ***API Security:*** *Secure inter-service communications using HTTPS .*

Chapter 5: Implementation

5.1 Development Environment

AI Model:
The facial recognition model is deployed as a hosted API on **Hugging Face Spaces**. This allows secure, scalable, and public access to the face embedding and verification logic without requiring on-device processing.

Backend:
The backend server is developed using **FastAPI** and deployed on **Render**. It acts as the central API gateway, handling all tasks .

Database:
A **PostgreSQL(Neon Console)** database is used to store student , teacher and course detail , and attendance logs.

Frontend:
The web-based frontend is deployed on **Netlify**. It provides the user interface for admins and faculty to upload attendance images, check attendance records, and manage student data.

Version Control:
The entire project is managed using **Git**, with source code hosted on **GitHub** for collaborative development and continuous deployment integration with Netlify and Render.

5.2 Backend Implementation

5.2.1 User Registration

Since all the students are registered at the starting of a year , and no new student is added , the admin can upload a .csv file to register all students .

5.2.2 Face Encoding and Storage

We use the VGG-Face model from the DeepFace library to extract 2622-dimensional facial embeddings from user registration images. These embeddings uniquely represent facial features and are used for later face recognition or verification. All these embeddings are stored with the model .

5.2.3 API Development

Endpoint	HTTP Method	Request body	Return value type	Model

/attendance/add/{course_id}	POST	Upload a image	Response model and 200 HTTP	Class {
				“present_students”:List[int]
				}

/login/student	POST	LoginBody	HTTP response and token	LoginBody Class{
				"password":String(not null)
				"email":String(not null)
				}

/login/teacher	POST	LoginBody	HTTP response and token	
/student/{student_id}	GET		StudentDashboard Class	StudentDashboard Class{ “student_name”:String “student_email”:String “course_id”:List[String “course_name”:List[String] “total_classes”:List[Int] “present_classes”:List[Int]

Endpoint	HTTP Method	Request body	Return value type	Model

/student/{student_id}/{course_id}	GET		List[StudentAttendance]
			List[Class]
				StudentAttendance Class{
				“date”:Date
				“status”:bool
				}

/teacher/{teacher_id}}	GET		TeacherDashboard class	TeacherDashboard class{
				“faculty_name”:String
				“faculty_email”:String
				“course_id”:List[Int]
				“courses_name”:List[String]
				}

/teacher/{course_id}	GET		CourseAttendance class	CourseAttendance Class{
				“total_student”:int
				“present”:List[int] (last 7 entry)
				“date”:List[date] (last 7 date)
				}

/teacher/{course_id}/{date}	GET		List[CourseStudentDetail]
			List[class]
				CourseStudentDetail class{
				“student_id”:int
				“student_name”:String
				“present”:bool
				“total_attedance”:float
				}

/teacher/{course_id}/student/{student_id}	GET		Detail Class	Detail Class{
				“student_name”:String
				“date”:List[date]
				“present”:List[bool]
				“total_attendance”:float
				}

/forgot-password/student	POST	Email class	HTTP 404 email doesn’t exist
			HTTP 200 opt sent PLUS id
				Email Class {
				email:str
				}

/forgot-password/faculty	POST	Email class	HTTP 404 email doesn’t exist
			HTTP 200 opt sent PLUS id
				Email Class {
				email:str
				}

/verify-otp	POST	Verify Class	HTTP 401 , otp expired
			HTTP 400 invalid otp
			HTTP 200 opt success
				Verify Class{
				“id”:int
				“otp”:str
				}

/reset-password	POST	ResetPassword Class	HTTP 400 illegal access
			HTTP 200 Password reset
				Class {
				“id”:int
				“otp”:str
				“new_password”:str
				}

5.3 Frontend Implementation

5.3.1 Teacher Dashboard

*The Teacher dashboard is built using **HTML, CSS, and JavaScript**, featuring a structured layout with navigation menus and summary cards. The interface displays real-time attendance summaries,and upload image options.*

5.3.2 Student Dashboard

*Implemented using **HTML** , **CSS** , and **JavaScript** for interactivity. The interface displays real-time attendance summaries for students.*

5.3.3 Login Page and Forgot Password

5.4 Key Algorithms (Face Detection & Recognition)

- ***Face Detection:** Implemented using **MTCNN** for accurate multi-face detection.*
- ***Face Recognition:** Facial embeddings are generated using **VGG-Face**.*
- ***Matching:KD Tree** is used to find the closest index at the shortest amount of time , with a set threshold (e.g., 0.6) to confirm matches.*
- *The entire process is performed on **Cloud***

Chapter 6: Testing and Evaluation

6.1 Testing Strategy

- ***Unit Tests**: Validated individual backend functions such as face embedding generation, database operations, and matching logic.*

- ***Integration Tests**: Verified communication between API endpoints and frontend, including image uploads and registration workflows.*

- ***System Tests**: Covered complete user flows — from uploading student data to marking attendance via live camera input.*

- ***Performance Tests**: Measured accuracy and speed under different lighting conditions and device capabilities.*

6.2 Results and Analysis

All components of the system functioned as expected.

- ***Accuracy**: High in well-lit conditions, with minor drops in low light, and also loss accuracy as distance between student and camera increases .*

- ***Speed**: Recognition time remained within acceptable real-time limits within 30 seconds .*

- ***Stability**: No major bugs found during testing; all user flows completed successfully.*
  The system is ready for deployment and real-world classroom usage.

Chapter 7: Deployment

7.1 Deployment Process and Steps

- ***Ai model** : Deployed the VGG-Face model on Hugging Face Spaces as an API for fast, GPU-accelerated face embedding generation.*
- ***Backend**: Deployed using **FastAPI** on **Render**, which handles API requests and database communication.*
- ***Frontend**: Built with **js** and deployed on **Netlify** for fast and secure UI access.*
- ***Database**: A managed **PostgreSQL** instance hosted on **Neon Console** for storing user data and attendance logs.*
- *Auto-deployment is configured through GitHub integration with Render and Netlify.*

7.4 Monitoring

- ***Monitoring** is done using Uptime Robot to receive a notification as soon as there is a error in website*

Chapter 8: Challenges and Solutions

8.1 Technical Challenges

- **8.1.1 Accuracy Variation:** Recognition accuracy fluctuated significantly with changes in ambient lighting and user pose.

- **8.1.2 Real-time Processing Performance:** Initial implementation struggled to process frames in real-time, leading to lag.

Chapter 9: Future Enhancements

9.1 Algorithm Improvements

- Improve handling of occlusions (masks, glasses) and extreme poses using more robust models or techniques.
- Explore alternative or newer face recognition models for potentially higher accuracy or speed.

9.2 Scalability and Performance Improvements

- Explore hardware acceleration (GPU) for face recognition processing if needed for higher throughput.

Chapter 10: Conclusion

10.1 Summary of Achievements

This project successfully designed, developed, and tested an Automated Attendance System based on facial recognition. The core objectives were met, resulting in a functional prototype capable of registering users, identifying them via a camera feed, and logging attendance automatically. The system provides a web-based interface for administration and reporting, demonstrating a significant improvement over manual methods. [Hypothetical: The final system achieved a promising recognition accuracy of 95% under optimal conditions.]

10.2 Final Remarks

While challenges related to environmental variables and performance exist, the developed system proves the viability of using facial recognition for automated attendance. The modular design allows for future improvements and integration. This project provides valuable experience in integrating AI/ML models into full-stack web applications and addresses real-world requirements for efficient and reliable attendance tracking.