

Ministry of Higher Education and Scientific Research  
Higher Institute of Engineering in El-Shorouk City  
Department of Communications and Computer Engineering



# College Smart Attendance & Management System

Submitted in partial fulfillment for the requirements for the degree of Bachelor's  
in Electronics and Communication \ Computer and Control Engineering

## Project Team

Mohamed Amr Madkour	Mohamed Salah Eldin Abdel Qader
Mohamed Abdel Baky Elsayed	Omar Mohamed Elsayed El-Tony
Mariam Ahmed Abdel Hady Salman	Weam Osama Mohamed Elsayed

## Supervisors

### Dr. Sahar Kamal

Communication and Computer Engineering Dept., Higher Institute of Engineering, Elshorouk  
Academy

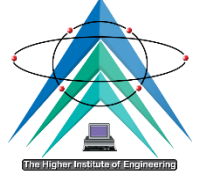
### Eng. Amira Essayed

Communication and Computer Engineering Dept., Higher Institute of Engineering, Elshorouk  
Academy

Cairo 2025



وزارة التعليم العالي والبحث العلمي  
المعهد العالي للهندسة بمدينة الشروق  
قسم هندسة الاتصالات والحاسبات



## نظام الحضور والإدارة الذكي للكلية

مقدمة كجزء من متطلبات الحصول علي درجة البكالوريوس في هندسة الإلكترونيات والاتصالات / الحاسبات والتحكم

### فريق المشروع

محمد صلاح الدين عبد القادر

محمد عمرو مذكور

عمر محمد السيد التونسي

محمد عبد الباقي السيد

ونائم أسامة محمد السيد

مريم أحمد عبد الهادي سلمان

### المشرفون

د. سحر كمال

قسم هندسة الاتصالات والحاسبات – المعهد العالي للهندسة – أكاديمية الشروق

م. أميرة السيد

قسم هندسة الاتصالات والحاسبات – المعهد العالي للهندسة – أكاديمية الشروق

القاهرة 2025

# Acknowledgement

To accomplish even the simplest task, the help of Allah Almighty is required and our project, with its complexities and difficulties, was not an easy task. After completing the task of its production, we are thankful to Allah the most Powerful that he helped us at every step and never let the courage waver even in the face of the worst problem.

We would love to thank our Project supervisors, Dr. Sahar Kamal and Eng. Amira Elsayed for their tremendous support and help throughout, even though it took up a lot of their precious time.

We wish to acknowledge all our university teachers who support and guide us throughout this degree. We are also thankful to our families who always support us in every single step and will always be there for us.

# Table of content

ACKNOWLEDGEMENT .....	I
ABSTRACT .....	VIII
<b>Chapter 1 : INTRODUCTION .....</b>	<b>1</b>
1.1 INTRODUCTION.....	1
1.2 BACKGROUND .....	1
1.3 PROBLEM STATEMENT.....	2
1.4 EVOLUTION OF ATTENDANCE SYSTEM .....	3
1.5 OBJECTIVES .....	5
1.6 SYSTEM FLOW CHART .....	5
<b>CHAPTER 2 : ARTIFICIAL INTELLIGENCE INTRODUCTION .....</b>	<b>6</b>
2.1 ARTIFICIAL INTELLIGENCE DEFINITION.....	6
2.2 HOW AI WORKS .....	6
2.3 COMPUTER VISION.....	7
2.4 IMAGE PROCESSING .....	7
2.5 MACHINE LEARNING.....	8
2.6 NEURAL NETWORKS.....	8
2.6 DEEP LEARNING .....	9
2.7 DEEP LEARNING VS. MACHINE LEARNING .....	9
2.8 FLOW CHART OF AI MODEL.....	9
<b>CHAPTER 3 : FACE DETECTION .....</b>	<b>10</b>
3.1 DEFINITION OF FACE DETECTION .....	10
3.2 HISTORY OF FACE DETECTION .....	10
3.3 APPLICATIONS FOR FACE DETECTION .....	10
3.4 TECHNIQUES USED IN FACE DETECTION .....	11
3.4.1 <i>Traditional methods</i> .....	11
3.4.2 <i>Neural network-based methods</i> .....	13
3.5 YOLO VERSIONS .....	17
3.6 CHOOSE THE BEST VERSION FOR FACE DETECTION .....	17
3.6.1 <i>YOLOv8n-Face-Detection</i> .....	18
3.6.2 <i>YOLOv11n-Face-Detection</i> .....	18
3.7 YOLOV11 IN DETAIL .....	20
3.7.1 <i>Overview</i> .....	20
3.7.2 <i>Key Features</i> .....	20
3.7.3 <i>Tasks can YOLO11 models perform</i> .....	20
3.7.4 <i>Detailed Explanation of the YOLOv11 Architecture</i> .....	21
3.8 TESTING RESULTS .....	23
<b>CHAPTER 4 : FACE RECOGNITION .....</b>	<b>24</b>
4.1 DEFINITION OF FACE RECOGNITION .....	24
4.2 HISTORY OF FACE RECOGNITION.....	24
4.3 APPLICATIONS FOR FACE RECOGNITION .....	24
4.4 TECHNIQUES USED IN FACE RECOGNITION .....	25
4.4.1 <i>Traditional Methods</i> .....	25

4.4.1.1 State-of-the-Art Models for Face Recognition.....	28
4.5 FACE NET WITH ARCHITECTURE INCEPTION-REST NET v1 .....	31
4.5.1 Definition.....	31
4.5.2 FaceNet with Inception-ResNet v1 Architecture Breakdown .....	32
4.6 DATASET COLLECTION AND CREATION OF EMBEDDINGS FOR EACH STUDENT .....	33
4.6.2 Collect about 20-30 images for each student and save images for each student in a folder named with student ID .....	33
4.6.3 Split images into 80% for training and 20% for testing.....	33
4.6.5 Preprocess faces for FaceNet .....	34
4.6.6 Generate Embedding .....	34
4.7 REAL TIME FACE RECOGNITION .....	35
4.7.1 Detect Faces from input image .....	35
4.7.2 Preprocessing .....	35
4.7.3 Embedding generation.....	36
4.7.4 Recognition .....	36
4.7.5 Results .....	36
4.8 LIMITATIONS .....	37
<b>CHAPTER 5 : SYSTEM BACK-END.....</b>	<b>38</b>
5.1 SYSTEM BACK-END DEVELOPMENT .....	38
5.1.1 Definition of Backend Development .....	38
5.1.2 Frameworks Used in the Backend.....	38
5.1.3 Backend Responsibilities.....	38
5.1.4 Technologies Used in the Backend .....	39
5.2 SYSTEM MODELING – BACKEND .....	40
5.2.1 MVC Architecture in Django.....	40
5.2.2 Class Diagram .....	41
5.2.3 Sequence Diagram .....	41
5.3 DATABASE: DESIGN AND TECHNOLOGIES.....	42
5.4 RESTFUL API DESIGN .....	44
5.4.1 Structure and Organization .....	44
5.4.2 HTTP Methods and Status Codes.....	45
5.4.3 Serialization and Validation .....	46
5.5 SECURITY IN THE BACKEND .....	46
5.5.1 Authentication with JWT .....	46
5.5.2 Role-Based Access Control.....	46
5.5.3 Input Validation and Error Handling .....	47
5.6 DOCUMENTATION AND MAINTAINABILITY .....	47
5.6.1 Inline Documentation and Code Readability .....	47
5.6.2 API Documentation.....	48
5.6.3 Folder Structure and Naming Conventions .....	48
5.6.4 Reusability and Modularity .....	48
5.6.5 Developer Guidelines and README Files.....	49
5.7 AI INTEGRATION FROM A BACKEND PERSPECTIVE .....	49
5.7.1 The Role of the Backend in AI Interaction .....	49
5.7.2 AI Processing Workflow.....	50
5.7.3 Error Handling and Fallbacks .....	50
5.7.4 Real-Time Operation and Performance .....	51
5.8 TESTING, DEBUGGING, AND LOGGING .....	51

5.8.1 Unit Testing and Integration Testing .....	51
5.8.2 Debugging Tools .....	52
5.8.3 Backend Logging .....	52
5.8.4 Exception Handling and Error Reporting .....	52
CI/CD and Automated Testing .....	53
5.9 DEPLOYMENT AND SCALING CONSIDERATIONS .....	53
5.9.1 Deployment Environment .....	53
5.9.2 WSGI & ASGI Gateways.....	54
5.9.3 Database Configuration.....	54
5.9.4 Performance Optimization .....	54
5.9.5 Horizontal Scaling .....	55
5.9.6 Logging and Monitoring in Production .....	55
5.10 SUMMARY OF BACKEND RESPONSIBILITIES.....	55
<b>CHAPTER 6 : SYSTEM MODELING AND FRONT-END .....</b>	<b>57</b>
6.1 THE CONTEXT DIAGRAM .....	57
6.2 MOBILE APPLICATION BUSINESS PROCESS (STUDENT SIDE) .....	58
6.2.1 Login and Authentication .....	58
6.2.2 Attendance Overview .....	58
6.2.3 Notifications .....	58
6.2.4 Absence Request Submission .....	58
6.2.5 Profile and Settings .....	59
6.3 WEB APPLICATION BUSINESS PROCESS (ADMIN/INSTRUCTOR SIDE).....	59
6.3.1 Admin Dashboard .....	59
6.3.2 Instructor Dashboard .....	59
6.3.3 Session Scheduling .....	60
6.3.4 Attendance Management.....	60
6.3.5 Absence Request Review .....	60
6.4 BUSINESS RULES .....	60
6.4.1 Authentication & Authorization Rules .....	60
6.4.2 Student Attendance Rules .....	61
6.4.3 Session & Schedule Rules .....	61
6.4.4 Absence & Request Rules.....	62
6.4.5 Notification Rules .....	62
6.4.6 Reporting Rules .....	62
6.5 ACTIVITY DIAGRAM .....	63
6.6 USE CASE DIAGRAMS .....	65
6.7 FRONTEND .....	67
6.7.1 Definition.....	67
6.7.2 Purpose and Importance of Frontend.....	68
6.7.3 Why Use a Frontend in This Project?.....	68
6.8 MOBILE FRONTEND USING FLUTTER .....	69
6.8.1 What is Flutter? .....	69
6.8.2 Why Flutter Was Chosen.....	69
6.9 WHAT IS WEB FRONTEND? .....	70
6.10 FIGMA DESIGN AND IMPORTANCE OF UI/UX .....	72
6.10.1 Web Application (Admin/Instructor Role) Design.....	72
6.10.2 Mobile Application (Student Role).....	76
6.11 IMPLEMENTATION OF UI/UX DESIGN (MOBILE + WEB).....	79

6.11.2 Why This Implementation Matters.....	80
<b>CHAPTER 7 : SYSTEM RESULTS.....</b>	<b>81</b>
7.1 CAMERA ACTIVATION AND FRAME CAPTURE .....	81
7.3 FACE RECOGNITION USING FACENET EMBEDDING.....	82
7.4 SESSION MATCHING (DATE, TIME & SUBJECT VALIDATION).....	82
7.5 ATTENDANCE RECORDING IN THE DATABASE.....	83
7.6 FRONTEND DISPLAY ON MOBILE AND WEB .....	83
CONCLUSION .....	85
FUTURE WORK .....	85
REFERENCES .....	7-1
شكر وتقدير.....	

# LIST OF FIGURES

FIGURE 1-1 MANUAL HANDWRITTEN SHEET .....	3
FIGURE 1-2 BIOMETRIC ATTENDANCE SYSTEMS .....	3
FIGURE 1-3 QR CODE ATTENDANCE SYSTEM .....	4
FIGURE 1-4 FACE RECOGNITION ATTENDANCE SYSTEM .....	4
FIGURE 1-5 SYSTEM FLOW CHART.....	5
FIGURE 2-1 THE EVOLUTION OF AI .....	6
FIGURE 2-2 HUMAN VISION VS COMPUTER VISION .....	7
FIGURE 2-3 SIMPLE CNN .....	8
FIGURE 2-4 ACTUAL CNN .....	8
FIGURE 2-5 MACHINE LEARNING VS DEEP LEARNING .....	9
FIGURE 2-6 FLOW CHART OF AI MODEL .....	9
FIGURE 3-1 SELECTING HAAR-LIKE FEATURS.....	11
FIGURE 3-2 INTEGRAL IMAGE .....	11
FIGURE 3-3 CLASSIFIER CASCADES .....	12
FIGURE 3-4 DEVIDE IMAGE INTO SMALL GRADE .....	12
FIGURE 3-5 HISTOGRAM FOR EACH CELL.....	13
FIGURE 3-6 HISTOGRAM FOR THE IMAGE .....	13
FIGURE 3-7 CNN LAYERS .....	14
FIGURE 3-8 MTCNN ARCHITECTURE.....	15
FIGURE 3-9 HOW YOLO WORKS .....	17
FIGURE 3-10 YOLO VERSIONS .....	17
FIGURE 3-11 YOLOV8 RESULTS .....	18
FIGURE 3-12 YOLOV8 CONFUSION MATRIX.....	18
FIGURE 3-13 YOLOV11 RESULTS .....	19
FIGURE 3-14 YOLOV11 CONFUSION MATRIX.....	19
FIGURE 3-15 YOLOV11 ARCHITECTURE.....	21
FIGURE 3-16 TESTING IMAGE .....	23
FIGURE 3-17 DETECTED FACES.....	23
FIGURE 4-1 EIGENFACES WORKING STRUCTURE .....	25
FIGURE 4-2 FISHERFACES WORKING STRUCTURE .....	26
FIGURE 4-3 LBPH WORKING STRUCTURE .....	27
FIGURE 4-4 DLIB WORKING STRUCTURE .....	27
FIGURE 4-5 FACENET STRUCTURE .....	28
FIGURE 4-6 DEEPFACE STRUCTURE.....	29
FIGURE 4-7 ARCFACE STRUCTURE .....	30
FIGURE 4-8 VGGFACE STRUCTURE .....	31
FIGURE 4-9 FACENET WITH INCEPTION-RESNET V1 ARCHITECTURE .....	32
FIGURE 4-10 DATASET.....	33
FIGURE 4-11 SPLIT DATASET .....	33
FIGURE 4-12 EXTRACT FACES AND RESIZE THEM.....	34
FIGURE 4-13 DETECTION OF FACES .....	35
FIGURE 4-14 TESTING IMAGE .....	36
FIGURE 4-15 OUTPUT .....	37
FIGURE 5-1 BACKEND CLASS DIAGRAM .....	41
FIGURE 5-2 BACKEND SEQUENCE DIAGRAM .....	42
FIGURE 5-3 DATABASE SCHEMA .....	44



FIGURE 6-1 CONTEXT DIAGRAM .....	57
FIGURE 6-2 MOBILE APP ACTIVITY DIAGRAM .....	64
FIGURE 6-3 WEB APP ACTIVITY DIAGRAM .....	65
FIGURE 6-4 FLUTTER APPLICATION USE CASE DIAGRAM .....	66
FIGURE 6-5 WEB APPLICATION USE CASE DIAGRAM .....	67
FIGURE 6-6 FLUTTER .....	70
FIGURE 6-7 HTML + CSS .....	71
FIGURE 6-8 LOGIN PAGE .....	72
FIGURE 6-9 FORGET PASSWORD PAGE .....	73
FIGURE 6-10 PASSWORD RESET PAGE .....	73
FIGURE 6-11 DASHBOARD PAGE .....	73
FIGURE 6-12 SUBJECTS PAGE .....	74
FIGURE 6-13 PROFILE PAGE .....	74
FIGURE 6-14 STUDENT DETAILS PAGE .....	74
FIGURE 6-15 MESSAGE PAGE .....	75
FIGURE 6-16 STUDENTS MONTHLY REPORT .....	75
FIGURE 6-17 LEAVE APP .....	75
FIGURE 6-18 MOBILE APP LOGIN PAGE .....	76
FIGURE 6-19 MOBILE APP FORGETS PASSWORD PAGE .....	76
FIGURE 6-20 MOBILE APP HOME PAGE .....	77
FIGURE 6-21 MOBILE APP ATTENDANCE REPORT PAGE .....	77
FIGURE 6-22 MOBILE APP NOTIFICATION PAGE .....	78
FIGURE 6-23 MOBILE APP ACCOUNT MANAGEMENT PAGE .....	78
FIGURE 6-24 MOBILE APP REQUEST ABSENCE .....	79
FIGURE 7-1 CAMERA CAPTURES IMAGE .....	81
FIGURE 7-2 LOAD STUDENTS CODE .....	82
FIGURE 7-3 MATCHES SESSION DATE AND TIME AND RECOGNIZE STUDENTS .....	82
FIGURE 7-4 DATABASE CHANGE IN THE STATUS .....	83
FIGURE 7-5 INSTRUCTOR WEB APP DISPLAY .....	84
FIGURE 7-6 STUDENT MOBILE APP DISPLAY .....	84

# Abstract

This project presents an AI-powered Face Recognition Attendance System designed to automate and streamline student attendance in academic institutions. It tackles the limitations of manual methods by combining YOLOv11 for real-time face detection and FaceNet with InceptionResNetV1 for high-accuracy face recognition. The system captures student faces through an IP camera, compares them to stored embeddings, and logs attendance automatically. A secure backend manages registration, course schedules, and attendance records. The system also supports leave requests and generates insightful reports. A mobile app enables student interaction, while a web dashboard empowers instructors to manage and monitor data. This end-to-end solution improves accuracy, prevents proxy attendance, and ensures operational efficiency. Its modular design supports scalability and ease of integration in educational environments.

# Chapter 1 : Introduction

## 1.1 Introduction

In modern education, tracking attendance is crucial for academic performance and accountability. Traditional methods like manual sign-ins or roll calls are inefficient, error-prone, and susceptible to manipulation—such as proxy attendance.

As institutions grow, there's a rising need for smart, automated solutions. AI and deep learning offer reliable tools for this, with face recognition emerging as a powerful non-intrusive method for real-time identity verification and attendance tracking.

Despite the availability of such technology, many schools—especially in the Arab region—still use outdated methods due to limited resources or lack of awareness. Our project aims to close this gap by developing a practical and scalable AI-powered attendance system.

We use YOLOv11 for fast face detection and FaceNet with InceptionResNetV1 for accurate recognition. An IP camera captures student faces as they enter the classroom. If the system finds a match within a threshold, attendance is marked automatically—no manual effort needed.

To ensure usability, we built a mobile app for students to view attendance and request leaves, and a web dashboard for instructors to manage sessions and reports. Data is stored securely and organized for easy access.

This system enhances accuracy, saves time, and reduces administrative load. It also minimizes fake attendance and gives institutions insights into attendance trends and student engagement.

Ultimately, our goal is to modernize attendance using AI—making it seamless, secure, and efficient—while laying the foundation for future smart classroom features like emotion detection and participation analysis.

## 1.2 Background

According to UNESCO reports and global education surveys, maintaining accurate student attendance is a major challenge faced by educational institutions worldwide. Attendance is not only linked to academic performance but also to government funding, institutional ranking, and student discipline. Traditional methods of attendance tracking.

As education systems grow in scale, the need for smart, automated attendance solutions becomes more urgent.

Our project aims to fill this gap by implementing a robust face recognition-based attendance system tailored for classroom use. It combines state-of-the-art AI models with a user-friendly interface, real-time detection, and a secure backend. By doing so, we provide an efficient, scalable, and practical solution to replace traditional attendance systems and ensure fair and transparent record-keeping.

### **1.3 Problem Statement**

Students and educational institutions continue to rely on outdated and inefficient methods for recording attendance, such as manual roll calls or paper-based registers. These traditional approaches are time-consuming, prone to human error, and highly susceptible to manipulation, including proxy attendance. Such issues not only reduce academic accountability but also distort student performance metrics and institutional reporting.

While some institutions have attempted to adopt semi-automated systems like RFID cards or fingerprint scanners, these methods still require physical interaction, are vulnerable to system failure or misuse, and often involve costly hardware installations. Moreover, these solutions do not offer real-time monitoring or integration with other academic systems like leave tracking or session validation.

Despite the growing availability of AI-powered tools globally, many educational institutions in the Arab region have yet to benefit from modern attendance systems. This is due to a combination of technical, financial, and infrastructural challenges. There remains a lack of localized, scalable, and affordable frameworks tailored to the needs of Arabic educational environments.

Thus, there is a pressing need for a fully automated, intelligent attendance system that can operate reliably using minimal hardware, support real-time recognition, and seamlessly integrate with academic schedules and databases—while being accessible and practical for regional institutions.

## 1.4 Evolution of Attendance System

### 1. Manual handwritten Sheet:

Surprisingly, most universities still use this method to track student attendance. Yet most modern companies don't use it anymore because it's inconvenient and requires having a responsible person for filling it in.



*Figure 1-1 manual handwritten sheet*

### 2. Biometric attendance systems:

Biometric attendance systems have been used for security purposes since 1996 and was first installed at the Olympic Village in Atlanta. Biometric systems are still used for security purposes and attendance tracking because it is impossible to copy a finger or handprint. But for students it will cause congestion due to the large number of students and the device may malfunction.



*Figure 1-2 biometric attendance systems*

### 3. QR code attendance system:

The QR system is currently used as a system for recording absences at Al Shorouk Academy and some other universities, but testing this system has proven that absences can be manipulated.

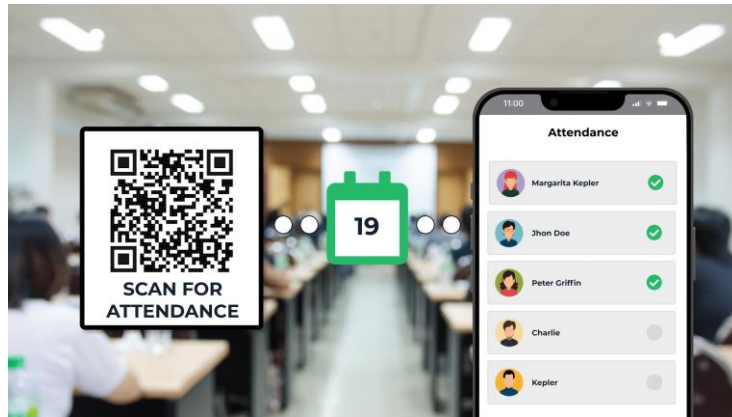


Figure 1-3 QR code attendance system

### 3. Face attendance system:

Face recognition is an Attendance system recognizing the person by using face biostatistics is aimed to accomplish digitization of the traditional system of taking attendance, Present strategies for taking attendance are tedious and time-consuming.

The traditional process of making attendance and present biometric systems are vulnerable to proxies.



Figure 1-4 face recognition attendance system

## 1.5 Objectives

This project aims to design and implement a smart and automated attendance system based on facial recognition technology to replace traditional and semi-automated attendance methods in educational institutions and also enable students and instructors to track attendance.

Expected achievements in order to fulfill the objectives are:

- To detect the face segment from the video frame.
- To extract the useful features from the face detected.
- To classify the features in order to recognize the face detected.
- To record the attendance of the identified student.
- Display student absence statistics for students and instructors.

In the next chapters we will show the theoretical background of the main technologies we use in this project.

## 1.6 System Flow chart

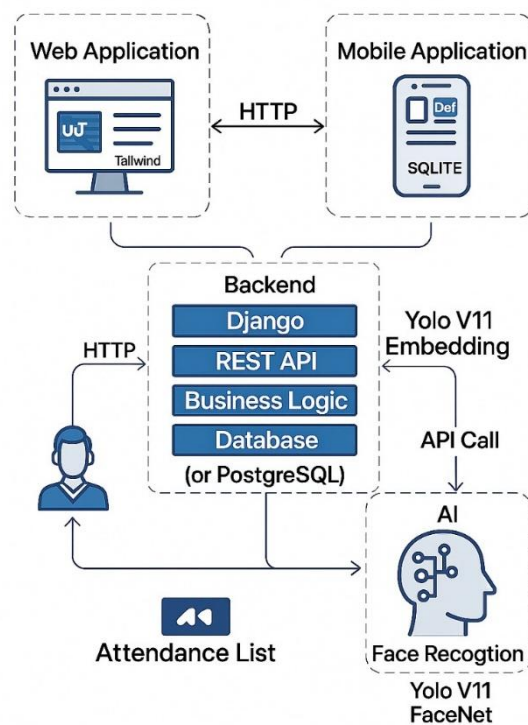


Figure 1-5 system flow chart

# Chapter 2 : Artificial Intelligence Introduction

Artificial Intelligence (AI) has evolved from theoretical concepts to practical applications, including computer vision tasks like face detection and recognition. Modern AI leverages deep learning to achieve high accuracy in these tasks.

## 2.1 Artificial Intelligence Definition

AI enables machines to perform tasks requiring human-like intelligence, such as reasoning, learning, and decision-making. In face detection and recognition, AI processes visual data to identify and verify faces.

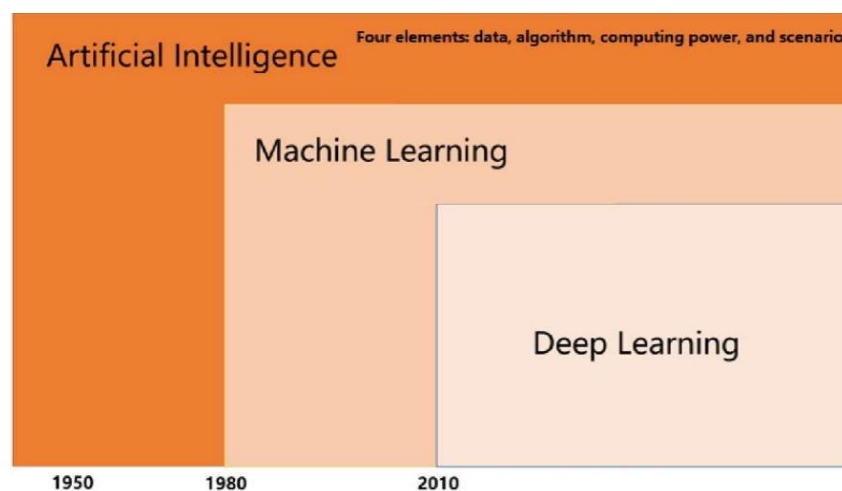


Figure 2-1 The Evolution of AI

## 2.2 How AI Works

AI systems for face detection and recognition:

- **Learn:** from labeled datasets (e.g., images with annotated faces).
- **Reason:** by selecting optimal algorithms (e.g., YOLOv11 for detection, FaceNet for recognition).
- **Self-correct:** through iterative training to improve accuracy.
- **Create:** embeddings (e.g., FaceNet's 128-D vectors) to represent faces uniquely.



## 2.3 Computer Vision

- **Definition:** Extracts meaningful information from visual inputs (images/videos).

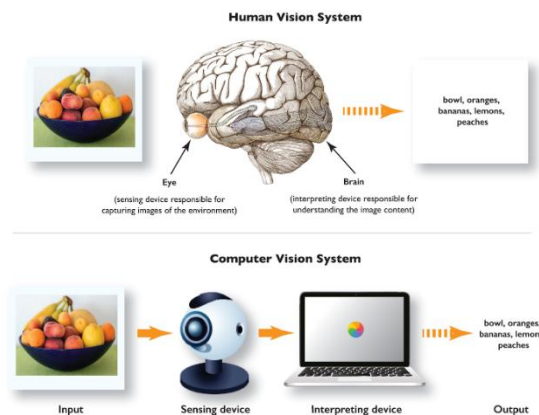


Figure 2-2 human vision vs computer vision

- **Key Tasks:**
  - **Face Detection:** Locates faces in images (e.g., YOLOv11 predicts bounding boxes).
  - **Face Recognition:** Identifies individuals (e.g., FaceNet maps faces to embeddings).
- **Technologies:**
  - **Deep Learning:** CNNs automatically learn features (edges, textures) from raw pixels.
  - **Convolutional Neural Networks (CNNs):** Process hierarchical features (layers detect edges → facial parts → full faces).

## 2.4 Image Processing

- **Steps:**
  1. **Acquisition:** Capture face images (e.g., via cameras).
  2. **Enhancement:** Improve contrast/resolution.
  3. **Segmentation:** Isolate faces from backgrounds.
  4. **Representation:** Convert faces to numerical embeddings (FaceNet).
- **Types:** Grayscale/RGB conversion, noise reduction, alignment.

## 2.5 Machine Learning

- **Supervised Learning:** Trains models on labeled face datasets (e.g., "Person A" vs. "Person B").
- **Algorithms:**
  - **YOLOv11:** Detects faces in real-time using bounding boxes.
  - **FaceNet:** Uses triplet loss to generate discriminative face embeddings.

## 2.6 Neural Networks

- **CNNs:** Dominant for face tasks:
  - **Convolutional Layers:** Detect local features (eyes, nose).
  - **Pooling Layers:** Reduce spatial dimensions.
  - **Fully Connected Layers:** Classify faces (FaceNet's inception-ResNetv1).

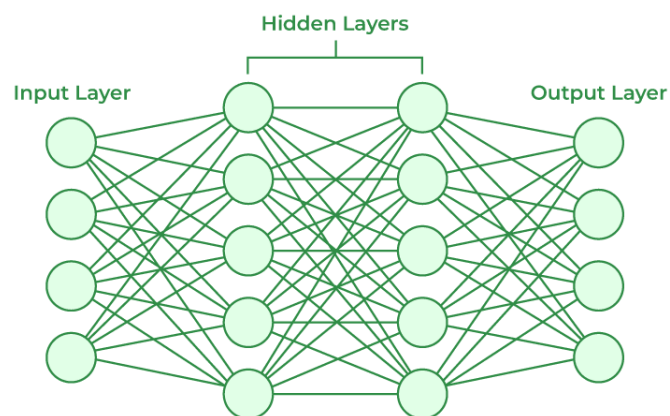


Figure 2-3 simple CNN

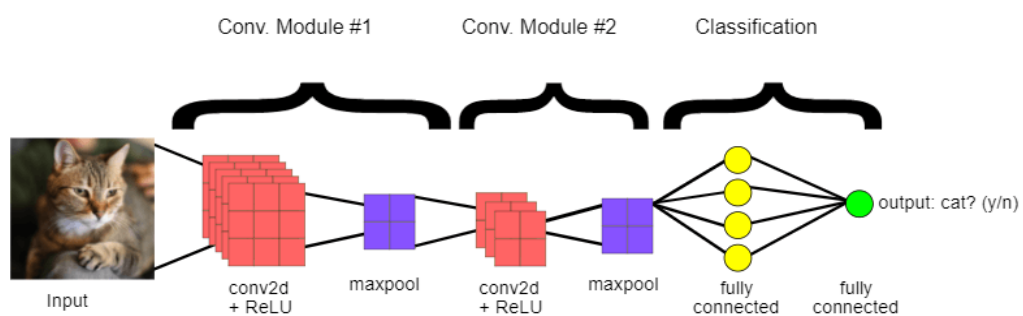


Figure 2-4 actual CNN

## 2.6 Deep Learning

- **FaceNet:** Combines inception-ResNetv1 architecture with triplet loss to map faces into a Euclidean space where distances correspond to similarity.
- **YOLOv11:** Optimized for speed/accuracy, using anchor boxes and non-max suppression for face detection.

## 2.7 Deep Learning vs. Machine Learning

- **Traditional ML:** Requires manual feature extraction (e.g., Haar cascades).
- **Deep Learning:** Automatically learns features (e.g., YOLOv11/FaceNet), scaling better with data.

### Key Technologies for Face Detection/Recognition

1. **YOLOv11:**
  - Real-time face detection via bounding boxes.
  - Backbone: CSPDarknet53 for feature extraction.
2. **FaceNet (inception-ResNetv1):**
  - Generates 128-D embeddings for face recognition.
  - Triplet loss ensures embeddings of the same person are closer in space.

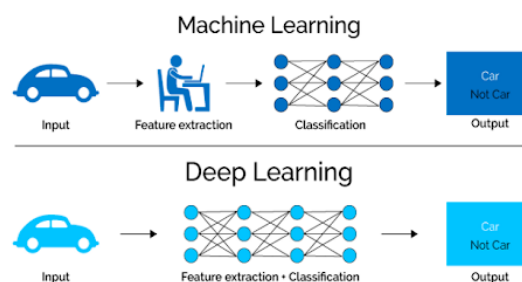


Figure 2-5 machine learning vs deep learning

## 2.8 Flow chart of AI model



Figure 2-6 flow chart of AI model

# Chapter 3 : Face Detection

## 3.1 Definition of Face Detection

Face detection is a sub-direction of object detection, and it is a computer technology that identifies and locates human faces in digital images or videos. It is a fundamental step in facial recognition systems, where the goal is to determine the presence, location, and sometimes the size of faces within an image or video frame.

## 3.2 History of Face Detection

- **1960s:** Early efforts involved manually marking facial features on images for computer comparison.
- **1980s-1990s:** Mathematical approaches like "Eigenfaces" (using PCA) significantly advanced automation, notably by Sirovich, Kirby, Turk, and Pentland.
- **2000s:** Face detection began to see practical applications in security and law enforcement, with commercial systems emerging.
- **2010s:** The rise of deep learning and AI, particularly Convolutional Neural Networks (CNNs), revolutionized accuracy and led to widespread consumer adoption (e.g., smartphone face unlock).
- **2020s:** Continued advancements focus on improved accuracy and robustness, alongside growing discussions about ethical implications and privacy.

## 3.3 Applications for Face Detection

**Face detection** is the cornerstone of all applications revolving around facial image analysis including, but not limited to, **face recognition and verification**, face tracking for surveillance, facial behavior analysis, facial attribute recognition (i.e., gender/age recognition and assessment of beauty), face relighting and morphing, facial shape reconstruction, image and video retrieval, as well as organization and presentation of digital photo-albums. Face detection is also the initial step to all modern vision-based human-computer and human-robot interaction systems (e.g., the recent commercial robots such as Nao come with an embedded face detection module). Furthermore, the majority of the commercial digital cameras have an embedded face detector that is used to help auto focusing. Finally, many social networks, such as Facebook, use face detection mechanisms for the purpose of image/person tagging.

## 3.4 Techniques Used in Face Detection

Face detection techniques have evolved over the years to achieve more accurate results.

### 3.4.1 Traditional methods

These are older computer vision techniques that rely on identifying and using distinct, informative patterns or points within an image. They were dominant before the rise of deep learning, proving effective in tasks like object recognition and image matching by creating stable representations of image content.

#### 3.4.1.1 Viola-Jones Algorithm

A classic algorithm (2001) for **real-time face detection**. Viola-Jones set the foundation for it in the field of facial detection.

#### How it works:

##### Step1: Selecting Haar-like features

Selecting Haar features. Haar features are digital image features.

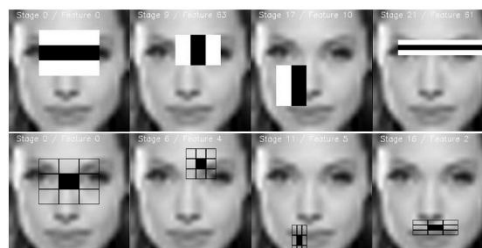


Figure 3-1 selecting haar-like features

##### Step2: Creating an integral image

An integral image gives a fast and simple way to calculate the value of any haar-like feature.

It is used to efficiently calculate the sum of dark/light rectangular regions in Viola-Jones.

0	1	1	1
1	2	2	3
1	2	1	1
1	3	1	0

0	1	2	3
1	4	7	11
2	7	11	16
3	11	16	21

Figure 3-2 integral image

### Step3: Running AdaBoost training

The AdaBoost, that is Adaptive Boosting algorithm, is a learn and predict ML model that identifies the best features. It does this through information on the training dataset

### Step4: Creating classifier cascades

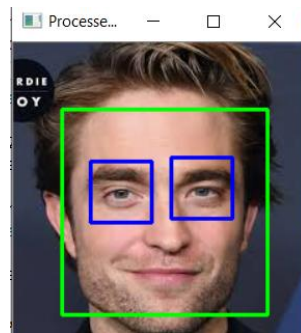


Figure 3-3 classifier cascades

### The disadvantages with Viola-Jones Algorithm

- **Slow Training:** Requires significant time and resources.
- **Sensitive to Lighting/Pose:** Accuracy drops in difficult conditions.
- **False Positives:** Can produce false positives in complex backgrounds.

#### 3.4.1.2 HOG (Histograms of Oriented Gradients)

A feature descriptor (2005) used for **object detection**, especially **pedestrian detection**. It's efficient at feature extraction and robust against lighting changes.

#### How it works:

**Step1:** The basic idea of HOG is dividing the image into small connected cells

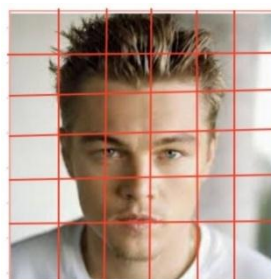


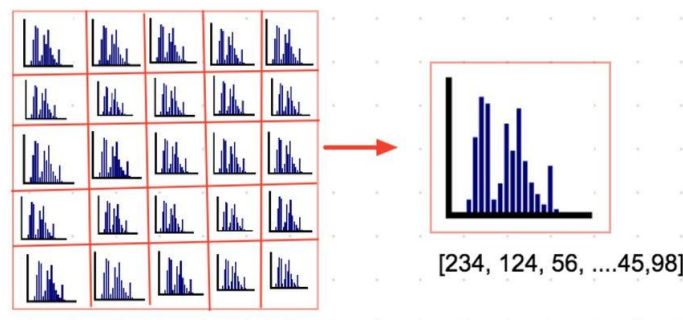
Figure 3-4 divide image into small grade

**Step2: Computes histogram for each cell.**



*Figure 3-5 histogram for each cell*

**Step3: Bring all histograms together to form feature vector i.e., it forms one histogram from all small histograms which are unique for each face**



*Figure 3-6 histogram for the image*

The disadvantage with HOG based face detection is that **it doesn't work on faces at odd angles**, it only works with straight and front faces. **It is really useful if you use it to detect faces from scanned documents** like driver's license and passport **but not a good fit for real-time video.**

### **3.4.2 Neural network-based methods**

This approach uses artificial neural networks to learn and perform computer vision tasks. It revolutionized the field by moving to **end-to-end learning from raw data**, achieving state-of-the-art performance across most computer vision applications.

### 3.4.2.1 Convolutional Neural Network (CNN)

An advanced version of **artificial neural networks (ANNs)**, primarily designed to extract features from grid-like matrix datasets. This is particularly useful for visual datasets such as images or videos, where data patterns play a crucial role.

#### How it works:

Convolutional Neural Networks (CNNs) are designed to process images using **shared filters** (kernels) that scan small regions (patches) of the input. Each filter learns to detect a specific feature (like edges or textures).

- **Convolution Layer:** Applies small filters (e.g. 3x3x3) across the input using a stride. Each filter creates a 2D feature map. Stacking multiple feature maps forms an output volume.
- **Activation Layer:** Adds non-linearity (e.g. ReLU) to the output, enabling complex pattern learning.
- **Pooling Layer:** Reduces spatial size (e.g. 2x2 max-pooling) to make computation efficient and avoid overfitting.
- **Flatten Layer:** Converts 3D feature maps into a 1D vector.
- **Fully Connected Layer:** Performs classification or regression based on learned features.
- **Output Layer:** Applies functions like softmax or sigmoid to output final probabilities.

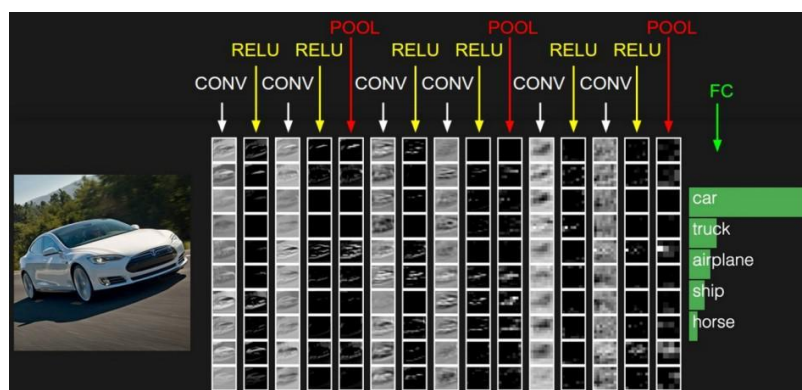


Figure 3-7 CNN layers



**The disadvantage** with Convolutional Neural Network (CNN):

- **High Computational Resources:** Deep CNNs still require significant power for training.
- **Requires Large Datasets.**
- **Information Loss (Pooling):** Pooling layers can lose fine-grained spatial detail, especially for small objects.

### 3.4.2.2 MTCNN (Multi-task Cascaded Convolutional Networks)

MTCNN is a neural network model specifically designed for face detection tasks. It detects faces at different scales and angles in a single pass.

#### How it works:

The MTCNN algorithm consists of three main stages, Here's how each of these stages works

1. **Proposal Network (P-Net):** The first stage of the MTCNN algorithm is the P-Net, which generates a set of candidate bounding boxes that may contain a face.
2. **Refinement Network (R-Net):** The second stage of the MTCNN algorithm is the R-Net, which refines the candidate bounding boxes generated by the P-Net.
3. **Output Network (O-Net):** The final stage of the MTCNN algorithm is the O-Net, which further refines the bounding boxes and extracts the facial landmarks.

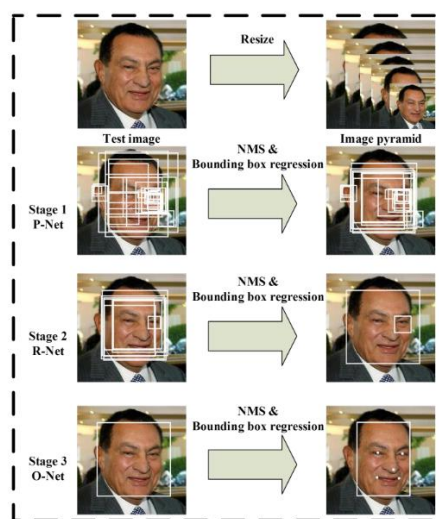


Figure 3-8 MTCNN architecture

### The disadvantage with MTCNN:

- Struggles with Extreme Angles or Occlusion
- Not Suitable for Tiny Faces
- Not End-to-End Trainable Easily
- Outdated Compared to Newer Models
- Hard to Scale for Large Datasets

#### 3.4.2.3 YOLO (You Only Look Once)

An object detection algorithm (2016) famous for its **exceptional speed**, processing the entire image in a single pass to predict bounding boxes and class probabilities. Ideal for real-time video analysis and robotics.

“YOLO is a state-of-the-art deep learning framework for real-time object detection.”

### How YOLO Works

1. Each Grid Predicts Bounding Boxes
  - For each cell in input image, YOLO predicts:
    - Box coordinates (x, y, width, height)
    - Confidence score (how sure it is there's an object)
    - Class probabilities (face, person, etc.)
    -
2. Face as a Class
  - When used for face detection, YOLO is trained to detect just one class: 'face'.
  - So instead of detecting many classes like dog, cat, car, it only focuses on faces.
3. Single Forward Pass
  - YOLO detects all faces in one go, without scanning the image multiple times.
  - This makes it super-fast — great for real-time applications.
4. Non-Max Suppression (NMS)
  - If multiple boxes overlap the same face, NMS removes duplicates, keeping the one with the highest confidence.

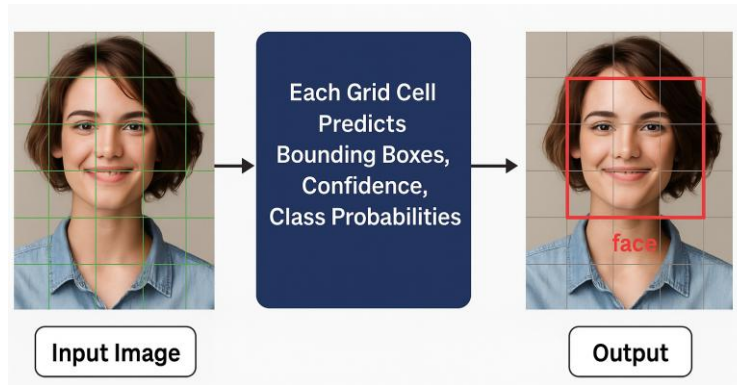


Figure 3-9 how YOLO works

This concludes our brief overview of the methods used for face detection. We will now delve more deeply into the specific algorithm chosen as the initial step for our project.

### 3.5 YOLO versions

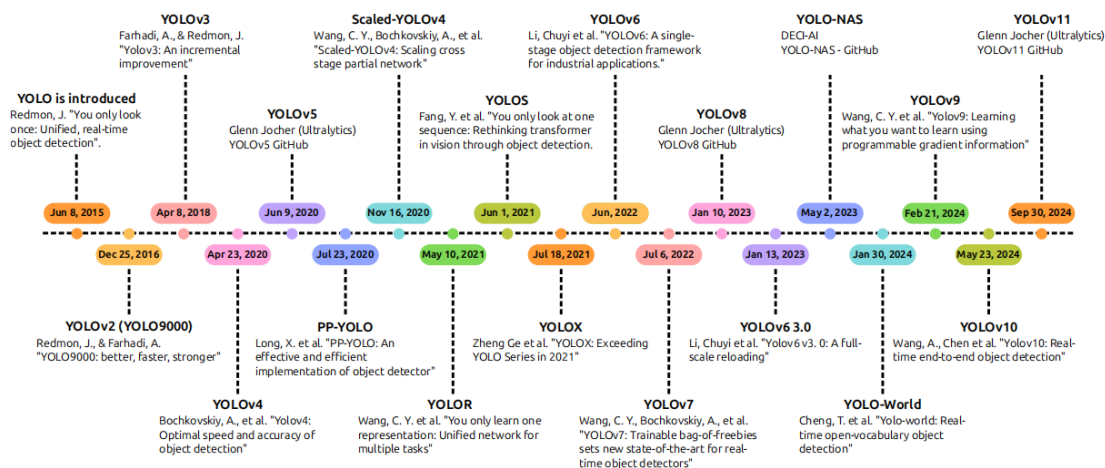


Figure 3-10 YOLO versions

### 3.6 Choose the best version for Face Detection

The selection of the optimal YOLO version for our face recognition attendance system project hinges upon two primary factors: speed and accuracy, in order to achieve the most favorable results.

Herein lies a comparison of several YOLO versions employed for face detection, undertaken to determine the optimal choice.

### 3.6.1 YOLOv8n-Face-Detection

This model was fine-tuned on a dataset of over 10k images containing human faces. The model was fine tuned for 100 epochs and gives these results.

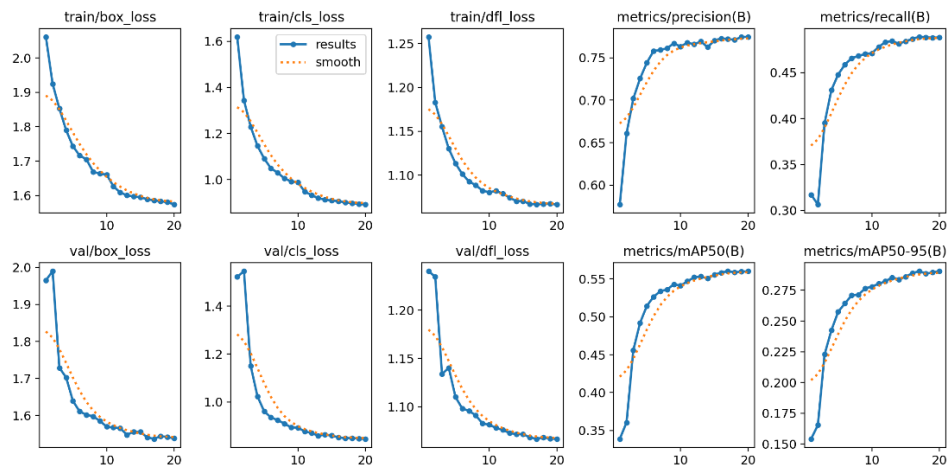


Figure 3-11 YOLOv8 results

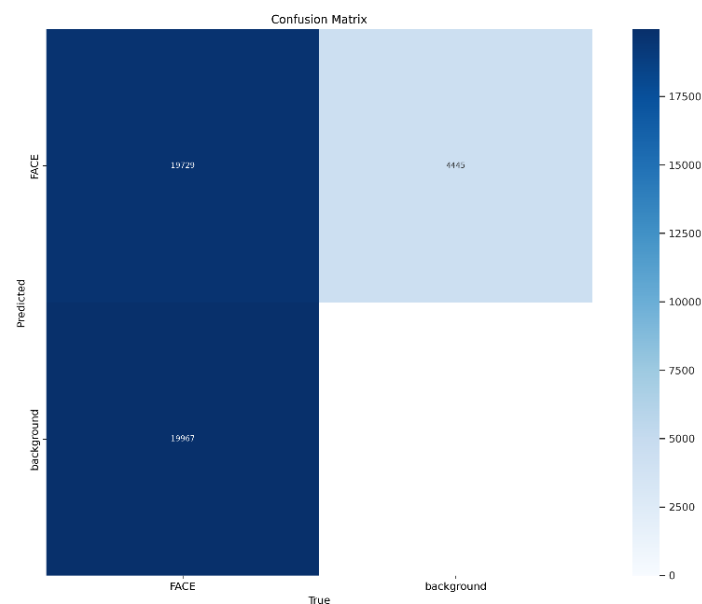


Figure 3-12 YOLOv8 confusion matrix

### 3.6.2 YOLOv11n-Face-Detection

WIDER FACE dataset is a face detection benchmark dataset, of which images are selected from the publicly available WIDER dataset. We chose 32,203 images and labeled 393,703 faces with a high degree of variability in scale, pose, and occlusion as depicted in the sample images.

A lightweight face detection model based on YOLO architecture (YOLOv11 nano), trained for 225 epochs on the WIDERFACE dataset and gives these results.

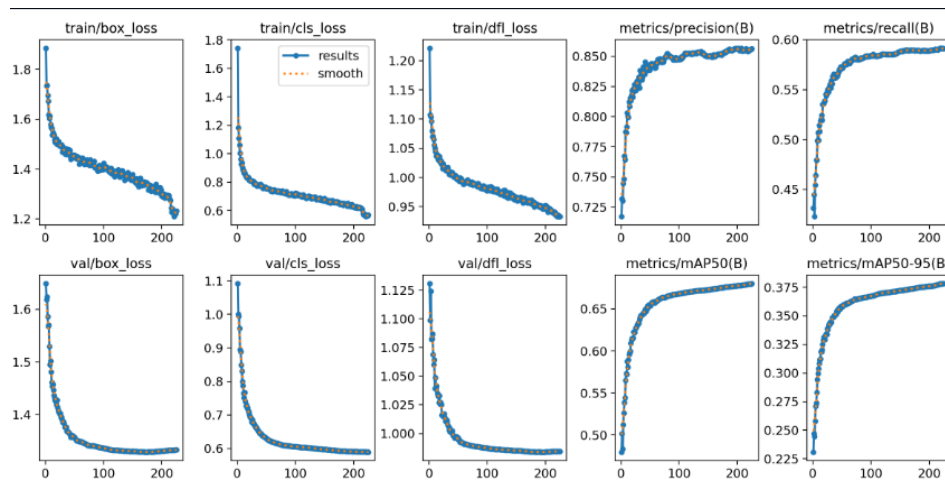


Figure 3-13 YOLOv11 results

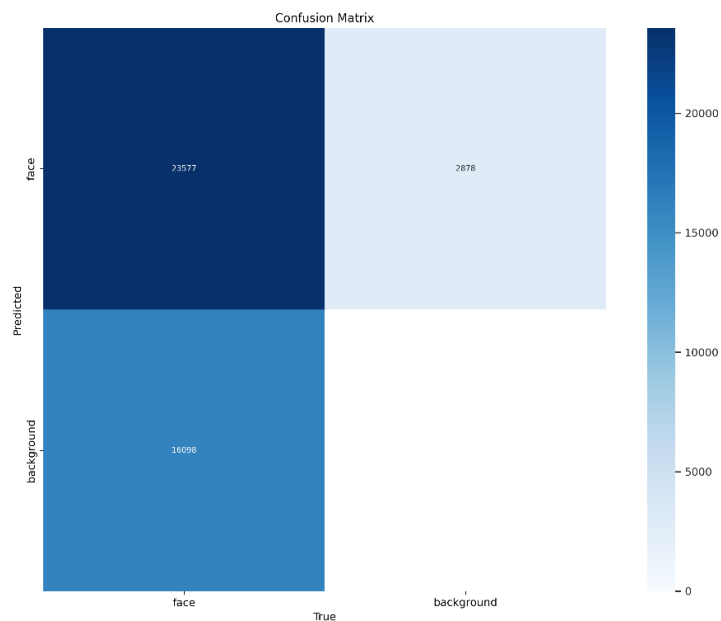


Figure 3-14 YOLOv11 confusion matrix

Based on training results, **YOLOv11** was selected as successfully meets the required criteria.

## 3.7 YOLOv11 in detail

### 3.7.1 Overview

YOLO11 is the latest iteration in the Ultralytics YOLO series of real-time object detectors, redefining what's possible with cutting-edge accuracy, speed, and efficiency. Building upon the impressive advancements of previous YOLO versions, YOLO11 introduces significant improvements in architecture and training methods, making it a versatile choice for a wide range of computer vision tasks.

### 3.7.2 Key Features

- **Enhanced Feature Extraction:** YOLO11 employs an improved backbone and neck architecture, which enhances feature extraction capabilities for more precise object detection and complex task performance.
- **Optimized for Efficiency and Speed:** YOLO11 introduces refined architectural designs and optimized training pipelines, delivering faster processing speeds and maintaining an optimal balance between accuracy and performance.
- **Greater Accuracy with Fewer Parameters:** With advancements in model design, YOLO11m achieves a higher mean Average Precision (mAP) on the COCO dataset while using 22% fewer parameters than YOLOv8m, making it computationally efficient without compromising accuracy.
- **Adaptability Across Environments:** YOLO11 can be seamlessly deployed across various environments, including edge devices, cloud platforms, and systems supporting NVIDIA GPUs, ensuring maximum flexibility.
- **Broad Range of Supported Tasks:** Whether it's object detection, instance segmentation, image classification, pose estimation, or oriented object detection (OBB), YOLO11 is designed to cater to a diverse set of computer vision challenges.

### 3.7.3 Tasks can YOLO11 models perform

YOLO11 models are versatile and support a wide range of computer vision tasks, including:

- **Object Detection:** Identifying and locating objects within an image.
- **Instance Segmentation:** Detecting objects and delineating their boundaries.
- **Image Classification:** Categorizing images into predefined classes.

- **Pose Estimation:** Detecting and tracking keypoints on human bodies.
- **Oriented Object Detection (OBB):** Detecting objects with rotation for higher precision.

### 3.7.4 Detailed Explanation of the YOLOv11 Architecture

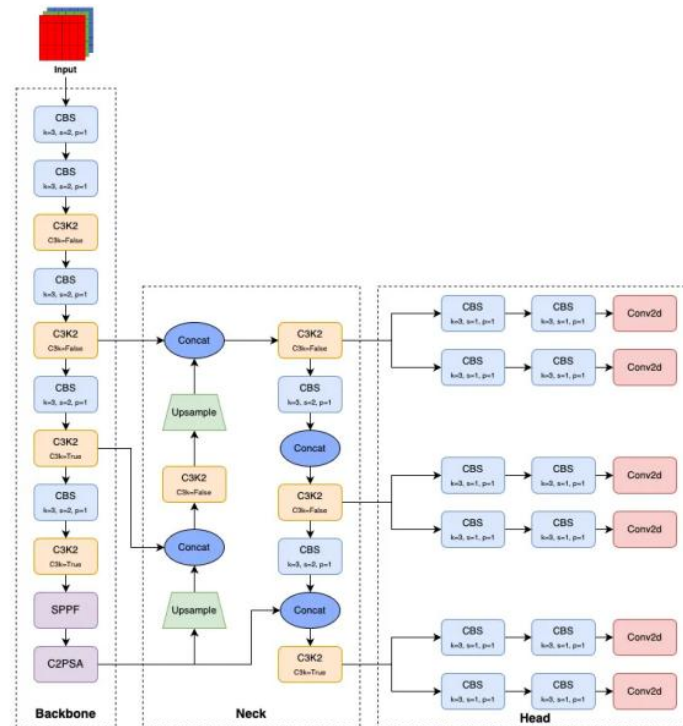


Figure 3-15 YOLOv11 architecture

#### 3.7.4.1 Individual Blocks Explanation

##### □ Input Block:

- Takes an RGB image of fixed size as input to the neural network.

##### □ CBS Block (Conv + BatchNorm + Swish/SiLU):

- **Convolution (C):** Extracts features using filters.
- **BatchNorm (B):** Normalizes output for faster, stable training.
- **Swish/SiLU (S):** Activation function to learn complex patterns.

##### □ C3K2 Block:

- Based on **CSPNet**, it splits features into two paths:
  - One goes through bottleneck layers (like CBS).
  - The other bypasses and then both are merged.
- Reduces computation while maintaining accuracy.
- K2 likely refers to specific configuration details (like kernel or layers).

#### □ **SPPF (Spatial Pyramid Pooling – Fast):**

- Extracts multi-scale context using chained pooling operations.
- Concatenates different scale outputs to help with detecting objects at varying sizes.

#### □ **C2PSA Block:**

- A variant combining CSP-like split with **spatial attention and pooling**.
- Helps the model focus on important regions in the image.

#### □ **Concat:**

- Merges feature maps from different layers (along the channel axis).

#### □ **Upsample:**

- Increases the resolution of feature maps (e.g., via interpolation) to help combine low-res semantic info with high-res spatial info.

#### □ **Conv2D (in Head):**

- Final convolutional layer (no activation/normalization) to output predictions: boxes, objectness, and class scores.

### 3.7.4.2 Stage-by-Stage Flow

#### 1. Backbone:

- Processes the input image through **CBS** and **C3K2** blocks, reducing size and increasing feature abstraction.
- **SPPF** captures multi-scale features.
- **C2PSA** enhances feature quality.
- Intermediate outputs are passed to the **Neck**.

#### 2. Neck:

- Combines features using a **PAN-FPN** structure:
  - **Top-down (FPN):**
    - Uses **Upsample + Concat** to merge deep, abstract features with shallow, detailed ones.
  - **Bottom-up (PAN):**
    - Further refines and strengthens features through more **CBS + C3K2** blocks.
- Produces multi-scale output for detecting objects of different sizes.



### 3. Head:

- Receives features at **3 different scales** (small, medium, large).
- Each scale goes through **CBS blocks**, then a **Conv2D** layer for:
  - **Bounding box coordinates**
  - **Objectness score**
  - **Class probabilities**

## 3.8 Testing results



Figure 3-16 Testing Image

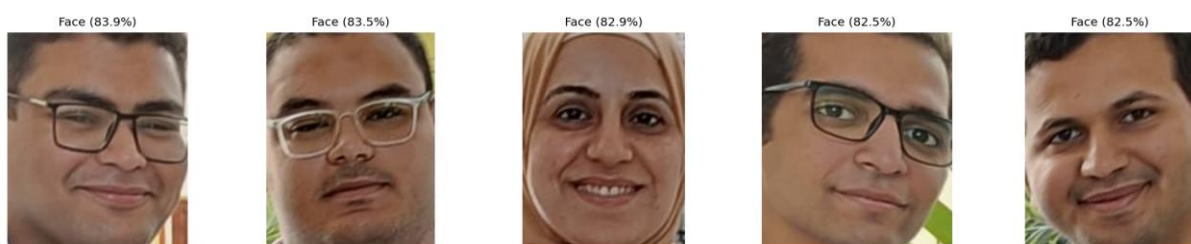


Figure 3-17 detected faces

# Chapter 4 : Face Recognition

## 4.1 Definition of Face Recognition

**Face recognition** is a sophisticated biometric technology that employs computer vision and machine learning algorithms to identify or verify an individual's identity by analyzing unique patterns and characteristics derived from their facial features.

It operates primarily in two modes:

1. **Identification (1-to-N matching):** Determining an unknown individual's identity by comparing their facial features against a database of many known faces.
2. **Verification (1-to-1 matching):** Confirming a claimed identity by comparing an individual's live or provided facial features against a single reference image associated with that claimed identity.

## 4.2 History of Face Recognition

The concept of face recognition is not new, nor is its implementation. The evolution of face recognition is fascinating and using computers to recognize faces has been dated back to the 1960's. The year **2014** was marked as an **important year** in the evolution of facial recognition as it reshaped the research landscape of this technology. It was the year when Facebook's DeepFace model's accuracy (97.35%) on the LFW benchmark dataset approached human performance (97.53%) for the first time. Just three years after this breakthrough, the accuracy of face recognition reached 99.80%.

## 4.3 Applications for Face Recognition

- **Security & Access** – Unlock devices, control entry to buildings, attendance systems.
- **Banking** – KYC, fraud prevention, secure transactions.
- **Personal Devices** – Face unlock, photo tagging, app security.
- **Healthcare** – Patient ID, monitoring, emotion/pain detection.
- **Education** – Online exam verification, smart attendance.
- **Smart Homes** – Personalized experiences, visitor alerts.

## 4.4 Techniques used in Face Recognition

### 4.4.1 Traditional Methods

#### 1. Eigenfaces (PCA-based Recognition)

A face recognition method that uses **Principal Component Analysis (PCA)** to reduce the dimensionality of face images and represent them using a set of "eigenfaces."

##### How it Works:

- Convert face images into grayscale and flatten them into vectors.
- Apply PCA to find the main components (eigenfaces) capturing variance across all training faces.
- Project a new face onto this eigenface space and compare it to known faces using distance metrics.



Figure 4-1 Eigenfaces working structure

##### Advantages:

- Simple and fast to compute.
- Good for dimensionality reduction and compression.
- Works in controlled environments.

##### Disadvantages:

- Sensitive to lighting, pose, and facial expressions.
- Struggles with complex or real-world scenarios.
- Poor performance on occluded faces.

## 2. Fisherfaces (LDA-based Recognition)

A method that applies **Linear Discriminant Analysis (LDA)** to project face images onto a space that **maximizes class separability**.

### How it Works:

- First applies PCA to reduce noise and dimensionality.
- Then uses LDA to find a projection that enhances class discrimination.
- Faces are classified based on their position in this new subspace.

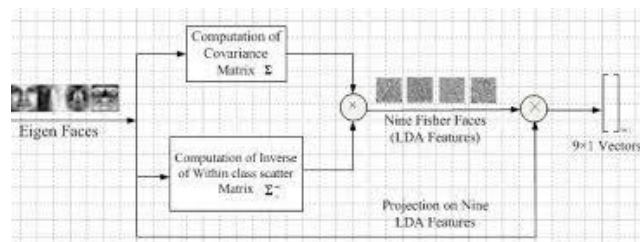


Figure 4-2 Fisherfaces working structure

### Advantages:

- Better than Eigenfaces in distinguishing between different individuals.
- More robust to lighting variations and within-class similarities.

### Disadvantages:

- Requires well-labeled training data.
- Less effective when the number of samples per class is small.

## 3. LBPH (Local Binary Patterns Histogram)

A **texture-based method** that analyzes local patterns in a face and builds histograms to represent the face structure.

### How it Works:

- Divides the face into small grids.
- For each pixel, compares neighboring pixels to form binary patterns.
- Converts these patterns into decimal values and builds a histogram.
- Concatenates all histograms as the feature vector for matching.

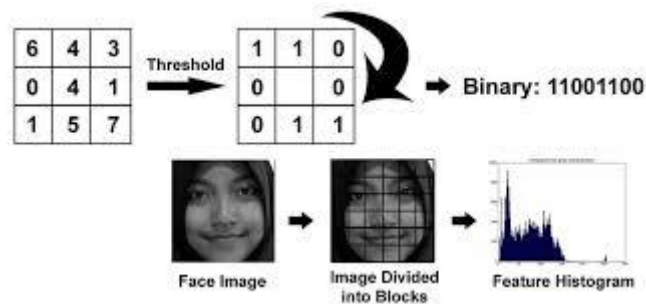


Figure 4-3 LBPH working structure

#### Advantages:

- **Fast and computationally efficient.**
- Works well with **lighting and expression changes**.
- Suitable for real-time and low-resource environments.

#### Disadvantages:

- Not very accurate in large-scale or complex datasets.
- More reliant on low-level texture patterns, less abstract than deep learning.

#### 4. DLib

An **open-source C++ toolkit** with Python support that provides high-quality face detection and recognition using **deep learning** (based on ResNet architecture).

#### How it Works:

- Detects facial landmarks to align faces.
- Extracts 128-dimensional face embeddings using a pre-trained deep CNN.
- Compares embeddings using Euclidean distance to identify or verify faces.



Figure 4-4 DLib working structure

### Advantages:

- **High accuracy**, comparable to modern deep learning methods.
- Offers **pre-trained models** for easy implementation.
- Supports facial alignment, detection, and recognition.

### Disadvantages:

- **Requires more computation** than traditional methods.
- Slower on devices without GPU.
- Less customizable compared to training your own model from scratch.

## 4.4.1.1 State-of-the-Art Models for Face Recognition

### 1. FaceNet

#### Definition:

FaceNet introduced **triplet loss** to train a deep CNN that maps face images to a **high-dimensional embedding space**, where similar faces are close together.

#### How it Works:

- Uses **triplets**: an anchor, a positive (same identity), and a negative (different identity).
- Optimizes the network so that the anchor is closer to the positive than the negative in embedding space.
- Embedding vectors can be compared using **Euclidean distance**.

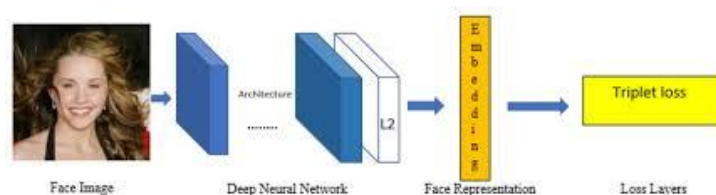


Figure 4-5 FaceNet structure

### Advantages:

- Learns **compact and discriminative embeddings**.
- Supports **face clustering, verification, and recognition** with a single embedding.

- No need for explicit classification layers.

### Disadvantages:

- Requires careful **triplet mining**.
- Training is **slower** and more complex than standard classification models.

## 2. DeepFace

One of the earliest deep learning-based face recognition models. Uses a **deep CNN** trained on a large labeled dataset to perform **face verification and identification**.

### How it Works:

- Aligns face using 3D modeling.
- Passes aligned image through a 9-layer CNN.
- Uses softmax classification and later performs verification using the final layer's features.

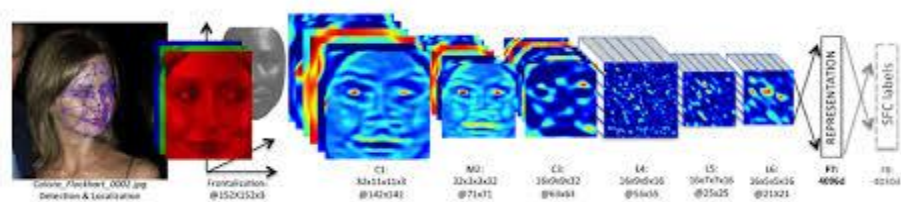


Figure 4-6 DeepFace structure

### Advantages:

- **First major breakthrough** in deep face recognition.
- High performance with **alignment + deep features**.
- Handles both **identification and verification**.

### Disadvantages:

- Dependent on **accurate face alignment**.
- Performance limited compared to modern models.

### 3. ArcFace

ArcFace introduced Additive Angular Margin Loss, improving the discriminative power of face embeddings by enforcing angular separation between classes.

#### How it works

- Uses softmax with an added angular margin to enhance class separability.
- Embeddings lie on a **hypersphere**, improving **inter-class separation** and **intra-class compactness**.

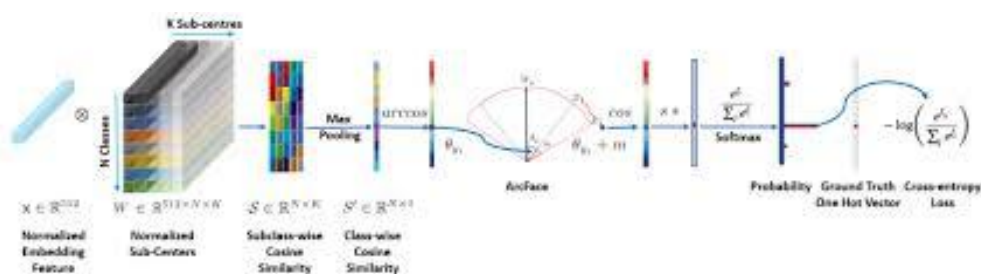


Figure 4-7 ArcFace structure

#### Advantages:

- **State-of-the-art accuracy** on many benchmarks.
- Simple to train, **end-to-end classification model**.
- Strong generalization across datasets.

#### Disadvantages:

- Requires **large-scale datasets** and **strong GPU hardware**.
- Works best with **balanced, high-quality training data**.

### 4. VGGFace & VGGFace2

Deep CNN models based on VGG architecture, trained on large-scale face datasets (VGGFace, VGGFace2) to learn robust face features.

#### How it Works:



- Uses deep convolutional layers (like VGG-16, VGG-19).
- Trained on millions of labeled faces.
- Extracts embeddings that can be compared using distance measures.

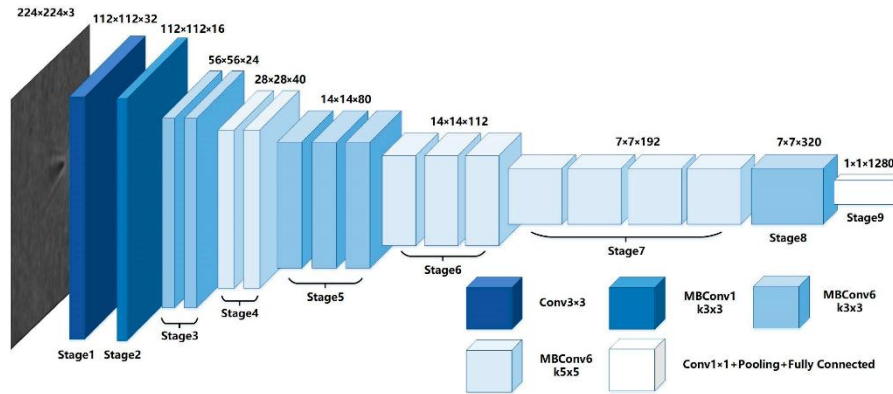


Figure 4-8 VGGFace structure

#### Advantages:

- Easy to implement with **public pre-trained models**.
- Works well in **identification and verification** tasks.
- VGGFace2 includes more diversity (pose, age, lighting).

#### Disadvantages:

- Larger and **heavier** than modern architectures.
- Less efficient compared to models like MobileFaceNet or ArcFace.

This concludes our brief overview of the methods used for face recognition. We will now delve more deeply into the specific algorithm chosen as the second step for our project.

## 4.5 FaceNet with architecture Inception-ResNet v1

### 4.5.1 Definition

FaceNet is a deep learning model developed by Google researchers in 2015 to directly learn a mapping from face images to a compact Euclidean space. In this space, distances directly correspond to a measure of face similarity. Unlike traditional classification networks, FaceNet doesn't output class probabilities; instead, it produces **128-dimensional embeddings** for each face.

## 4.5.2 FaceNet with Inception-ResNet v1 Architecture Breakdown

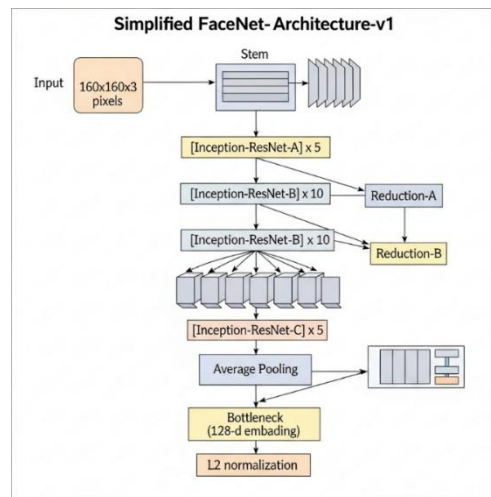


Figure 4-9 FaceNet with Inception-ResNet v1 Architecture

1. **Input (160×160×3):**
  - The model receives a cropped and aligned RGB face image. Standardizing input size ensures consistent feature learning.
2. **Stem (Initial Convolutions):**
  - A few convolutional layers with strides and batch normalization.
  - Purpose: Downsample the image and extract basic visual features.
3. **Inception-ResNet-A ×5:**
  - Combines **Inception** (multi-path filters) and **ResNet** (skip connections).
  - Repeated 5 times to capture increasingly complex features while preserving information.
4. **Reduction-A:**
  - Downsamples feature maps using pooling or strided convolutions.
  - Increases receptive field for next layers.
5. **Inception-ResNet-B ×10:**
  - Another type of residual Inception block with a new configuration.
  - Deeper feature extraction stage, repeated 10 times for refinement.
6. **Reduction-B:**
  - Further downsampling with possibly different strategies than Reduction-A.
  - Prepares features for final processing.
7. **Inception-ResNet-C ×5:**
  - Final specialized block tailored to deep, abstract feature learning.
  - Captures high-level identity features.
8. **Global Average Pooling:**
  - Averages each feature map into a single value.
  - Output is a compact vector, reduces model size and overfitting risk.
9. **Bottleneck (128/512-d Embedding):**
  - A fully connected layer turns the pooled features into a face **embedding**.
  - This numerical vector represents the identity of the face.

## 10. L2 Normalization:

- Scales the embedding to have unit length (norm = 1).
- Ensures stable training and makes distance comparisons meaningful.

## 4.6 Dataset Collection and Creation of Embeddings for each student

### 4.6.2 Collect about 20-30 images for each student and save images for each student in a folder named with student ID

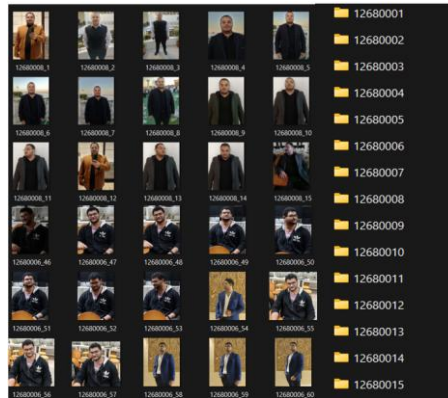


Figure 4-10 dataset

### 4.6.3 Split images into 80% for training and 20% for testing

```
Class 12680001: 24 images (19 train, 5 val)
Class 12680002: 16 images (12 train, 4 val)
Class 12680003: 16 images (12 train, 4 val)
Class 12680004: 22 images (17 train, 5 val)
Class 12680005: 20 images (16 train, 4 val)
Class 12680006: 60 images (48 train, 12 val)
Class 12680007: 61 images (48 train, 13 val)
Class 12680008: 65 images (52 train, 13 val)
Class 12680009: 24 images (19 train, 5 val)
Class 12680010: 24 images (19 train, 5 val)
Class 12680011: 33 images (26 train, 7 val)
Class 12680012: 25 images (20 train, 5 val)
Class 12680013: 45 images (36 train, 9 val)
Class 12680014: 22 images (17 train, 5 val)
Class 12680015: 24 images (19 train, 5 val)
Splitting completed successfully!
```

Figure 4-11 split dataset

#### 4.6.4 Extract faces from images using pretrained model YOLOv11 and resize them to 160\*160



Extracted face shape: (160, 160, 3)

Figure 4-12 extract faces and resize them

#### 4.6.5 Preprocess faces for FaceNet

- Color Conversion: Models like FaceNet expect RGB images. Many input streams (e.g., from OpenCV) are in BGR format.
- Tensor Conversion: To feed data into deep learning models (e.g., FaceNet), the pre-processed face image must be converted into a PyTorch tensor with appropriate dimensions
- Cropping: Extract the face region from the image using the bounding box.
- Resizing: Resize the cropped face to 160×160 pixels as expected by FaceNet.
- Normalization: Scale pixel values (e.g., between -1 and 1 or 0 and 1).

#### 4.6.6 Generate Embedding

The core of the system is the **FaceNet model**, which utilizes the **Inception-ResNet v1** architecture.

##### What happens here:

- The preprocessed face image is passed through the network.
- The network generates a **128-dimensional embedding vector** that captures the unique features of the face.
- This embedding is **L2-normalized** so that all output vectors lie on a unit hypersphere.

Key Concept: **Similar faces** result in embeddings that are **closer** in Euclidean space.

```
Loaded dataset shapes: (380, 160, 160, 3) (380,) (101, 160, 160, 3) (101,)
Processed training faces shape: torch.Size([380, 3, 160, 160])
Training embeddings shape: (380, 512)
Student embeddings created for: ['126800001', '126800002', '126800003', '126800004', '126800005', '126800006', '126800007', '126800008', '126800009', '126800010', '126800011', '126800012', '126800013', '126800014', '126800015']
```

“Now everything is ready for real time Face Recognition.”

## 4.7 Real time Face Recognition

### 4.7.1 Detect Faces from input image

Before recognition, the system must first **locate faces** within an input image or video frame. This is done using a **face detection algorithm (Pretrained Model YOLOv11 as shown in chapter 3)**

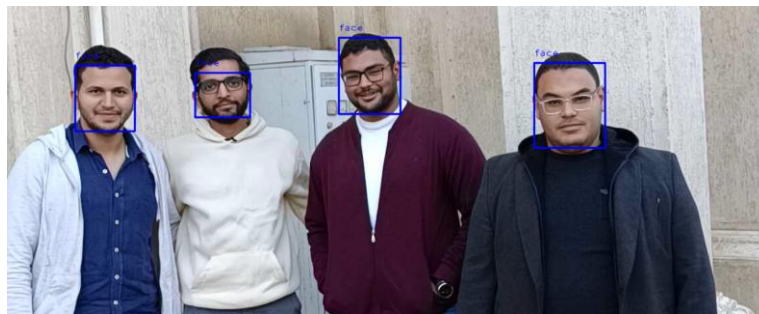


Figure 4-13 detection of faces

The Output of this step: Bounding boxes around detected faces.

### 4.7.2 Preprocessing

Once faces are detected, they must be **preprocessed** to ensure consistency across all inputs to the network.

#### Steps in Preprocessing:

- **Color Conversion:** Models like FaceNet expect RGB images. Many input streams (e.g., from OpenCV) are in BGR format.
- **Tensor Conversion:** To feed data into deep learning models (e.g., FaceNet), the pre-processed face image must be converted into a PyTorch tensor with appropriate dimensions
- **Cropping:** Extract the face region from the image using the bounding box.
- **Resizing:** Resize the cropped face to 160×160 pixels as expected by FaceNet.

- Normalization: Scale pixel values (e.g., between -1 and 1 or 0 and 1).

This step ensures that all faces are standardized before feeding into the embedding model.

### 4.7.3 Embedding generation

Generate Embedding for detected Faces to compare them with stored Embedding.

### 4.7.4 Recognition

The final stage is to determine whether a new face matches any known identity.


- **Compare** the new embedding with a reference embedding.
- **Use Euclidean distance** to calculate similarity.
- If the distance is **below a threshold** (e.g., 0.7), consider it a match.

### 4.7.5 Results



*Figure 4-14 testing image*

```
0: 640x480 5 faces, 51.2ms
Speed: 3.0ms preprocess, 51.2ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 480)
Final report with only student IDs generated.
```



```
12680006
12680007
12680008
12680011
```

*Figure 4-15 output*

**\*\*Note that the system detects all people in the image but does not recognize all because the dataset contains specific student's images. \*\***

## **4.8 Limitations**

The model's efficiency may be impacted by various constraints, including:

1. **LIGHT:** Performance may vary in extreme lighting conditions.
2. **FACE ANGLE:** Best suited for frontal and slightly angled faces.
3. **RESOLUTION:** Camera resolution may affect model performance.

# Chapter 5 : System Back-End

## 5.1 System Back-End Development

### 5.1.1 Definition of Backend Development

Backend development refers to the server-side logic and infrastructure that support the functionality of an application. It is responsible for processing incoming requests, managing data, communicating with external services, and ensuring secure, accurate, and consistent execution of the application's logic. In the context of the Smart Attendance System, the backend acts as the core engine that powers all system functionalities — from receiving face recognition results to updating attendance records and handling user authentication. While the frontend represents what users interact with visually, the backend operates behind the scenes to make everything function seamlessly.

### 5.1.2 Frameworks Used in the Backend

The backend of the Smart Attendance System is developed using Django, a high-level Python web framework. Django is chosen for its rapid development capabilities, clean architectural design, and built-in tools for security and scalability. It includes an Object-Relational Mapping (ORM) system that lets developers work with the database using Python classes instead of raw SQL queries, simplifying development and reducing errors.

In addition, Django REST Framework (DRF) is integrated into the system to build RESTful APIs. DRF allows the backend to expose its functionalities over HTTP so that the mobile app and other clients can interact with it. These APIs cover core actions such as login, attendance marking, data retrieval, and student request handling.

### 5.1.3 Backend Responsibilities

The backend of the Smart Attendance System is responsible for a wide range of tasks:

- **Data Management:** It handles all CRUD operations (Create, Read, Update, Delete) on the system's database, including storing student records, attendance logs, and leave requests.
- **Logic Implementation:** Business rules such as role-based permissions, attendance conditions, and request workflows are enforced here.



- **AI Integration:** It communicates with the face recognition module. When an image is sent to the backend, it is passed to the AI system (YOLO for face detection, FaceNet for recognition), and the result is used to update the database automatically.
- **Security Enforcement:** The backend authenticates users using JWT (JSON Web Tokens) and ensures that all actions are authorized based on user roles (student, instructor, admin).
- **API Communication:** All features are exposed through RESTful APIs, allowing frontend applications to consume and interact with backend resources in a standardized way.

### 5.1.4 Technologies Used in the Backend

The main technologies and tools used to build and run the backend include:

- **Python:** The primary programming language used in both Django and the AI modules.
- **Django:** The main web framework for handling routing, views, models, and backend logic.
- **Django REST Framework:** A powerful toolkit for building REST APIs used by the mobile app and frontend.
- **PostgreSQL:** The relational database management system used to store all records securely and efficiently.
- **JWT (JSON Web Tokens):** Used for secure, stateless user authentication across requests.
- **YOLO:** An AI model used for real-time object detection, specifically to identify faces in images.
- **FaceNet:** An embedding-based face recognition model that verifies identities by comparing facial embeddings.
- **OpenCV & Torch:** Supporting libraries for image processing and running AI models.

## 5.2 System Modeling – Backend

The modeling of the backend was designed with scalability, modularity, and maintainability in mind. From the very beginning, the system was broken into separate Django applications — each responsible for a single major domain of the system. This modular structure allows developers to work on different parts of the system simultaneously, without overlapping responsibilities or increasing the chance of bugs.

Each app encapsulates its own logic, models, views, serializers, and API routes. For example, the users app is in charge of managing accounts, user roles, and login functionality. The attendance app handles everything related to session tracking and student presence. Meanwhile, the academics app is responsible for organizing departments, academic years, and subjects. The recognition app deals with image processing and interaction with the AI model, and the requests app provides a structured way for students to submit absence or leave requests.

### 5.2.1 MVC Architecture in Django

Even though Django officially follows the MTV (Model-Template-View) design pattern, the backend in this project aligns more with the traditional MVC (Model-View-Controller) structure — especially since the system exposes APIs rather than HTML templates.

- **Model:** Defines the data structure and schema. For example, models such as Student, Attendance, ClassSession, and Request each correspond to real database tables and include field definitions and data relationships.
- **View:** Handles incoming requests, processes logic, and returns a response. In this system, views are used to handle tasks such as marking attendance, submitting leave requests, or fetching academic data.
- **Controller (Serializer in DRF):** Serializers validate incoming data, convert complex Python objects into JSON format for the frontend, and help structure output for APIs.

By following this structure, the system ensures clarity in data flow, separation of concerns, and easier maintenance.

## 5.2.2 Class Diagram

A class diagram provides a clear blueprint of the relationships between models and objects in the backend. In this system, the diagram starts with a central Account model — representing the base user. This model is extended through one-to-one relationships into Student and Instructor models, depending on the user’s role.

Students are linked to academic years, subjects, and attendance records. Instructors are associated with departments and have the ability to approve or reject student requests. Each class has well-defined fields — such as `student_code`, `department_name`, `session_date` — and each relationship is enforced through Django’s ORM using `ForeignKey`, `OneToOneField`, or `ManyToManyField`.

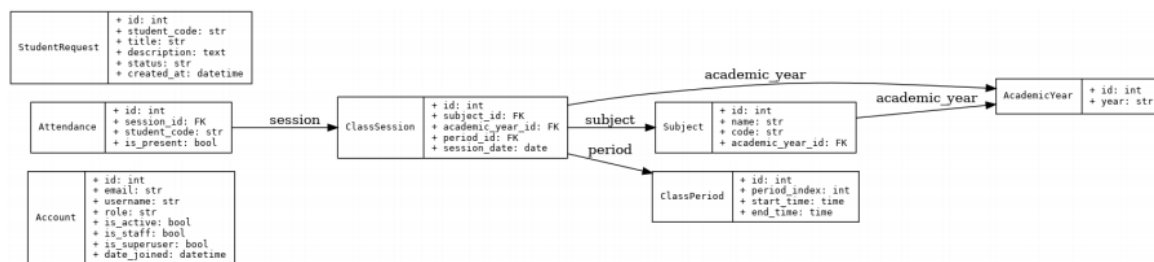


Figure 5-1 backend class diagram

This design ensures relational integrity, efficient queries, and a normalized database schema.

These interactions are supported entirely by backend logic. Each use case represents a real action within the system, such as calling an API to submit a request or process recognition data.

## 5.2.3 Sequence Diagram

Sequence diagrams are used to illustrate the chronological flow of operations within the system. In the backend of the Smart Attendance System, a typical sequence starts with an instructor initiating the attendance process.

1. The instructor logs into the system through the frontend (mobile or web).
2. The instructor begins a new class session. The backend logs this session and generates a reference ID.

3. The camera is activated, and a live image of the classroom is captured.
4. This image is sent via HTTP POST to the backend's /recognize/ API.
5. The backend receives the image and forwards it to the AI module (YOLO + FaceNet).
6. The AI module returns the identity (student code) for each recognized student.
7. The backend verifies these codes against the database.
8. Valid matches are recorded in the attendance table and linked to the active session.
9. The instructor receives confirmation that attendance has been recorded.

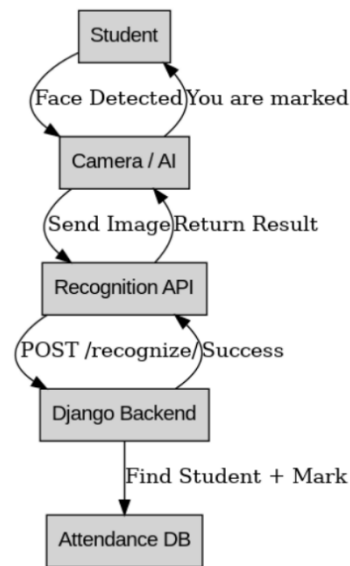


Figure 5-2 backend sequence diagram

This real-time interaction between the user, backend, and AI system requires precise synchronization, and the backend is responsible for maintaining this flow reliably.

### 5.3 Database: Design and Technologies

The backend's database serves as the system's long-term memory. It holds all structured data required to operate the platform — from user details and student records to attendance history and AI recognition logs.

PostgreSQL was selected as the database engine due to its stability, support for advanced queries, and compatibility with Django's ORM. Each Django model in the backend corresponds to a table in PostgreSQL, and the relationships between them are enforced by foreign keys.

The database is normalized to avoid redundancy. For instance:

- Student records are not duplicated across sessions. Instead, attendance records point to a specific student and session using foreign keys.
- Academic data is centralized in models like `AcademicYear` and `Subject`, which can be referenced throughout the system.
- Many-to-many relationships are used where needed, such as linking students to subjects.

Django's ORM automatically handles migrations — meaning whenever a model is created or updated in code, the system can generate SQL scripts to reflect those changes in the database schema. This streamlines the development process and reduces the chance of manual database errors.

```
1. Account
-----
id            INTEGER      PRIMARY KEY
email         VARCHAR      UNIQUE NOT NULL
username      VARCHAR      NOT NULL
role          VARCHAR      CHECK(role IN ['admin', 'instructor', 'student'])
is_active     BOOLEAN      DEFAULT TRUE
is_staff      BOOLEAN
is_superuser  BOOLEAN
date_joined   TIMESTAMP

2. AcademicYear
-----
id            INTEGER      PRIMARY KEY
year          VARCHAR      UNIQUE NOT NULL

3. Subject
-----
id            INTEGER      PRIMARY KEY
name          VARCHAR
code          VARCHAR      UNIQUE
academic_year_id  INTEGER    FOREIGN KEY -> AcademicYear(id)
```

```

4. ClassPeriod
-----
id            INTEGER          PRIMARY KEY
period_index  INTEGER
start_time    TIME
end_time      TIME

5. ClassSession
-----
id            INTEGER          PRIMARY KEY
subject_id    INTEGER          FOREIGN KEY -> Subject(id)
academic_year_id INTEGER      FOREIGN KEY -> AcademicYear(id)
period_id     INTEGER          FOREIGN KEY -> ClassPeriod(id)
session_date  DATE
UNIQUE(subject_id, period_id, session_date)

6. Attendance
-----
id            INTEGER          PRIMARY KEY session_id
INTEGER       FOREIGN KEY -> ClassSession(id) student_code
VARCHAR is_present    BOOLEAN UNIQUE(session_id,
student_code)

7. StudentRequest
-----id            INTEGER          PRIMARY KEY student_code
VARCHAR title        VARCHAR description    TEXT status        VARCHAR
CHECK(status IN ['pending', 'approved', 'rejected']) created_at
TIMESTAMP

```

*Figure 5-3 database schema*

## 5.4 RESTful API Design

The Smart Attendance System’s backend is fully API-driven, which means all core features are accessed and controlled through RESTful endpoints. These APIs act as the gateway between users (via mobile or web) and the backend logic. Each request sent to the system is processed, validated, and responded to using HTTP — often returning structured data in JSON format.

### 5.4.1 Structure and Organization

The APIs are structured around Django REST Framework (DRF), which allows each Django app to expose its own URLs for different operations. For example:

- The users app provides authentication endpoints such as:
  - POST /login/ — for logging in using JWT.
  - POST /register/ — for creating a new user account (admin only).
- The attendance app includes endpoints such as:
  - POST /attendance/mark/ — used to record a student’s presence.

- GET /attendance/history/<student\_id>/ — used to retrieve attendance logs.
- The recognition app contains:
  - POST /recognize/ — receives an image file and interacts with the AI model to recognize students.
- The requests app provides:
  - POST /requests/ — for submitting new leave or absence requests.
  - GET /requests/ — for instructors to view pending requests.
  - PATCH /requests/<id>/ — to approve or reject a specific request.

This modular routing system allows each feature to evolve independently while preserving a clear and logical API structure.

### 5.4.2 HTTP Methods and Status Codes

All APIs follow the REST standard, using HTTP methods appropriately:

- GET for retrieving data.
- POST for submitting new data.
- PUT or PATCH for updating existing records.
- DELETE for removing entries (if permitted).

Each API responds with meaningful HTTP status codes. For instance:

- 200 OK indicates a successful response.
- 201 Created is used after creating a new record.
- 400 Bad Request signals a validation failure.
- 401 Unauthorized when a user fails to authenticate.
- 403 Forbidden if a user lacks permission for a given action.

By following these conventions, the backend maintains predictability and ease of integration for developers working on the frontend or mobile app.

### 5.4.3 Serialization and Validation

All data that enters or exits the backend passes through DRF **serializers**. These serializers are responsible for:

- Validating input data (e.g., making sure required fields are present).
- Enforcing rules (e.g., only instructors can update requests).
- Formatting outgoing data into JSON for the client.

Each model in the system has a corresponding serializer. For example:

- StudentSerializer includes fields like name, student\_code, academic\_year.
- AttendanceSerializer formats the presence status and session time.
- RequestSerializer ensures a request includes a valid reason and student ID.

This layer ensures security, consistency, and clarity in all API communication.

## 5.5 Security in the Backend

Security is a critical element of any system that stores personal data, handles authentication, and processes sensitive academic records. The backend is designed with multiple layers of protection to guard against unauthorized access, data leaks, and malicious actions.

### 5.5.1 Authentication with JWT

The system uses **JWT (JSON Web Tokens)** for user authentication. When a user logs in successfully, the backend returns a signed token that contains encrypted user data. This token must be attached to every subsequent API request in the headers (typically via Authorization: Bearer <token>). The backend then verifies the token's authenticity before granting access.

This method of stateless authentication allows for scalability and ensures that user sessions are handled securely without storing session data on the server.

### 5.5.2 Role-Based Access Control

Each user in the system is assigned a **role** at the time of registration. These include:

- **Admin:** Full control over system configuration, academic data, and users.
- **Instructor:** Manages student attendance and reviews leave requests.
- **Student:** Can view personal attendance data and submit absence requests.



Permissions are enforced in the backend through:

- Custom permission classes in DRF.
- Role checks in views and serializers.
- Filtering querysets to prevent unauthorized data exposure.

This guarantees that each user only sees or modifies data relevant to their role.

### 5.2.3 Input Validation and Error Handling

To prevent vulnerabilities such as injection attacks or invalid data submissions, the backend uses strict validation on all incoming data. Serializers check for required fields, correct formats, and logical consistency (e.g., ensuring a request can't be submitted for a past session).

Additionally, the system includes error handling mechanisms that log exceptions and return meaningful error messages, helping developers troubleshoot and keeping the system stable under unexpected conditions.

## 5.6 Documentation and Maintainability

One of the key indicators of a high-quality backend system is not just how well it works, but how easy it is for other developers to understand, use, and extend it. In a project like the Smart Attendance System — where multiple contributors may work on the codebase or future enhancements might be added — clear and comprehensive documentation is critical.

### 5.6.1 Inline Documentation and Code Readability

Throughout the backend, functions, classes, and critical code blocks are annotated with **inline comments** that explain what each part of the code does. These comments are written in simple, developer-friendly language and follow Python's PEP8 formatting style.

For example:

- Every model includes comments describing its fields and purpose.
- Views explain the logic behind processing requests and permissions.
- Serializers are documented to clarify how data is validated or transformed.

This makes onboarding new developers or debugging much faster and more efficient.

## 5.6.2 API Documentation

The backend APIs are documented using tools such as:

- **DRF's built-in browsable API:** When running the backend locally, developers can visit a web interface to test each endpoint, view request formats, and see example responses.
- **Swagger/OpenAPI** (if enabled): This tool provides a visual interface with live documentation, showing all available routes, expected parameters, response structures, and error codes.

Each endpoint is grouped by app and function — e.g., authentication, attendance, AI recognition, or student requests — making it easy to navigate and test.

## 5.6.3 Folder Structure and Naming Conventions

The backend codebase is organized using Django's best practices. Each Django app follows a consistent layout:

- `models.py`: Defines the data structure and database tables.
- `serializers.py`: Manages data validation and formatting for APIs.
- `views.py`: Handles request processing and interaction with the database.
- `urls.py`: Maps API routes to their respective views.

Additionally, the naming of variables, classes, and functions follows clear, descriptive patterns. For instance:

- `mark_attendance()` is immediately clear in its purpose.
- `get_student_requests()` indicates the function retrieves all leave requests for a student.

This predictability makes the code easier to explore and maintain.

## 5.6.4 Reusability and Modularity

Thanks to the app-based structure of Django, each feature is encapsulated within its own module. This means if the attendance system ever needs to be reused in another project, it can be extracted with minimal changes. Similarly, new features can be added as separate apps — for example, a future notifications app — without affecting the existing codebase.

The use of **Django signals** also improves maintainability by allowing decoupled behavior. For instance, automatically logging timestamps or sending notifications when attendance is marked can be implemented without modifying the core logic.

### 5.6.5 Developer Guidelines and README Files

Each app contains a **README.md** file that serves as an entry point for developers. These files explain:

- The purpose of the app.
- The main models and their relationships.
- How to run and test the app.
- Any dependencies required (e.g., AI model setup for the recognition app).

These guides serve as mini-documentation sets and help developers get started quickly with specific parts of the system.

## 5.7 AI Integration from a Backend Perspective

The integration of artificial intelligence into the backend of the Smart Attendance System is one of its most powerful and transformative features. Rather than relying on manual input or ID cards for attendance, the system uses face recognition to automate this process. The backend plays a crucial role in enabling and managing this AI interaction — acting as the bridge between raw camera input and structured attendance data.

### 5.7.1 The Role of the Backend in AI Interaction

The AI models used in the project — **YOLOv11** for face detection and **FaceNet** for face recognition — operate outside the Django framework, typically in Python scripts that use computer vision libraries such as OpenCV and deep learning libraries like PyTorch or TensorFlow.

However, these models alone do not create a usable system. It is the backend that:

- Accepts image input (typically via a POST request to `/recognize/`).
- Sends the image to the AI model.
- Waits for a response containing recognized face embeddings or student IDs.
- Verifies this identity against the database.

- Marks attendance accordingly.

In this way, the backend wraps the AI logic in a secure, accessible, and user-friendly interface — turning it from a model into a full-fledged system feature.

### 5.7.2 AI Processing Workflow

Below is the high-level workflow that occurs when an image is submitted to the backend:

1. **Image Upload:** The mobile frontend or web interface captures an image and sends it to the backend using a POST request.
2. **AI Model Call:** The backend reads the image and passes it to a custom module (e.g., `camera_attendance.py`) where YOLO detects the faces.
3. **Embedding and Matching:** Detected faces are passed through FaceNet, which generates embeddings. These are compared against pre-stored embeddings in a dataset of known students.
4. **Student Code Extraction:** If a face matches with high enough confidence, the corresponding student code is retrieved.
5. **Attendance Logging:** The backend logs this match in the Attendance model, linking the student with the active `ClassSession`.

This workflow is executed in real-time and must handle multiple faces in a single frame efficiently.

### 5.7.3 Error Handling and Fallbacks

The backend also implements safeguards in case of:

- **Unrecognized Faces:** If no match is found, the student is not marked, and an error message is returned.
- **Duplicate Entries:** The backend ensures students aren't marked twice in the same session.
- **Poor Image Quality:** If the image cannot be processed due to lighting or occlusion, the backend returns a message asking for a better image.

These checks ensure the system remains robust, user-friendly, and resistant to common real-world issues.

### 5.7.4 Real-Time Operation and Performance

The backend is optimized for **low-latency communication** with the AI models. Image handling is done in-memory using libraries like PIL or NumPy, avoiding disk I/O where possible. This ensures fast performance even when multiple students are detected in a single frame.

The AI module can also be deployed as a **microservice**, which the backend contacts via internal HTTP calls or gRPC. This allows for horizontal scaling if needed — for example, handling recognition for multiple classrooms in parallel.

## 5.8 Testing, Debugging, and Logging

A robust backend is not only about writing code that works — it's about writing code that can be tested, maintained, debugged, and monitored. In the Smart Attendance System, several practices were implemented to ensure the backend is reliable, traceable, and easy to troubleshoot when issues arise.

### 5.8.1 Unit Testing and Integration Testing

Testing is essential to verify that each individual function of the backend behaves as expected. The backend codebase includes **unit tests** for models, views, and serializers. These tests check that:

- Models enforce constraints (e.g., required fields, unique values).
- Serializers validate data and return correct formats.
- Views handle expected inputs and reject invalid or unauthorized requests.

In addition to unit tests, **integration tests** are used to simulate real workflows. For example, a test might:

- Create a mock student.
- Simulate a login using the `/login/` API.
- Simulate submitting an image to `/recognize/`.
- Check if attendance is correctly created in the database.

These tests are automated and can be run after every change to the code to ensure nothing breaks unexpectedly.

## 5.8.2 Debugging Tools

During development, debugging is made easier through the use of:

- **Django Debug Toolbar:** A development tool that provides detailed information about each request, including SQL queries, middleware performance, and request/response content.
- **Postman or Insomnia:** Used to test API endpoints manually during development, helping to simulate frontend behavior and observe detailed responses.

Additionally, developers use logging and exception tracking to pinpoint issues, especially in error-prone areas like AI processing or database transactions.

## 5.8.3 Backend Logging

Logging is one of the most important tools for understanding what happens inside a running backend. Logs are used for:

- Tracking user activity (e.g., when a student logs in or submits a request).
- Debugging recognition failures (e.g., AI returned “no match”).
- Monitoring API usage (e.g., how many times /recognize/ is called daily).
- Capturing exceptions or unusual behavior.

The Django backend uses Python’s built-in logging module, which allows for messages to be saved at different levels (INFO, WARNING, ERROR, CRITICAL). These logs can be stored in text files locally or sent to external monitoring tools in production.

Example log entries:

INFO: Student 126213122 recognized in session 3 (2025-06-15)

WARNING: Unrecognized face detected – skipping attendance

ERROR: AI module timeout – image could not be processed

## 5.8.4 Exception Handling and Error Reporting

Backend views are wrapped with **try/except blocks** to catch unexpected errors without crashing the server. When an exception occurs:

- A descriptive error message is returned to the client.

- The error is logged in detail.
- Sensitive data is never exposed to the user.

Custom error responses are formatted to be frontend-friendly, helping developers or support teams quickly identify and resolve issues. For example:

```
{"detail": "Unable to mark attendance. Face not recognized. Please try again."}
```

## CI/CD and Automated Testing

If the system is to be deployed in a professional environment, the testing infrastructure can be linked to a **Continuous Integration/Continuous Deployment (CI/CD)** pipeline. This ensures:

- Tests are run automatically when changes are pushed to the repository.
- Code is only deployed if it passes all quality checks.
- Developers receive alerts if something fails.

## 5.9 Deployment and Scaling Considerations

Designing and coding a backend system is only part of the process. For the Smart Attendance System to work in a real-world environment, it must be **deployed** properly and capable of **scaling** to support multiple classrooms, campuses, or institutions. This section outlines how the backend was prepared for production use and how it can grow to accommodate future demands.

### 5.9.1 Deployment Environment

The backend is built using Django, a framework well-suited for both development and production environments. When preparing for deployment, several adjustments are made:

- **Debug mode is disabled**, ensuring sensitive error details are not exposed to users.
- **Static files** (such as admin dashboard styles) are collected and served through a web server like Nginx.
- **Environment variables** are used to store database credentials, secret keys, and AI paths securely.
- **Allowed hosts** are set in the Django settings to restrict access to authorized domains.

The system is typically deployed on platforms like:

- **Heroku, Render, or DigitalOcean** for smaller projects or testing.
- **AWS, Azure, or Google Cloud** for production environments.
- **Docker** containers for scalable, portable environments across development and production.

### 5.9.2 WSGI & ASGI Gateways

Django communicates with web servers through interfaces:

- **WSGI (Web Server Gateway Interface)** is used for traditional HTTP requests.
- **ASGI (Asynchronous Server Gateway Interface)** is prepared if future upgrades require real-time features like WebSocket communication (e.g., live attendance updates).

This dual readiness ensures the backend can evolve to meet future technical requirements.

### 5.9.3 Database Configuration

In production, the backend connects to a **managed PostgreSQL instance**, which offers benefits like:

- Automated backups and recovery options.
- Monitoring and performance metrics.
- Security features like SSL encryption and IP whitelisting.

Database migrations are applied via commands such as:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

These ensure that the production database schema always matches the Django models.

### 5.9.4 Performance Optimization

Several backend strategies are used to optimize speed and responsiveness:

- **Query optimization:** Reducing unnecessary database joins and using `select_related` or `prefetch_related` where needed.
- **Pagination:** Limiting data sent in each API response (e.g., only 10 attendance records at a time).



- **Caching:** Results from frequently requested queries (like subject lists) can be cached in memory using tools like Redis.
- **Image processing in memory:** The backend processes image uploads from the camera in memory (RAM) instead of saving them to disk, speeding up AI communication.

### 5.9.5 Horizontal Scaling

As more classrooms or devices begin using the system simultaneously, the backend must scale to handle this increased load. Horizontal scaling involves adding more instances of the backend server — each capable of handling requests independently.

This is enabled by:

- Stateless design (using JWT authentication instead of session storage).
- Decoupled architecture (e.g., AI module as a microservice).
- Using load balancers (e.g., Nginx, AWS ELB) to distribute traffic evenly.

### 5.9.6 Logging and Monitoring in Production

To maintain visibility over system health, the production backend includes:

- **Centralized logging** using tools like LogDNA, Papertrail, or ELK Stack.
- **Monitoring tools** like Prometheus or New Relic to track API response times, errors, and server metrics.
- **Alerts** to notify developers of unusual spikes in errors or downtime.

## 5.10 Summary of Backend Responsibilities

The backend of the Smart Attendance System is the engine that powers every function the system offers. Although it operates behind the scenes, it is responsible for nearly everything — from interpreting AI results and recording attendance, to authenticating users and ensuring secure, real-time communication between all components.

Through a modular, scalable, and secure architecture built with Django and Django REST Framework, the backend delivers a powerful API-based interface for mobile applications, administrators, instructors, and even the AI module. It enables seamless automation of tasks that would otherwise be repetitive and error-prone — such as attendance tracking, student record management, and leave request processing.

Key responsibilities of the backend include:

- **User Management:** Creating, updating, and authenticating users based on their roles (student, instructor, admin), with role-based permission handling.
- **Academic Data Handling:** Managing departments, academic years, class sessions, subjects, and their inter-relationships.
- **Face Recognition Integration:** Receiving images from the camera, interacting with the AI model to recognize students, and marking attendance accordingly.
- **Data Storage and Processing:** Structuring and saving records in a PostgreSQL database, using Django models to ensure relational integrity and optimized queries.
- **Request Handling:** Allowing students to submit absence requests, and instructors to approve or reject them — with state transitions and audit logging.
- **Security:** Using JWT authentication and strict access control to protect user data and system operations.
- **REST API Exposure:** Making every backend service available through RESTful endpoints so it can interact with frontends and third-party systems.
- **Testing, Debugging, and Maintenance:** Ensuring stability through automated testing, extensive logging, and development-friendly tools.
- **Deployment Readiness:** Supporting production deployment with configuration for performance, scalability, and monitoring.

All these responsibilities are carefully orchestrated and encapsulated within well-structured Django applications. Together, they form a backend system that is not only intelligent and functional, but also scalable, secure, and future-ready.

# Chapter 6 : System modeling and Front-End

## 6.1 The Context Diagram

represents the Smart Attendance System as a single unit and outlines its interaction with three main external entities: students, instructors, and administrators. Students can log in, send attendance requests, and receive real-time notifications. Instructors interact with the system by managing sessions and receiving analytics-based attendance reports. The admin or registrar manages academic structures, student records, and instructor assignments. This diagram helps in understanding the data exchange between users and the system without revealing internal complexities.

### 1. Mobile Application:

Platform: Android/iOS.

Framework: Flutter.

### 2. Desktop Application:

Platform: Windows/Mac/Linux.

Framework: Flutter.

### 3. Framework: Django

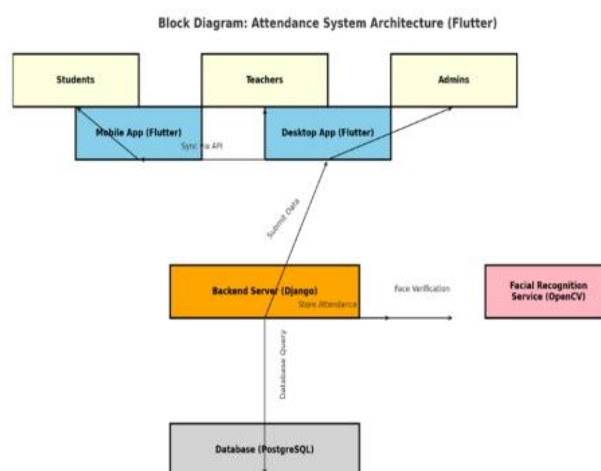


Figure 6-1 context diagram

## **6.2 Mobile Application Business Process (Student Side)**

The mobile application provides students with an intuitive and user-friendly interface to interact with the Smart Attendance System. The business processes involved in the mobile application include:

### **6.2.1 Login and Authentication**

- The student opens the application and logs in using their registered email and password.
- Upon successful authentication via the backend API, the student is redirected to the main dashboard.

### **6.2.2 Attendance Overview**

- The student can view a list of enrolled subjects.
- For each subject, the application displays:
  - Total number of sessions.
  - Number of sessions attended.
  - Number of absences or late entries.
  - A detailed session history including dates, session types (lecture/lab), attendance status, and any remarks.

### **6.2.3 Notifications**

- The system provides real-time push or in-app notifications such as:
  - Missed attendance alerts.
  - Late arrival warnings.
  - Upcoming session reminders with time and location.
- These notifications are dynamically generated based on student attendance records and session scheduling.

### **6.2.4 Absence Request Submission**

- Students can submit formal absence requests through the app.
- The request form includes:
  - Absence reason.

- Start and end date.
- Optional document upload (e.g., medical report).
- Requests are sent to the instructor/administrator for review and approval.

### **6.2.5 Profile and Settings**

- Students can view and update certain profile information (e.g., phone number, password).
- Language preferences and notification toggles can be adjusted via the settings page.
- Logout functionality is also available.

## **6.3 Web Application Business Process (Admin/Instructor Side)**

The web panel is designed for use by instructors and administrative staff to manage attendance, sessions, and academic data. Key processes include:

### **6.3.1 Admin Dashboard**

- The admin user can:
  - Manage student and instructor accounts.
  - Create academic years and assign departments.
  - Add, edit, or delete subjects.
  - Enroll students into subjects based on their academic level.

### **6.3.2 Instructor Dashboard**

- Instructors can:
  - View the subjects they teach.
  - Manage class sessions, including:
    - Room assignments.
    - Session types (lecture, lab, tutorial).
    - Weekly schedules (start date + number of weeks).
  - View student attendance reports filtered by subject and date.
  - Mark manual attendance if needed.

### **6.3.3 Session Scheduling**

- A reusable class session structure is defined for each subject and instructor.
- A separate SessionDate table dynamically generates specific dates (e.g., every Monday 10:00 AM) across the academic term.

### **6.3.4 Attendance Management**

- Attendance is auto-generated from AI-based facial recognition or manually adjusted by instructors.
- The system stores attendance records per student per session date, including:
  - Status: Present, Late, Absent.
  - Time of entry.
  - Optional remarks (e.g., reason for lateness).

### **6.3.5 Absence Request Review**

- Admins or instructors can view all submitted absence requests.
- Requests can be:
  - Approved or rejected.
  - Commented with justification.
  - Filtered by student, subject, or date

## **6.4 Business Rules**

The business rules define the core logic and constraints that govern the operation of the Smart Attendance System across both the mobile and web platforms. These rules ensure consistency, integrity, and security of the system's behavior while interacting with students, instructors, and administrators.

Below are the categorized business rules applied in the system:

### **6.4.1 Authentication & Authorization Rules**

1. Each user must register or be registered by an administrator before gaining access to the system.

2. Users must log in with valid credentials (email and password) to access system features.
3. Roles are defined as admin, instructor, and student, and permissions are granted accordingly:
  - Admins can manage users, subjects, departments, and sessions.
  - Instructors can only access their assigned subjects and related student attendance.
  - Students can only view and interact with their own data.

### **6.4.2 Student Attendance Rules**

1. A student can only attend sessions for subjects they are enrolled in.
2. A student cannot check in to a session before the session's start time.
3. A student arriving later than the defined grace period (e.g., 10 minutes) will be marked as Late.
4. If the student does not attend the session, they are automatically marked as Absent.
5. Attendance is recorded per session instance (date, time, subject, instructor).
6. Each attendance record is unique per student and per session (no duplicate entries).

### **6.4.3 Session & Schedule Rules**

1. Sessions are predefined by the administrator or instructor with:
  - Subject
  - Instructor
  - Location
  - Weekly recurrence
  - Duration and period index (e.g., 1st period)
2. Sessions are linked to multiple dates automatically based on start date and semester duration.
3. A session cannot be assigned to more than one instructor at the same time slot.
4. Instructors can only view and manage attendance for sessions they are assigned to teach.

#### **6.4.4 Absence & Request Rules**

1. A student can submit an absence request before the session date.
2. A student cannot submit an absence request for past dates.
3. An absence request must include:
  - Subject
  - Date range
  - Reason
  - Optional attachment (e.g., medical report)
4. Admins or instructors can approve or reject absence requests.
5. Approved absence requests may override automatic Absent status to Excused.

#### **6.4.5 Notification Rules**

1. Students receive notifications for:
  - Upcoming sessions (reminder with location and time)
  - Missed sessions
  - Late arrivals
  - Approved or rejected absence requests
2. Notifications are sent in real-time or scheduled via backend triggers.

#### **6.4.6 Reporting Rules**

1. Students can view attendance reports per subject, including:
  - Total sessions
  - Number of attendances, lateness, and absences
  - Session-wise breakdown (date, status, remarks)
2. Instructors can view aggregated reports per subject showing:
  - Attendance rate per student
  - Monthly summary
  - Session-wise data



3. Admins can generate institutional-wide reports and filter by:

- Department
- Grade
- Instructor
- Subject

## 6.5 Activity Diagram

### 1. Mobile Application (Student Role)

The activity diagram for the mobile app outlines the flow of how a student interacts with the system:

- Login Process:
  - The app is launched, checks if a user is logged in.
  - If not, the user is redirected to the login screen to enter credentials.
  - If the credentials are valid, the user is navigated to the home screen; otherwise, an error is shown.
- Subject and Attendance Report:
  - The student can view subjects, select one, and navigate to the attendance report screen.
  - From there, the student can access the absence request form to submit a leave request.
- Absence Request:
  - The form includes fields for reasons and date ranges.
  - On submission, if successful, a confirmation is shown; otherwise, an error is displayed.
- Notifications:
  - Students can navigate to the notifications screen to view system-generated alerts (e.g., session reminders, lateness, missed attendance).
- Profile Management:
  - Users can update their profile details, which are validated before saving.
  - Feedback is given for success or failure of the update.
- Logout Flow:

- Users can log out, which clears the session and redirects to the login screen.

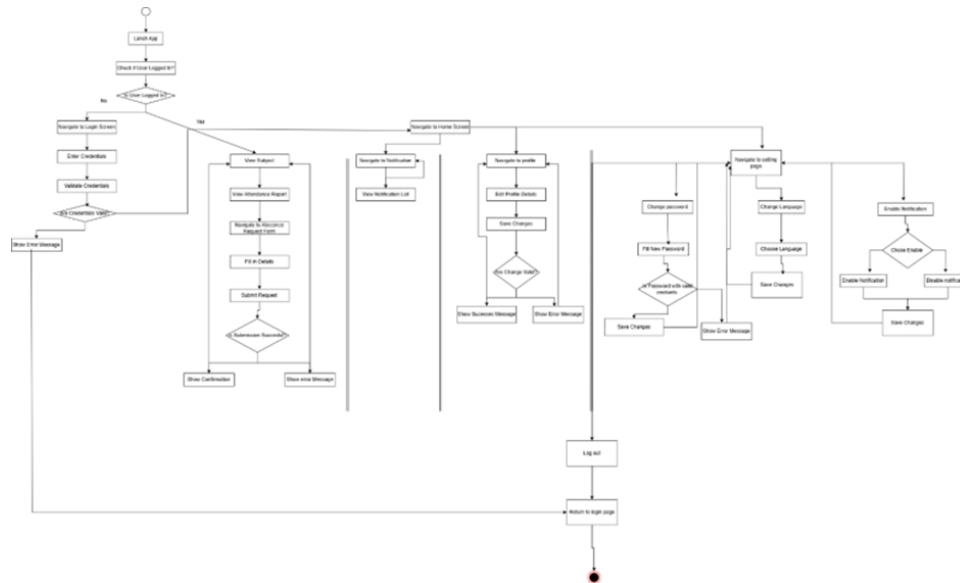


Figure 6-2 Mobile App activity diagram

## 2. Web Application (Admin/Instructor Role)

This activity diagram represents how an instructor or admin manages student data, attendance, and leave requests through the dashboard:

- Login Flow:
  - Admin/instructor opens the login page and submits credentials.
  - If validation succeeds, access is granted to the dashboard.
- Dashboard and Analytics:
  - Admin views attendance statistics, opens the student list, and selects a profile to see detailed reports.
- Messaging and Leave Management:
  - Admin can navigate to the messaging module to review leave request notifications.
  - From the leave applications section, requests can be approved or rejected with a single click.
- Status Transitions:
  - Requests update their status accordingly: either Approved or Rejected, which is then reflected in the student's profile and notification log.

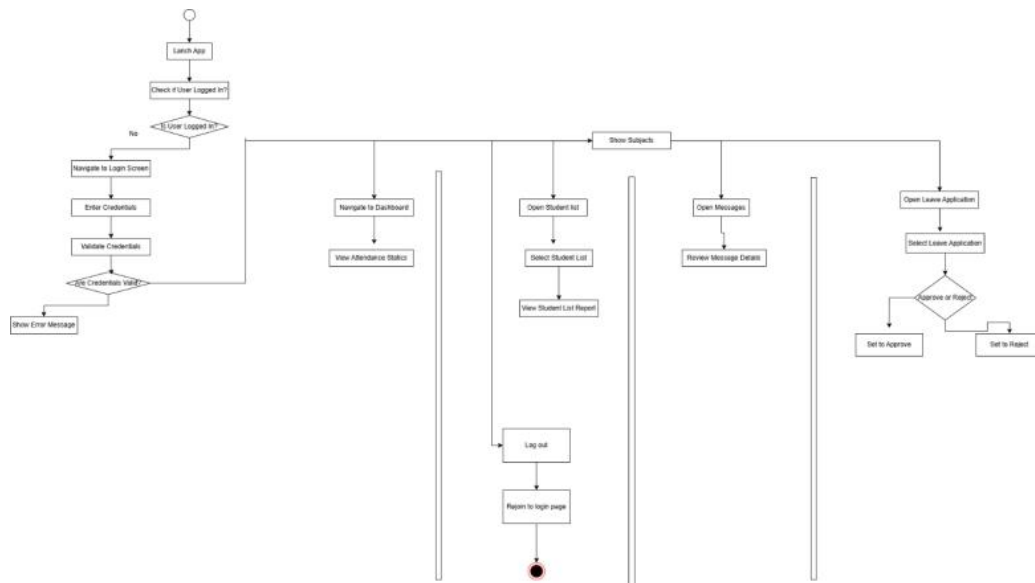


Figure 6-3 Web App activity diagram

## 6.6 Use Case Diagrams

Use case diagrams model the functional behavior of a system from the user's perspective. They illustrate the interactions between external actors (users) and the system to achieve a goal. Below are the detailed use cases for both the mobile and web interfaces of the Smart Attendance System.

### 1. Flutter Application Use Case Diagram (Student)

This case focuses on the operations available to students through the mobile app:

Primary Actor: Student

Use Cases:

- Login: The student provides email and password to access the app.
- Logout: Ends the current session and returns to the login screen.
- View Attendance Report:
  - Includes: *Fetch Attendance Data* from the API.
- View Subjects: Retrieves a list of enrolled subjects.
- Submit Attendance Request:
  - Extends: *Upload File* – the student may attach documents (e.g., leave note).
- Edit Profile:

- Extends: *Change Password* – optional feature inside profile.
- View Notifications: Displays real-time alerts (e.g., absence, lateness, upcoming sessions).

External System Interactions:

- API handles all requests (fetch subjects, upload files, notifications, etc.).
- The database stores persistent student data and attendance records.

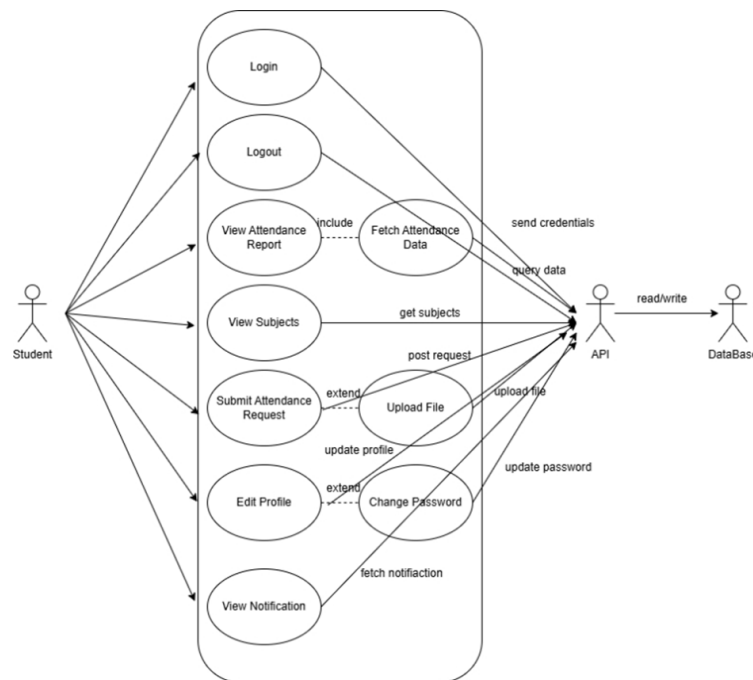


Figure 6-4 Flutter Application Use Case Diagram

## 2. Web Application Use Case Diagram (Instructor)

This diagram reflects the teacher's or instructor's workflow in the web dashboard:

Primary Actor: Teacher

Use Cases:

- Login / Logout: Authenticate and manage access.
- View Dashboard Summary: Quick statistics on attendance.
- View Student Profiles: Browse through enrolled students.
  - Extends: *Send Message to Student* – optionally notify students.

- View Student Attendance:
  - Includes: *Fetch Attendance Records* – retrieve from backend.
- Approve / Reject Absence: Review leave requests and update their status.
- View Messages: Notifications or leave applications from students.
- View Reports:
  - Extends: *Export Report* – download detailed attendance reports.

#### External System Interactions:

- The API communicates with the AI model (face recognition insights) and database (attendance logs, student profiles).
- AI feedback may influence auto-marking of attendance.

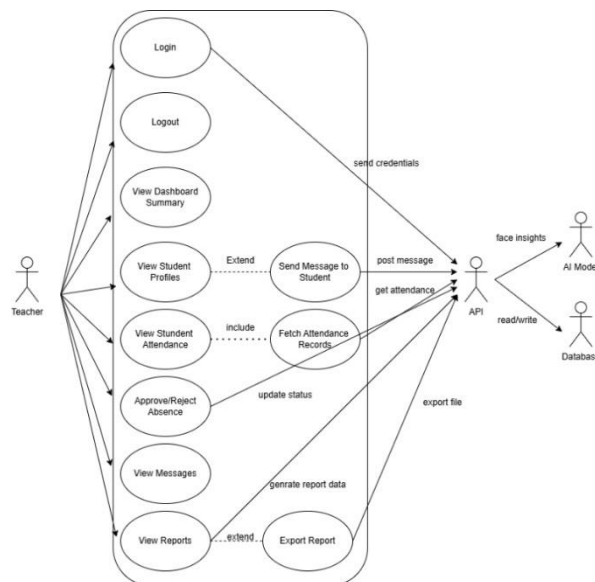


Figure 6-5 Web Application Use Case Diagram

## 6.7 Frontend

### 6.7.1 Definition

The frontend of a software system refers to the user interface (UI) and the user experience (UX) that users directly interact with. It is the visual and interactive part of the application that communicates with the backend (server or database) to display content, accept user input, and provide feedback.

In web or mobile applications, the frontend includes everything from:

- Layouts and components (buttons, forms, tables)
- Navigation systems (menus, drawers, tabs)
- Input validation and interactivity
- Displaying real-time data from backend systems

### **6.7.2 Purpose and Importance of Frontend**

The frontend is crucial because it:

- Bridges the gap between user and system: It translates backend logic and data into a usable and visual format.
- Defines the user experience: A smooth, intuitive, and responsive UI improves user satisfaction.
- Validates input before submission: Reduces server load and provides real-time feedback to the user.
- Promotes usability and accessibility: Ensures the system is accessible across different devices, screen sizes, and for users with disabilities.

### **6.7.3 Why Use a Frontend in This Project?**

In the Smart Attendance System, the frontend is essential for both the mobile app (students) and web portal (instructors/admins) to:

- View attendance reports visually
- Interact with system features like absence requests, notifications, profiles
- Allow real-time interaction with backend data (e.g., fetch subjects, submit forms)
- Provide a modern, professional user experience for multiple roles (student, teacher, admin)

Without a frontend:

- Users would have no visual access to the system
- Manual interactions (like calling the backend with Postman) would be required
- The system would be technically functional but practically unusable for real users

## **6.8 Mobile Frontend Using Flutter**

### **6.8.1 What is Flutter?**

Flutter is an open-source UI software development framework created by Google. It allows developers to build cross-platform applications using a single codebase — meaning the same code can run on Android, iOS, Web, and desktop platforms.

Flutter uses the Dart programming language and provides:

- A rich widget library for fast, beautiful UI design.
- Hot reload for fast development and testing.
- Native performance with direct compilation to machine code.

### **6.8.2 Why Flutter Was Chosen**

In the context of the Smart Attendance System, Flutter was selected for mobile development because of the following reasons:

#### **1. Cross-Platform Capability**

We wanted the app to be available for both Android and potentially iOS in the future without rewriting the UI twice. Flutter supports this with a single codebase.

#### **2. Modern UI/UX**

Flutter provides a flexible widget-based architecture which helped in building:

- Custom attendance report views
- Reusable components like bottom navigation bars, profile forms, and subject cards

#### **3. Speed of Development**

Using Flutter's hot reload feature, we could test UI instantly, which sped up the development process — especially when building forms, interactive charts, and live feedback UIs.

## 4. Integration with REST APIs

Flutter provides powerful HTTP packages (like `http`) to easily communicate with the Django backend API. We used it for:

- Logging in students
- Fetching attendance records
- Submitting absence requests
- Viewing notifications and profiles

## 5. Community and Plugin Ecosystem

With a large ecosystem, Flutter supports packages like:

- `provider` or `get` for state management
  - `image_picker`, `charts_flutter`, and more for interactive features
- This reduced the need to build features from scratch.



*Figure 6-6 Flutter*

## 6.9 What is Web Frontend?

The Web Frontend is the visible part of the system that users interact with through their web browser. It includes:

- Visual layout (pages, buttons, forms)
- Navigation and interactivity
- Styling and responsiveness for different screen sizes

Frontend development is essential to deliver a user-friendly interface that connects with the backend services (APIs) to present and manage data dynamically.



## 1. HTML + Tailwind Simplicity

HTML provides a lightweight structure, while Tailwind gives full control over design using utility-first classes, making the UI responsive, modern, and fast to build.

## 2. Consistency Across Screens

Using Tailwind CSS, we ensured all pages followed a clean design system with consistent:

- Button styles
- Table layouts
- Colors and spacing
- Responsive layout for tablets and desktops

## 3. Integration with Django

Django's template engine allowed us to:

- Dynamically render subjects, attendance reports, notifications
- Use logic (like `{% if %}`, `{% for %}`) inside HTML for real-time data display
- Easily plug into the Django backend and session management

## 4. Faster Design Prototyping

Tailwind CSS's ready-to-use utility classes helped speed up development of:

- Dashboards
- Student profiles
- Attendance tables
- Leave approval modals



### Advantages of This Stack

*Figure 6-7 HTML + CSS*

- Performance: Static HTML + Tailwind renders fast with low memory usage.
- SEO-Friendly: Django templates render server-side, ensuring crawlable content.
- Customizability: Utility-based Tailwind allows custom design without third-party overhead.

## 6.10 Figma Design and Importance of UI/UX

In our Smart Attendance System, UI/UX design played a vital role in ensuring a smooth and intuitive user experience. We used Figma as the primary tool for creating interactive and collaborative interface designs before any development began.

Figma allowed us to:

- Visualize the layout and flow of both mobile and web screens.
- Ensure consistent visual language across platforms.
- Collaborate easily between designers and developers.
- Gather feedback quickly and iterate on designs efficiently.

Good UI/UX design enhances usability, reduces user confusion, and increases satisfaction, especially for systems that serve students, instructors, and admins daily.

### 6.10.1 Web Application (Admin/Instructor Role) Design

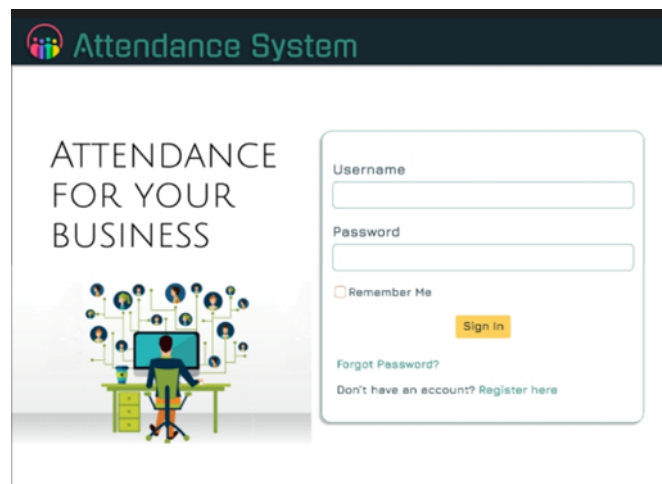
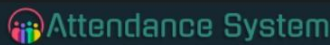


Figure 6-8 Login page



## FORGOT PASSWORD

Username

New Password

Confirm Password

Submit

Figure 6-9 Forget password page



Figure 6-10 Password reset Page

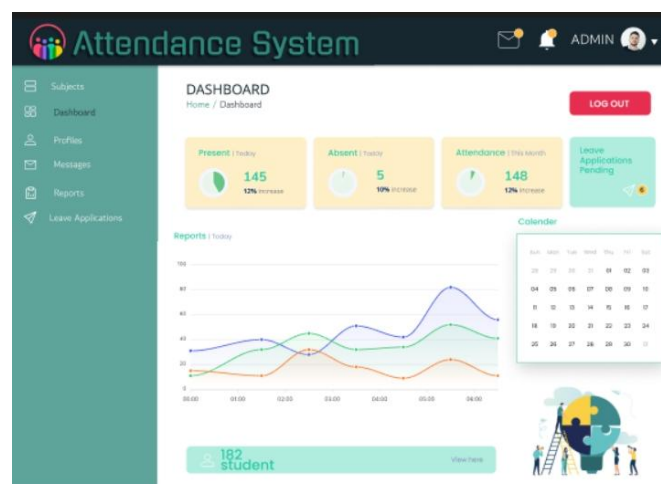


Figure 6-11 Dashboard page

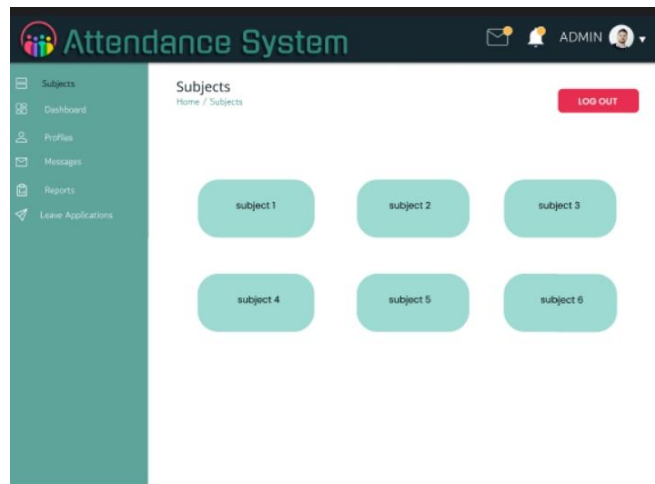


Figure 6-12 Subjects page

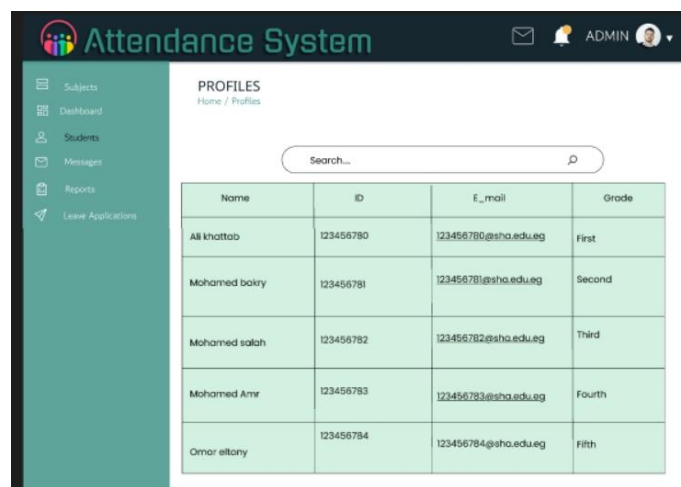


Figure 6-13 Profile page

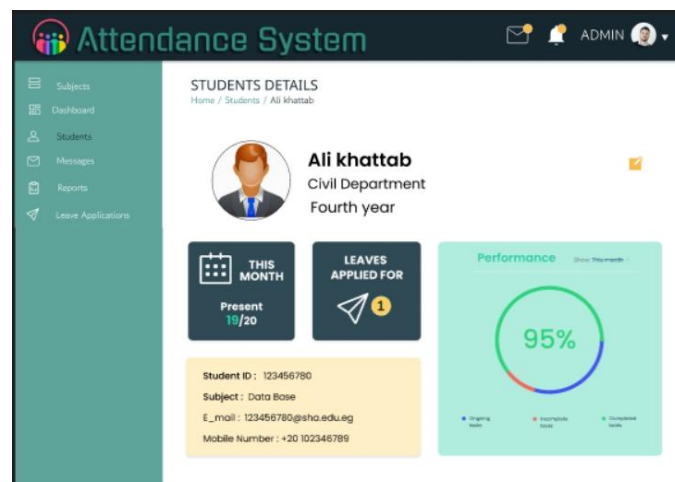


Figure 6-14 Student details page

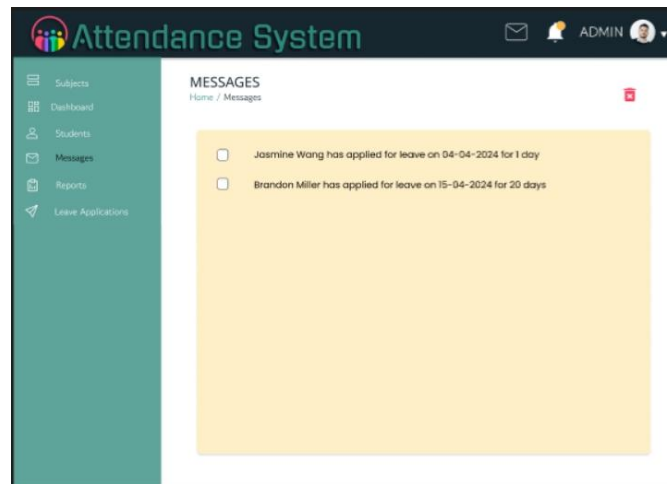


Figure 6-15 Message page

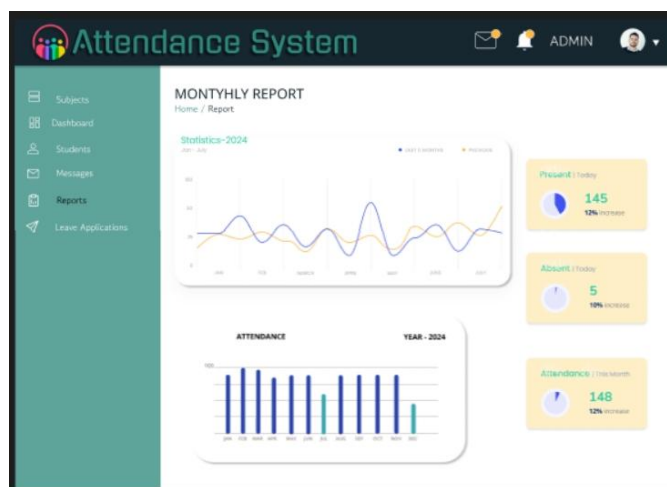


Figure 6-16 Students monthly report

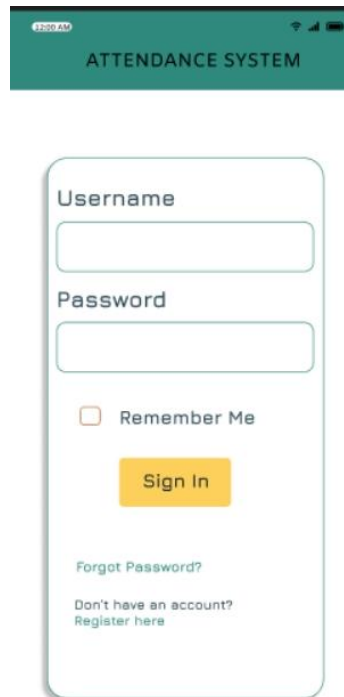
The screenshot shows the 'LEAVE APPLICATIONS' page of the Attendance System. The left sidebar contains links to Subjects, Dashboard, Students, Messages, Reports, and Leave Applications. The main content area displays a table of leave applications:

DATE APPLIED	NAME	ABSENCE REASON	DURATION	STATUS
14-02-2024	Jasmine Wang	Sick	1	REJECTED
14-02-2024	Tooshu Jay	Sick	1	REJECTED
28-02			5	APPROVED
01-04			1	APPROVED
04-04			1	PENDING
06-04			28	PENDING

A modal dialog titled 'LEAVE APPROVAL' is open, showing 'REJECT' and 'APPROVE' buttons.

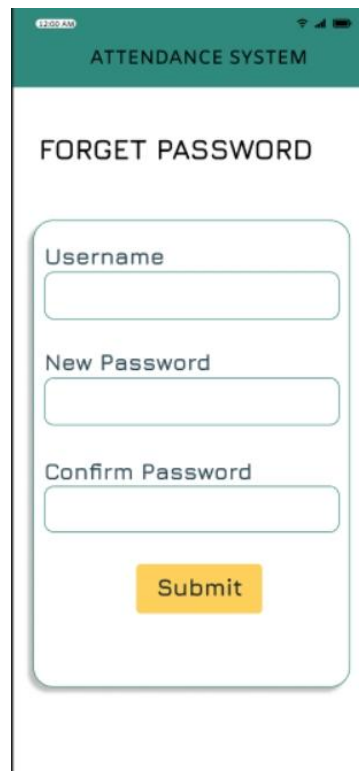
Figure 6-17 Leave App

### 6.10.2 Mobile Application (Student Role)



The image shows a mobile application login screen. At the top, there is a green header bar with the text "ATTENDANCE SYSTEM" in white. Below the header, the screen is white. There are two input fields: "Username" and "Password". Below the "Password" field, there is a checkbox labeled "Remember Me". Below the checkbox, there is a yellow button labeled "Sign In". At the bottom, there are two links: "Forgot Password?" and "Don't have an account? Register here".

Figure 6-18 Mobile app login page



The image shows a mobile application "FORGET PASSWORD" screen. At the top, there is a green header bar with the text "ATTENDANCE SYSTEM" in white. Below the header, the screen is white. There is a title "FORGET PASSWORD" in bold. Below the title, there are three input fields: "Username", "New Password", and "Confirm Password". Below the "Confirm Password" field, there is a yellow button labeled "Submit".

Figure 6-19 Mobile app forgets password page

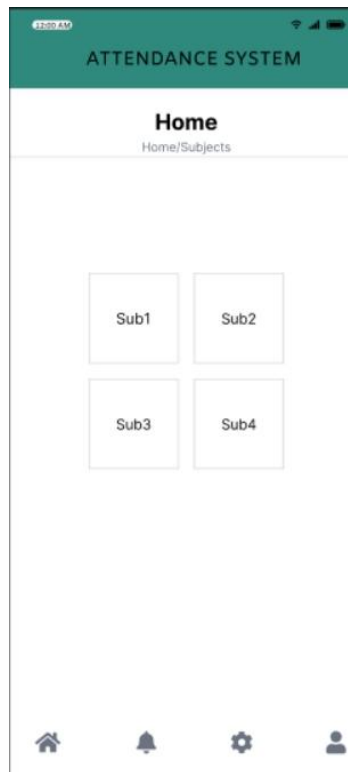


Figure 6-20 Mobile app home page

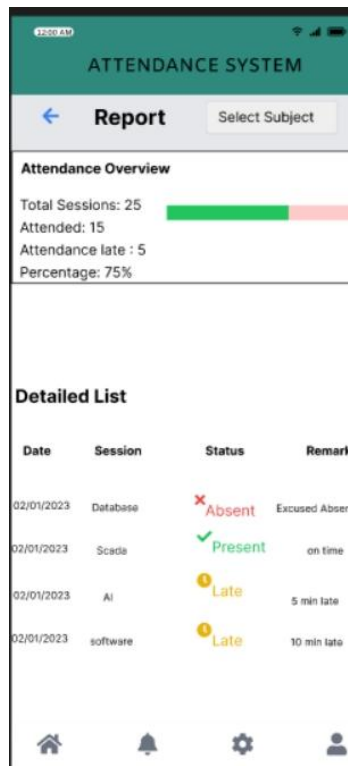


Figure 6-21 Mobile app attendance report page

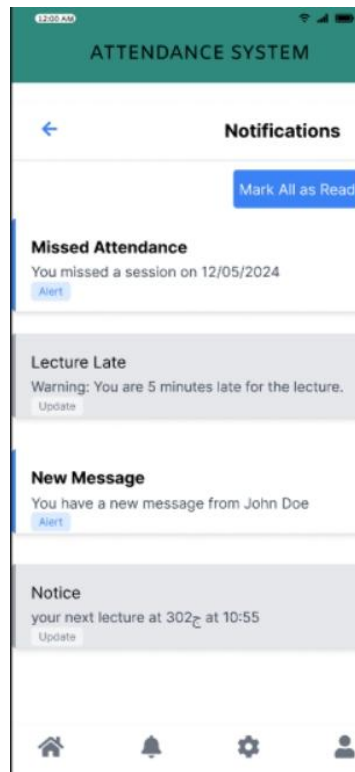


Figure 6-22 Mobile app notification page

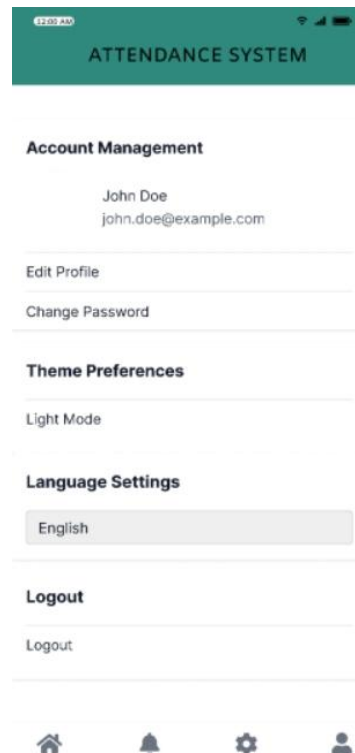


Figure 6-23 Mobile app account management page



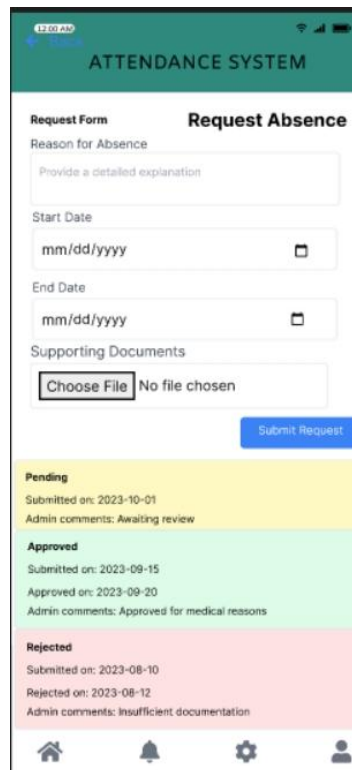


Figure 6-24 Mobile app request absence

## 6.11 Implementation of UI/UX Design (Mobile + Web)

After completing the UI/UX design using Figma, we translated the visual designs into functional interfaces within both the mobile and web platforms. The goal was to preserve consistency, improve usability, and maintain alignment with the system's branding and structure as envisioned in the design phase.

### 1. Mobile Implementation (Flutter)

The mobile application was developed using Flutter, a cross-platform UI toolkit. We implemented the screens by:

- Mapping each Figma frame to a corresponding Flutter screen.
- Utilizing core widgets such as Scaffold, AppBar, ListView, and Card to structure the UI.
- Maintaining spacing, typography, and colors based on the Figma specs using Flutter's ThemeData and custom TextStyle.
- Adding custom widgets for reusability (e.g., SubjectCard, BottomNavBar, SubHeaderBar) to match the design components.

This implementation allowed for a fast, responsive, and consistent user interface across different devices and screen sizes.

## **2.Web Implementation (HTML + Tailwind CSS)**

For the web frontend, we relied on HTML5 for semantic structure and Tailwind CSS for styling. The web UI was built by:

- Structuring the layout using HTML tags like div, section, and form, reflecting the visual grouping shown in the design.
- Using Tailwind utility classes to apply paddings, margins, font sizes, borders, and colors as defined in Figma.
- Ensuring responsiveness across screen sizes using Tailwind's breakpoint classes (sm:, md:, lg:).
- Creating a consistent user interface that visually matches the mobile design while being optimized for desktop interaction.

### **6.11.2 Why This Implementation Matters**

- It ensures the user experience remains unified across both platforms.
- The design-to-code transition reduces rework and confusion between designers and developers.
- It enhances user engagement and satisfaction, making the system accessible and user-friendly for students, instructors, and admins alike.

## Chapter 7 : System Results

This chapter describes the full operational flow of the Smart Attendance System, starting from camera activation to the final presentation of student attendance data. Each phase in the system is handled by a specific layer—AI detection, backend logic, and frontend visualization—ensuring accurate, real-time attendance tracking. The following are the main execution stages:

### 7.1 Camera Activation and Frame Capture

As the class session begins, the external camera is triggered to start capturing live video frames of the lecture hall. This image feed serves as the input for the AI system to detect and recognize students in the room. The frames are processed at periodic intervals within the valid class time window.

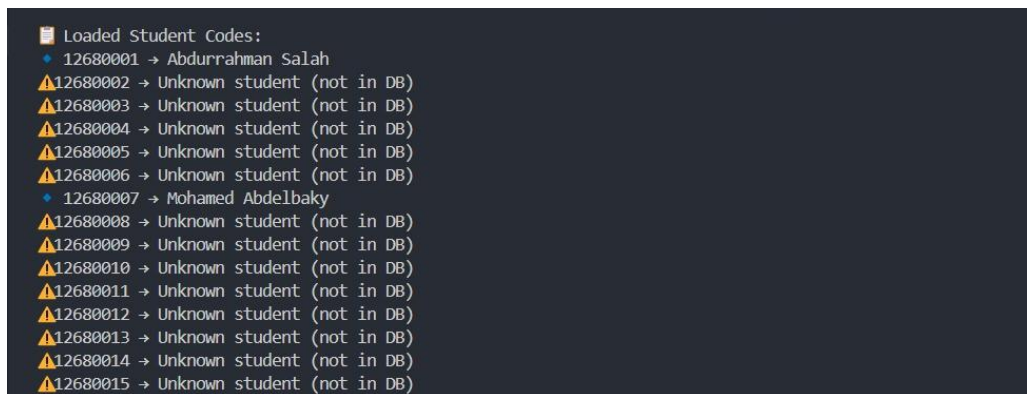


*Figure 7-1 camera captures image*

## 7.3 Face Recognition using FaceNet Embedding

Each cropped face is passed to the Face Recognition module, which uses a pre-trained FaceNet model. The model compares each detected face against the known training dataset (students' faces), and if a match is found, it returns:

- \* The student's unique code (student\_code)



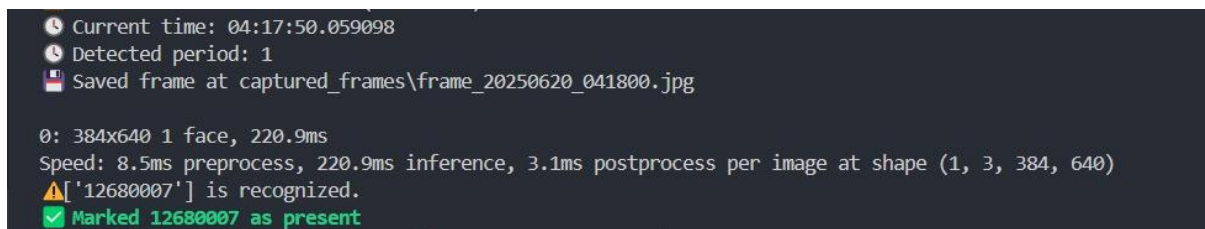
```
Loaded Student Codes:
• 12680001 → Abdurrahman Salah
⚠ 12680002 → Unknown student (not in DB)
⚠ 12680003 → Unknown student (not in DB)
⚠ 12680004 → Unknown student (not in DB)
⚠ 12680005 → Unknown student (not in DB)
⚠ 12680006 → Unknown student (not in DB)
• 12680007 → Mohamed Abdelbaky
⚠ 12680008 → Unknown student (not in DB)
⚠ 12680009 → Unknown student (not in DB)
⚠ 12680010 → Unknown student (not in DB)
⚠ 12680011 → Unknown student (not in DB)
⚠ 12680012 → Unknown student (not in DB)
⚠ 12680013 → Unknown student (not in DB)
⚠ 12680014 → Unknown student (not in DB)
⚠ 12680015 → Unknown student (not in DB)
```

Figure 7-2 load students codes

## 7.4 Session Matching (Date, Time & Subject Validation)

Before marking attendance, the system must verify that the current recognition attempt falls under a valid Class Session. This is done by checking:

- \* If the current session\_date matches today's date
- \* If the current time fits within a known ClassPeriod
- \* If the student is enrolled in the subject/session linked to that slot



```
🕒 Current time: 04:17:50.059098
🕒 Detected period: 1
📁 Saved frame at captured_frames\frame_20250620_041800.jpg

0: 384x640 1 face, 220.9ms
Speed: 8.5ms preprocess, 220.9ms inference, 3.1ms postprocess per image at shape (1, 3, 384, 640)
⚠ ['12680007'] is recognized.
✅ Marked 12680007 as present
```

Figure 7-3 matches session date and time and recognize students

## 7.5 Attendance Recording in the Database

Once the student identity and session are verified, the backend accesses the Attendance table.

If the student's record is found with status pending for the session, it is updated to present.

```
'id': 59,  
'student': {  
  "account": {  
    "id": 7,  
    "name": "Mohamed Abdelbaky",  
    "email": "m.a@stu.com",  
    "mobile_number": "01023795695",  
    "role": "student",  
    "last_login": "2025-06-19T16:15:34.824710Z",  
    "is_active": true,  
    "is_verified": true  
  },  
  "student_code": "12680007",  
  "date_of_birth": "2001-05-01",  
  "gender": "Male",  
  "address": "",  
  "enrollment_date": "2025-06-19",  
  "department": 1,  
  "grade": "1"
```

```
  "status": "present",  
  "check_in_time": "2025-06-20T01:18:00.903100Z",  
  "remarks": null  
}
```

Figure 7-4 database change in the status

## 7.6 Frontend Display on Mobile and Web

After successful marking, the attendance data is made available to both:

The mobile application (Flutter with SQLite)

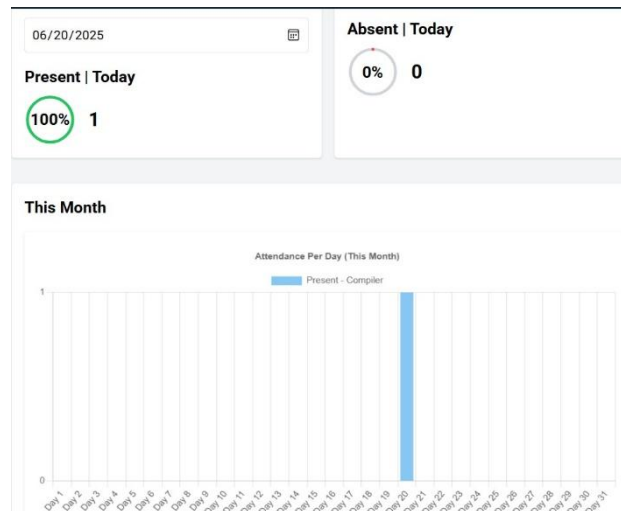


Figure 7-5 instructor web app display

The web dashboard (Tailwind/Django Admin or custom UI

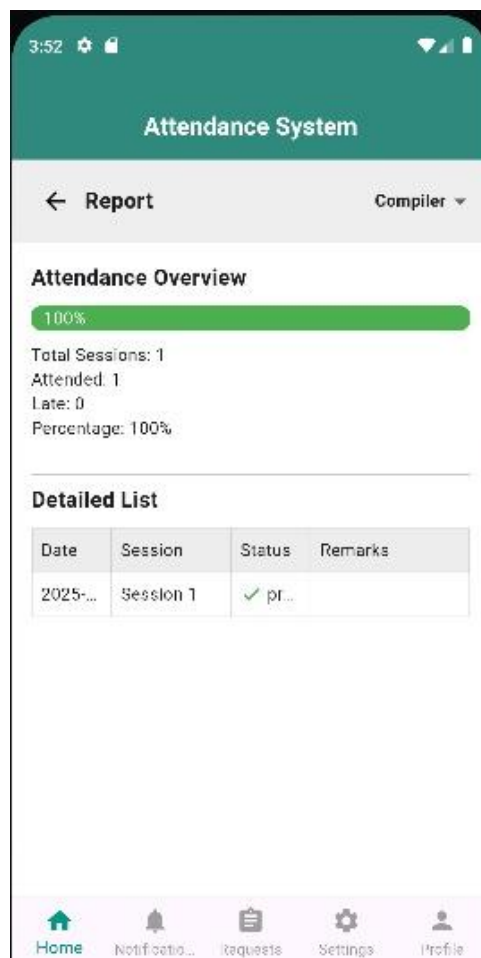


Figure 7-6 student mobile app display

## Conclusion

The Smart Attendance System is a comprehensive solution that combines artificial intelligence, web technologies, and mobile integration to revolutionize the way student attendance is managed. The backend, built using Django and PostgreSQL, provides a secure and scalable environment for data processing, session management, and API communication. The frontend (admin/staff web interface) allows for real-time attendance monitoring, leave management, and academic session tracking.

On the other hand, the mobile application enables students to stay updated on their attendance status, submit leave requests, and receive notifications. At the core of the system is the AI-based camera module, which performs face detection and recognition to automatically mark attendance—removing the need for traditional manual methods.

The system is designed following clean architecture and modular design patterns to ensure maintainability, extensibility, and ease of future upgrades. Each component, from database schema to RESTful APIs to mobile interface, works in harmony to deliver a seamless and efficient attendance experience for both administrators and students.

## Future Work

To further improve and expand the Smart Attendance System, the following areas can be considered for future development:

### Backend Enhancements:

- **Microservices Architecture:** Splitting AI recognition, attendance, and user management into separate services for better scalability and fault isolation.
- **Advanced Caching and Performance:** Using Redis or Memcached to optimize frequently accessed data like academic schedules and user roles.
- **Audit Logs & Monitoring:** Implementing full audit trails for all system actions and integrating tools like Prometheus + Grafana for real-time monitoring.

**Mobile Application Expansion:**

- Offline Mode Support: Allowing students to view attendance history and submit requests even without internet, syncing later when connected.
- QR Code Check-in (optional feature): As a fallback or manual alternative, supporting QR-based attendance for special cases.

**AI Module Extensions:**

- Multi-camera Support: Handling larger rooms by connecting multiple IP cameras to one central recognition service.
- False Detection Prevention: Using more robust algorithms or multi-frame confirmation to reduce false positives in face detection.
- Continuous Learning: Integrating a retraining mechanism to improve face recognition accuracy over time with real usage



## References

- P. Viola and M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001*, 2001, pp. I-511–I-518. doi: 10.1109/CVPR.2001.990517.
- N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, pp. 886–893. doi: 10.1109/CVPR.2005.177.
- N. Jegham, C. Y. Koh, M. Abdelatti, and A. Hendawi, "YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions," *arXiv preprint arXiv:2411.00201*, 2024. doi: 10.48550/arXiv.2411.00201.
- <https://docs.ultralytics.com/models/yolo11/#overview>, (October 10,2024).
- M. A. Turk and A. P. Pentland, "Face Recognition using Eigenfaces," in *Proceedings of the 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Maui, HI, USA, 1991, pp. 586–591. doi: 10.1109/CVPR.1991.139758.
- P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 711–720, 1997. doi: 10.1109/TPAMI.1997.554220.
- T. Ahonen, A. Hadid, and M. Pietikäinen, "Face Recognition With Local Binary Patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 2037–2041, 2006. doi: 10.1109/TPAMI.2006.24.
- Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 1701–1708. doi: 10.1109/CVPR.2014.220.
- F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 815–823. doi: 10.1109/CVPR.2015.7298682.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

Larman, C. (2002). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall.

McLaughlin, B., Pollice, G., & West, D. (2006). *Head First Object-Oriented Analysis and Design*. O'Reilly Media.

Sierra, K., & Bates, B. (2005). *Head First Java* (2nd ed.). O'Reilly Media.

## شكر وتقدير

لإنجاز أبسط الأمور، نحتاج إلى عون الله تعالى، ومشروعنا هذا بما فيه من تعقيدات وصعوبات لم يكن مهمة سهلة. وبعد الانتهاء من تنفيذه، نتوجه بالشكر لله سبحانه وتعالى، القوي القدير، الذي أعاننا في كل خطوة، ولم يدع عزيمةتنا تضعف حتى في مواجهة أصعب المشكلات.

ونود أن نعبر عن شكرنا العميق لمشرفيتنا في المشروع، الدكتورة سحر كمال والمهندسة أميرة السيد، على دعمهما الكبير ومساعدتهما لنا طوال فترة المشروع، رغم أن ذلك استغرق الكثير من وقتهما الثمين. كما نود أن نشكر جميع أساتذتنا في الجامعة الذين قدموا لنا الدعم والإرشاد طوال سنوات الدراسة.

ولا ننسى أن نعبر عن امتناننا العميق لعائلاتنا التي كانت دوماً سنداً لنا في كل خطوة، والتي ستظل دائماً بجانبنا.