**INTRODUCTION**

FaceAuth 2.0 – Student Attendance System is an advanced AI-driven application designed to automate attendance recording in educational institutions using secure face recognition and liveness detection technologies. Traditional attendance systems—such as manual calling, ID cards, barcodes, or fingerprint biometrics—often face issues such as inaccuracies, proxy attendance, time consumption, hygiene concerns, and administrative inefficiency.

With the rapid advancement of Machine Learning (ML) and Computer Vision, the use of face recognition for secure attendance monitoring has become a viable and essential solution for modern institutions.

FaceAuth 2.0 integrates DeepFace, OpenCV, ML-based liveness detection, and auto-model training to ensure real-time, highly accurate, and spoof-proof verification. The system provides role-based interfaces for Admin, Faculty, and Students, implemented through a Flutter mobile application and supported by a Flask backend connected to Firebase Authentication and a secure database.

Through automated recognition, centralized reporting, and ERP compatibility, FaceAuth 2.0 significantly improves institutional efficiency, minimizes human intervention, and enhances transparency and accountability. The system also offers an intuitive UI/UX, real-time notifications, secure cloud integration, and AI-based identity validation**,** making it a reliable solution for smart educational campuses.

Technology and Field: This project lies at the intersection of Artificial Intelligence, Computer Vision, and Educational Technology (EdTech).

**It leverages:**

1. Python + Flask for backend logic and API services
2. Flutter for the mobile app frontend
3. SQLite / MySQL / Firebase(Cloud Storage) for database management
4. OpenCV + Face Recognition libraries (dlib/DeepFace/liveness\_ml\_model) for model training and real-time verification

FaceAuth 2.0 isn’t just another attendance app; it’s a move toward smarter campuses and workplaces — contactless, fast, and secure.

**OBJECTIVES**

The objectives of the **FaceAuth 2.0 – Student Attendance System** are divided into **General Objectives** and **Specific Objectives** to clearly define the purpose, direction, and expected outcomes of the project.

**1. General Objectives**

1. **To develop an efficient, secure, and automated attendance system** using advanced face recognition technology.
2. **To eliminate proxy attendance and manual errors** through AI-based verification and liveness detection.
3. **To streamline the attendance process** across departments, faculty, and students within the institution.
4. **To create a fully digital, contactless, and user-friendly solution** for modern educational environments.
5. **To integrate AI and ML technologies** into administrative processes for smart campus management.

**2. Specific Objectives**

**Admin-Level Objectives**

* To enable the admin to add, edit, and manage faculty and student profiles.
* To enroll users with **multi-angle face capture** and automatic **ML model training**.
* To manage departments, subjects, and academic configurations.
* To export and sync attendance data directly to institutional ERP systems.
* To maintain centralized data logs for analysis and reporting.

**Faculty-Level Objectives**

* To allow faculty to mark attendance effortlessly using real-time face recognition.
* To verify student identity using **DeepFace** and **Liveness Detection** during attendance sessions.
* To provide accurate attendance logs and allow faculty to export daily/weekly/monthly reports.
* To support correction/update of attendance records for their own subjects.

**Student-Level Objectives**

* To provide students with access to their **overall and subject-wise attendance percentage**.
* To allow viewing of daily attendance punch history for transparency.
* To send alerts when attendance falls below the threshold (e.g., 75%).

**Technical Objectives**

* To design a smooth, responsive **Flutter mobile interface** for all user roles.
* To build a secure backend using **Flask**, **DeepFace**, **OpenCV**, and **Liveness Detection Models**.
* To integrate **Firebase Authentication & Cloud Storage** for secure user and media data handling.
* To maintain data integrity and security through proper database management and validation.

**SCOPE & LIMITATIONS**

This section describes what the **FaceAuth 2.0 – Student Attendance System** covers (scope) and what is currently beyond its capabilities (limitations). It helps define realistic boundaries for the project.

**1. SCOPE OF THE PROJECT**

**A. Functional Scope**

1. **Automated Face Recognition Attendance**
   * Real-time detection and verification using DeepFace.
   * Anti-spoofing with liveness detection (blink/movement).
2. **Role-Based Access System**
   * Admin
   * Faculty
   * Student
3. **Admin Capabilities**
   * Add/Edit/Delete Faculty & Student Records.
   * Manage departments, subjects, and academic structure.
   * Enroll face data with auto-capture and ML training.
   * Activate/Deactivate user accounts.
   * Sync attendance with ERP.
   * Access all reports and logs.
4. **Faculty Capabilities**
   * Login via Face Authentication or Faculty ID credentials.
   * Start attendance session for assigned subjects.
   * Real-time student detection and verification.
   * Export Attendance (PDF/Excel).
   * Update/correct attendance for their subjects.
5. **Student Capabilities**
   * Login via Roll No + Password.
   * Access attendance percentage (overall/subject-wise).
   * View daily punch history.
   * Receive low-attendance alerts.
6. **Support & Help Features**
   * User manual guide.
   * Support request form linked to developer email.
   * About application and developer details.

**2. TECHNICAL SCOPE**

* **Frontend:** Flutter (cross-platform mobile app)
* **Backend:** Python Flask
* **AI/ML Models:** DeepFace, Haar Cascade/SSD, Liveness Detection
* **Database:** SQLite / Firebase
* **Authentication:** Firebase Authentication
* **Cloud Integration:** Firebase Storage
* **ERP Sync:** Exporting structured attendance data for institutional ERP
* **File Export:** PDF/Excel attendance reports
* **Notifications:** UI pop-ups & audio feedback

**3. ORGANIZATIONAL SCOPE**

* Suitable for:
  + Schools
  + Colleges
  + Universities
  + Coaching/Vocational institutes
* Supports multiple departments and subjects simultaneously.
* Can be extended to large-class environments with proper server scaling.

**4. LIMITATIONS**

**A. Technical Limitations**

1. **Requires good lighting conditions** for high-accuracy face recognition.
2. **Performance depends on device camera quality** and processing speed.
3. **High-quality training images required** for reliable identification.
4. **Real-time recognition may slow down on low-end devices**.
5. **Internet connectivity required** for:
   * Firebase Authentication
   * Cloud data syncing
   * ERP upload

**B. Functional Limitations**

1. Cannot detect face if:
   * Face is covered (mask/scarf/helmet).
   * User avoids camera angle.
2. Liveness detection may require **slight user movement**.
3. Admin must enroll users manually before system use.
4. Attendance marking works **only within faculty’s assigned subjects**.

**C. Future Enhancements Required**

1. Integration with **AI-based attendance forecasting**.
2. SMS / WhatsApp alerts for low attendance.
3. Timetable-based auto-session scheduling.
4. Integration with a full **ERP API** instead of file upload.

**LITERATURE REVIEW / EXISTING SYSTEMS REVIEW**

The Literature Review highlights existing research, technologies, and attendance systems previously developed in academic and industrial environments. It establishes the foundation for **FaceAuth 2.0** and explains how the project improves upon existing systems.

**1. INTRODUCTION**

Attendance management has traditionally relied on manual or semi-digital methods such as attendance registers, ID card scanning, barcode systems, and biometric fingerprint devices. With recent advancements in **Artificial Intelligence (AI)** and **Computer Vision**, face recognition-based attendance systems have become a practical, contactless, and secure alternative.

This review explores existing systems, methodologies, and technologies that form the basis for creating a more accurate and efficient solution like FaceAuth 2.0.

**2. EXISTING SYSTEMS REVIEW**

**2.1 Manual Attendance System**

**Description:**  
Faculty call out names or students sign attendance registers.

**Limitations:**

* Time-consuming
* High chances of proxy attendance
* Errors in recording
* Difficult to analyze and maintain large data

**2.2 RFID / Smart Card-Based Attendance Systems**

**Description:**  
Students swipe ID cards at a reader to register attendance.

**Limitations:**

* Cards can be exchanged → proxy possible
* Hardware maintenance required
* Loss or damage of cards is common
* Not truly automated

**2.3 Fingerprint Biometric Attendance**

**Description:** Widely used in many institutions for staff attendance.

**Limitations:**

* Requires physical contact → hygiene concerns
* Sensors wear out over time
* Not suitable for large classroom environments
* Can be bypassed using artificial fingerprints

**2.4 Basic Face-Recognition Attendance Systems**

**Description:** Camera captures face → software matches against database.

**Limitations:**

* No liveness detection → vulnerable to photos/videos
* Low accuracy in poor lighting
* Basic models fail with pose variations
* Limited scalability

**3. RELATED RESEARCH WORK**

**3.1 Deep Learning-Based Face Recognition (DeepFace, FaceNet, VGGFace)**

Studies highlight that deep neural networks outperform traditional feature-based methods (LBP, Eigenfaces, etc.) in:

* Accuracy
* Pose tolerance
* Speed
* Feature learning

DeepFace by Facebook Research achieves **97.35%+ accuracy**, making it reliable for authentication systems.

**3.2 Liveness Detection Techniques**

Research demonstrates the importance of anti-spoofing to prevent:

* Printed photo attacks
* Mobile screen replay attacks
* 3D mask attacks

Common methods:

* Eye blink detection
* Head movement detection
* Texture-based CNN classification

FaceAuth 2.0 integrates these techniques to prevent spoof attempts.

**3.3 Realtime Detection Systems Using OpenCV**

OpenCV-based systems are widely used for:

* Frame capture
* Face landmark detection
* Pre-processing  
  They provide fast frame handling for mobile and desktop environments.

**3.4 Cloud-Based Authentication (Firebase)**

Research supports cloud-based solutions for:

* Fast authentication
* Cross-platform access
* Secure credential management
* Scalable backend operations

This aligns well with FaceAuth 2.0’s architecture.

**4. GAP ANALYSIS (WHAT IS MISSING IN EXISTING SYSTEMS)**

|  |  |
| --- | --- |
| **Existing Systems** | **Problems Identified** |
| Manual System | Slow, error-prone, proxy attendance |
| RFID/ID Card | Easily misused, hardware cost |
| Fingerprint Bio | Contact-based, unhygienic, spoofable |
| Basic Face Recog | No liveness, low accuracy, poor training |
| ERP Attendance | No automated capture, relies on human input |

**FaceAuth 2.0 fills these gaps by providing:**

* DeepFace-based Recognition
* Auto ML Training (2–3 iterations)
* Liveness Detection (blink/movement)
* Admin–Faculty–Student role-based system
* PDF/Excel export capability
* ERP sync ready
* Firebase authentication & cloud storage
* Fully mobile-based, no specialized hardware

**5. CONCLUSION OF LITERATURE REVIEW**

The reviewed literature and existing attendance technologies show the need for a **more secure, intelligent, efficient, and automated** system.

**FaceAuth 2.0** positions itself as an improved solution utilizing advanced deep learning techniques, real-time image processing, and mobile-first architecture, making it suitable for educational institutions aiming for digital transformation.

**SYSTEM REQUIREMENTS SPECIFICATION (SRS)**

This section defines the **Hardware** and **Software** requirements essential for developing, deploying, and running **FaceAuth 2.0 – Student Attendance System** smoothly and efficiently.

**1. HARDWARE REQUIREMENTS**

**A. For Development Environment**

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum Requirement** | **Recommended** |
| **Processor** | Intel i3 7th Gen or AMD Equivalent | Intel i5/i7 or AMD Ryzen 5+ |
| **RAM** | 4 GB | 8–16 GB |
| **Storage** | 40 GB free space | SSD 256 GB+ |
| **Graphics** | Integrated Graphics | GPU support (NVIDIA) for faster ML training |
| **Camera** | 720p HD webcam | 1080p HD camera or mobile camera (8MP+) |
| **Network** | Basic Wi-Fi | Stable high-speed connection (for Firebase/ERP sync) |

**B. For Mobile Devices (Flutter App)**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Minimum** | **Recommended** |
| **Operating System** | Android 8.0+, iOS 12+ | Android 10+, iOS 14+ |
| **RAM** | 2 GB | 4+ GB |
| **Camera** | 5 MP | 8–12 MP with good low-light performance |
| **Storage** | 200 MB free | 500 MB free |

**C. For Server Deployment (optional if hosting Flask API)**

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum** | **Recommended** |
| **CPU** | 1 vCPU | 2–4 vCPUs |
| **RAM** | 1 GB | 4 GB |
| **Storage** | 10 GB SSD | 20+ GB SSD |
| **Cloud Provider** | Any basic VM | AWS / GCP / PythonAnywhere / Render |

**2. SOFTWARE REQUIREMENTS**

**A. Development Tools**

|  |  |  |
| --- | --- | --- |
| **Software** | **Version** | **Purpose** |
| **Python** | 3.10+ | Backend (Flask Server) |
| **Flask Framework** | Latest | API development |
| **Flutter SDK** | 3.x+ | UI/Frontend |
| **Android Studio / VS Code** | Latest | App development |
| **Firebase Console** | Active project | Auth & Storage |
| **SQLite / MySQL** | 3.x / 8.x | Database |
| **Figma** | Latest | UI/UX Design |
| **Git & GitHub** | Latest | Version control |

**B. ML & Computer Vision Libraries**

|  |  |
| --- | --- |
| **Library / Tool** | **Purpose** |
| **OpenCV** | Face detection, image processing |
| **DeepFace** | Face recognition & embedding |
| **TensorFlow / PyTorch** | Optional ML enhancement |
| **NumPy, Pandas** | Data handling |
| **scikit-learn** | ML preprocessing/training |
| **Liveness Detection Plugin (Flutter)** | Anti-spoofing detection |
| **MediaPipe (optional)** | Landmark detection |

**C. Mobile App Dependencies**

|  |  |
| --- | --- |
| **Package** | **Purpose** |
| flutter\_liveness\_detection\_randomized\_plugin | Liveness detection |
| camera | Live camera stream |
| file\_picker | Export files |
| pdf & excel packages | Report generation |
| firebase\_auth | Login/Authentication |
| firebase\_storage | Store face images |
| cloud\_firestore (optional) | Cloud DB |

**D. Operating System Requirements**

|  |  |
| --- | --- |
| **Component** | **Requirement** |
| Development OS | Windows 10+, macOS 11+, Ubuntu 18+ |
| Mobile OS | Android / iOS |
| Server OS | Linux recommended |

**3. OTHER REQUIREMENTS**

**A. Network Requirements**

* Stable internet required for:
  + Firebase Authentication
  + Cloud Storage uploads
  + ERP upload
* Offline mode possible only for temporary storage, not for recognition.

**B. Security Requirements**

* Strong password policy
* Face data encryption
* Authentication using Firebase
* Secure REST APIs (HTTPS)
* Role-based access control

**C. Performance Requirements**

* Real-time face detection (<1 second capture)
* Recognition accuracy ≥ 95%
* Liveness detection required before training
* Smooth UI performance at 60 FPS (Flutter)

**SYSTEM DESIGN OVERVIEW**

*(Architecture + Technologies Explanation)*

This section explains how **FaceAuth 2.0 – Student Attendance System** is designed, how its components interact, and which technologies are used at each stage. The system follows a **modular, scalable, and secure AI-based architecture**.

**1. SYSTEM ARCHITECTURE (HIGH-LEVEL)**

Below is a clear breakdown of how the system flows from user actions → camera → ML model → backend → database → ERP sync.

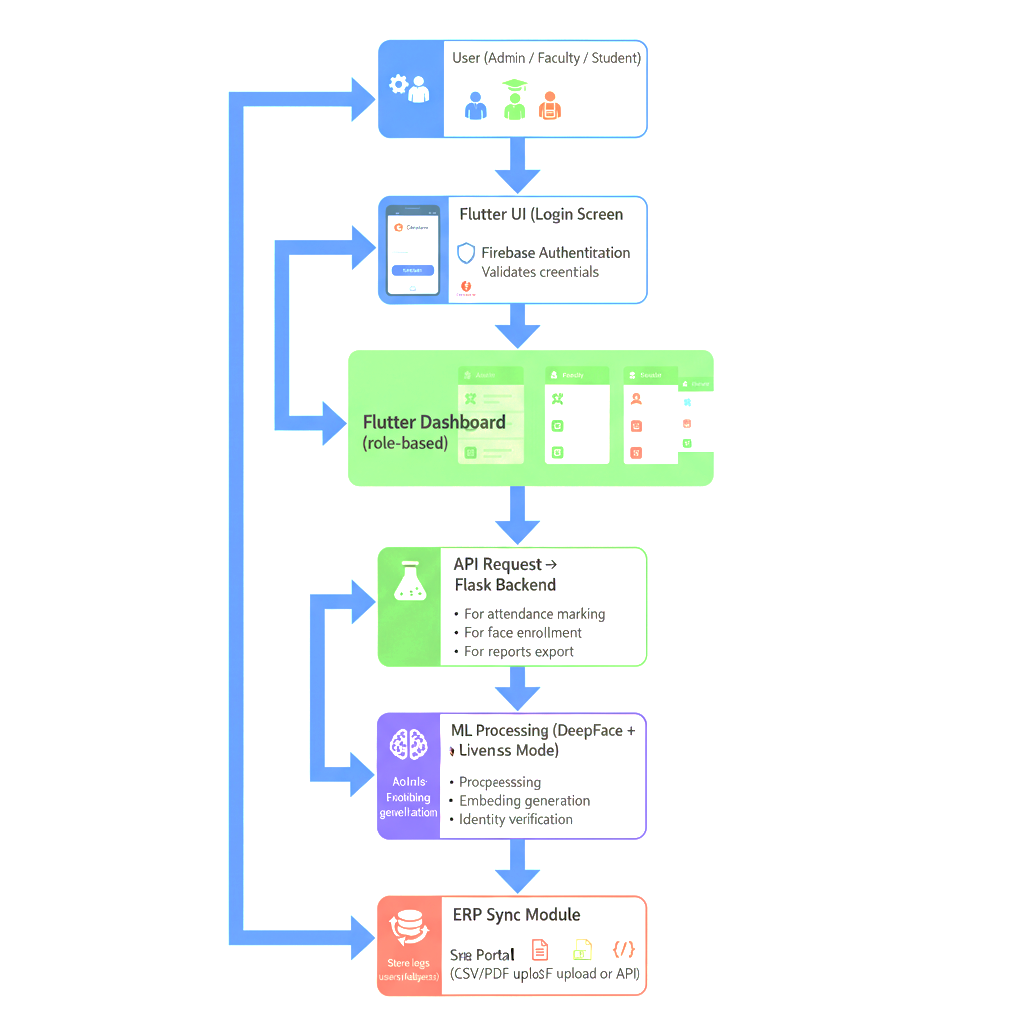
**A. High-Level Architecture Components**

1. **Flutter Mobile Application (Frontend)**
   * User interface for Admin, Faculty, and Students
   * Camera access for face scanning and liveness detection
   * Authentication using Firebase Auth
   * API requests sent to Flask backend
2. **Flask Backend (Server Layer)**
   * Handles attendance logic, recognition API, user management
   * Processes images using DeepFace, OpenCV, and Liveness detection model
   * Generates embeddings and validates matches
   * Communicates with database (SQLite/MySQL)
3. **Machine Learning Engine (AI/ML Layer)**
   * **DeepFace Recognition Model**
   * **Liveness Detection Model** (blink/movement)
   * **Auto-Model Training** for new face enrollments (2–3 training rounds)
4. **Firebase Services**
   * Firebase Authentication → login & identity management
   * Firebase Cloud Storage → store training images/templates
5. **Database (SQLite / MySQL)**
   * Stores users, face embeddings, attendance logs, session data, and support requests
6. **ERP Integration Module**
   * Allows Admin to export and sync attendance to the university ERP system

**2. SYSTEM DESIGN DIAGRAM (TEXTUAL DESCRIPTION)**

**User (Admin / Faculty / Student)**  
⬇  
**Flutter UI (Login Screen)**  
⬇  
**Firebase Authentication**

* Validates credentials  
  ⬇  
  **Flutter Dashboard (role-based)**  
  ⬇  
  **API Request → Flask Backend**
* For attendance marking
* For face enrollment
* For reports export  
  ⬇  
  **ML Processing (DeepFace + Liveness Model)**
* Preprocessing
* Embedding generation
* Identity verification  
  ⬇  
  **Database Layer (SQLite/MySQL)**
* Store logs
* Retrieve users/subjects  
  ⬇  
  **ERP Sync Module**  
  ⬇  
  **ERP Portal (CSV/PDF upload or API)**



**3. MODULE-WISE ARCHITECTURE**

**A. Admin Module**

* Add faculty/students
* Face enrollment
* Department & subject management
* ERP attendance sync
* Credentials update
* Export and download reports

**Technologies:** Flutter UI → Firebase Auth → Flask API → SQLite/MySQL

**B. Faculty Module**

* Faculty login (face/fallback)
* Start class attendance session
* Camera recognition → DeepFace match
* Mark attendance
* Export PDF/Excel reports

**Technologies:** OpenCV, DeepFace, Flutter Camera Plugin

**C. Student Module**

* Student login
* View attendance %
* View attendance history
* Receive alerts

**Technologies:** Flutter UI + Firebase Auth + SQLite Views

**4. TECHNOLOGY STACK (EXPLAINED)**

|  |  |  |
| --- | --- | --- |
| **Layer** | **Technology** | **Purpose** |
| **Frontend** | Flutter | Cross-platform UI, camera access |
| **Backend** | Flask (Python) | Processing API, logic, DB communication |
| **AI/ML** | DeepFace, OpenCV, Liveness Detection | Identity verification and anti-spoofing |
| **Database** | SQLite/MySQL | Store user & attendance data |
| **Cloud** | Firebase Auth, Firebase Storage | Authentication + image storage |
| **ERP Sync** | CSV/PDF export, API-ready format | Integration with institutional systems |
| **Design** | Figma | UI/UX prototypes |

**5. SECURITY DESIGN**

**A. Authentication**

* Firebase Authentication for Admin, Faculty, Student login
* Role-based data access layers

**B. Liveness Detection**

* Prevents spoof using images/videos
* Ensures real user presence

**C. Encrypted Face Data**

* Embeddings stored securely
* Image storage restricted to authenticated users

**D. Database Security**

* Input validation
* SQL injection protection
* Secure token access for API calls

**6. PERFORMANCE DESIGN**

* Fast preprocessing (OpenCV optimized)
* ML inference optimized for mobile
* Bulk attendance export enabled
* Low-latency recognition (<1 second per frame on modern devices)
* Caching face embeddings for faster lookup

1. **ARCHITECTURE PATTERN USED**

* **Client–Server**
* **Modular Architecture** (Admin, Faculty, Student modules)
* **MVC (Flutter)**
* **REST API integration**
* **Microservice-ready structure** for future scaling

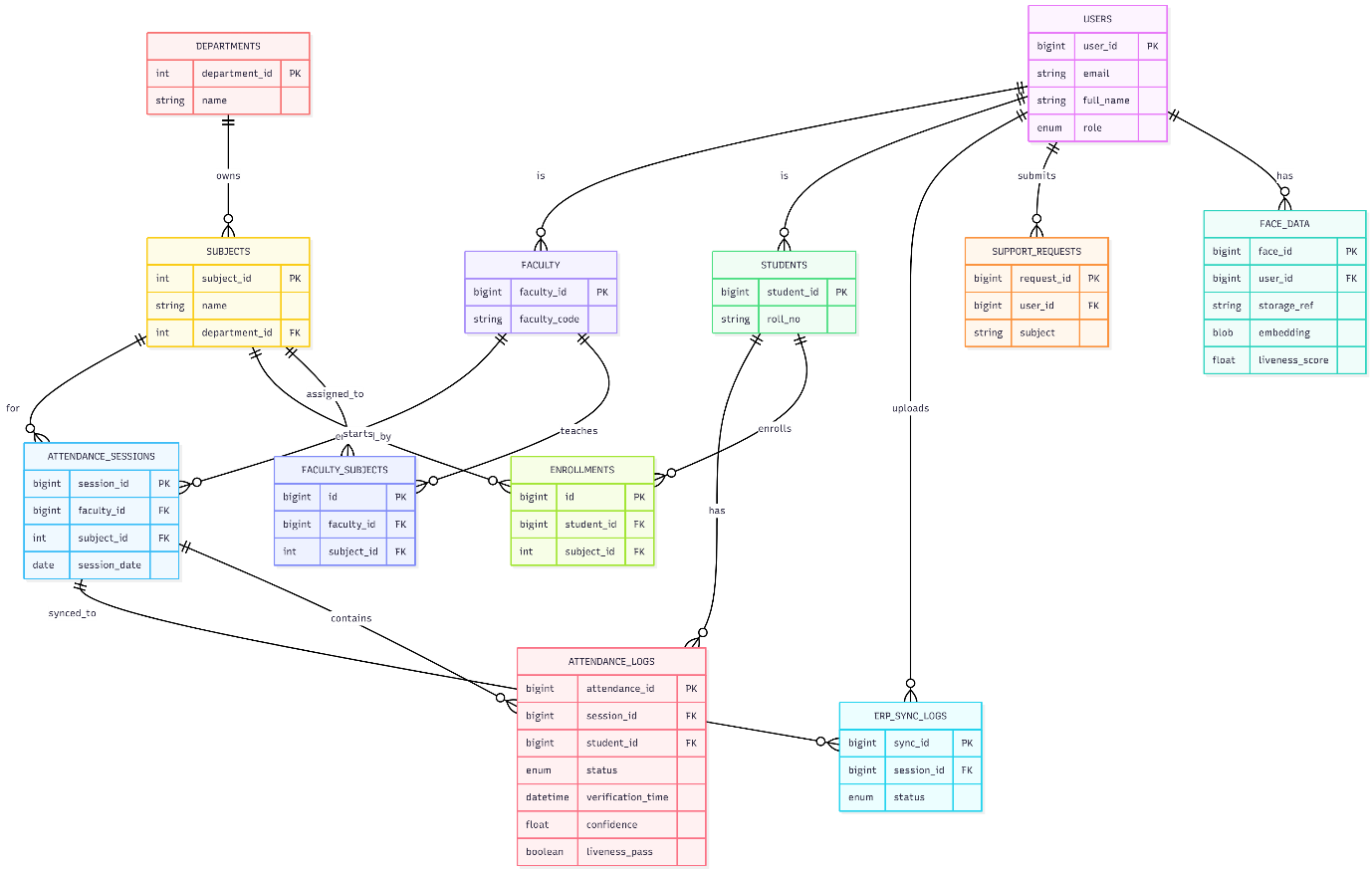
**ER DIAGRAM EXPLANATION & TABLE DESCRIPTIONS**

This section explains your ER Diagram in a clear, professional, and academic manner, followed by detailed **table descriptions** for each entity used in **FaceAuth 2.0 – Student Attendance System**.

**1. ER DIAGRAM OVERVIEW**

The Entity–Relationship Diagram (ERD) for **FaceAuth 2.0** represents how data is organized and interconnected within the system. The ERD includes entities for users, faculty, students, subjects, attendance sessions, attendance logs, face data, and support requests.

The system uses a **role-based structure**, where **Admin**, **Faculty**, and **Students** are stored inside a shared Users table, and additional details are extended through separate tables like Faculty, Students, and Subjects.

**Diagram:**

**The ERD supports:**

* AI-based face enrollment
* DeepFace recognition
* Attendance tracking
* ERP sync
* Role-based access modules

**2. ENTITY DESCRIPTIONS**

Below are professional explanations of each entity in the database.

**2.1 Users -** This is the core table containing login and role-based information for all system users — Admin, Faculty, and Students.

**Attributes:**

* user\_id – Primary key
* email – Firebase linked email
* password\_hash – Encrypted password
* full\_name – User’s full name
* role – admin / faculty / student
* phone – Contact number
* is\_active – Enables or disables account

**Purpose:** Handles authentication, authorization, and general identity information.

**2.2 Departments -** Represents academic departments within the university.

**Attributes:**

* department\_id
* name
* code

**Purpose:** Helps classify subjects and faculty.

**2.3 Subjects -** Represents individual subjects taught by faculty.

**Attributes:**

* subject\_id
* code
* name
* department\_id
* semester

**Purpose:** Used in attendance session mapping and faculty assignment.

**2.4 Faculty -** Contains additional details for faculty users.

**Attributes:**

* faculty\_id (FK from Users)
* faculty\_code
* department\_id
* designation

**Purpose:** Assigns faculty to departments & connects to subjects they teach.

**2.5 Students -** Holds academic details for student users.

**Attributes:**

* student\_id (FK from Users)
* roll\_no
* branch
* section
* year
* batch

**Purpose:** Supports subject enrollment and attendance logging.

**2.6 Faculty\_Subjects -** Maps which faculty teaches which subject.

**Attributes:**

* faculty\_id
* subject\_id
* assigned\_from
* assigned\_to

**Purpose:** Defines faculty-subject assignment for attendance sessions.

**2.7 Enrollments -** Maps each student to the subjects they are enrolled in.

**Attributes:**

* student\_id
* subject\_id
* enrolled\_on
* is\_active

**Purpose:** Used for validating attendance lists in each class.

**3. ATTENDANCE-RELATED ENTITIES**

**3.1 Attendance\_Sessions -** Stores the session details whenever a faculty starts attendance using face recognition.

**Attributes:**

* session\_id
* faculty\_id
* subject\_id
* session\_date
* start\_time
* end\_time
* location
* is\_finalized

**Purpose:** Links attendance logs to specific classes.

**3.2 Attendance\_Logs -** Records each student’s attendance status for every session.

**Attributes:**

* attendance\_id
* session\_id
* student\_id
* status (present/absent/late/etc.)
* verification\_time
* confidence
* liveness\_pass
* face\_template\_ref

**Purpose:** Stores the output of DeepFace and liveness detection results.

**4. FACE DATA ENTITIES**

**4.1 Face\_Data -** Stores ML training information and face embedding metadata.

**Attributes:**

* face\_id
* user\_id
* storage\_ref
* embedding (BLOB)
* embedding\_version
* liveness\_score
* training\_iterations

**Purpose:** Used for face recognition and verification.

**5. ERP & SUPPORT ENTITIES**

**5.1 ERP\_Sync\_Logs -** Stores all ERP upload attempts and statuses.

**Attributes:**

* sync\_id
* session\_id
* uploaded\_by
* status
* response\_message

**5.2 Support\_Requests -** Stores help/support queries sent from the app.

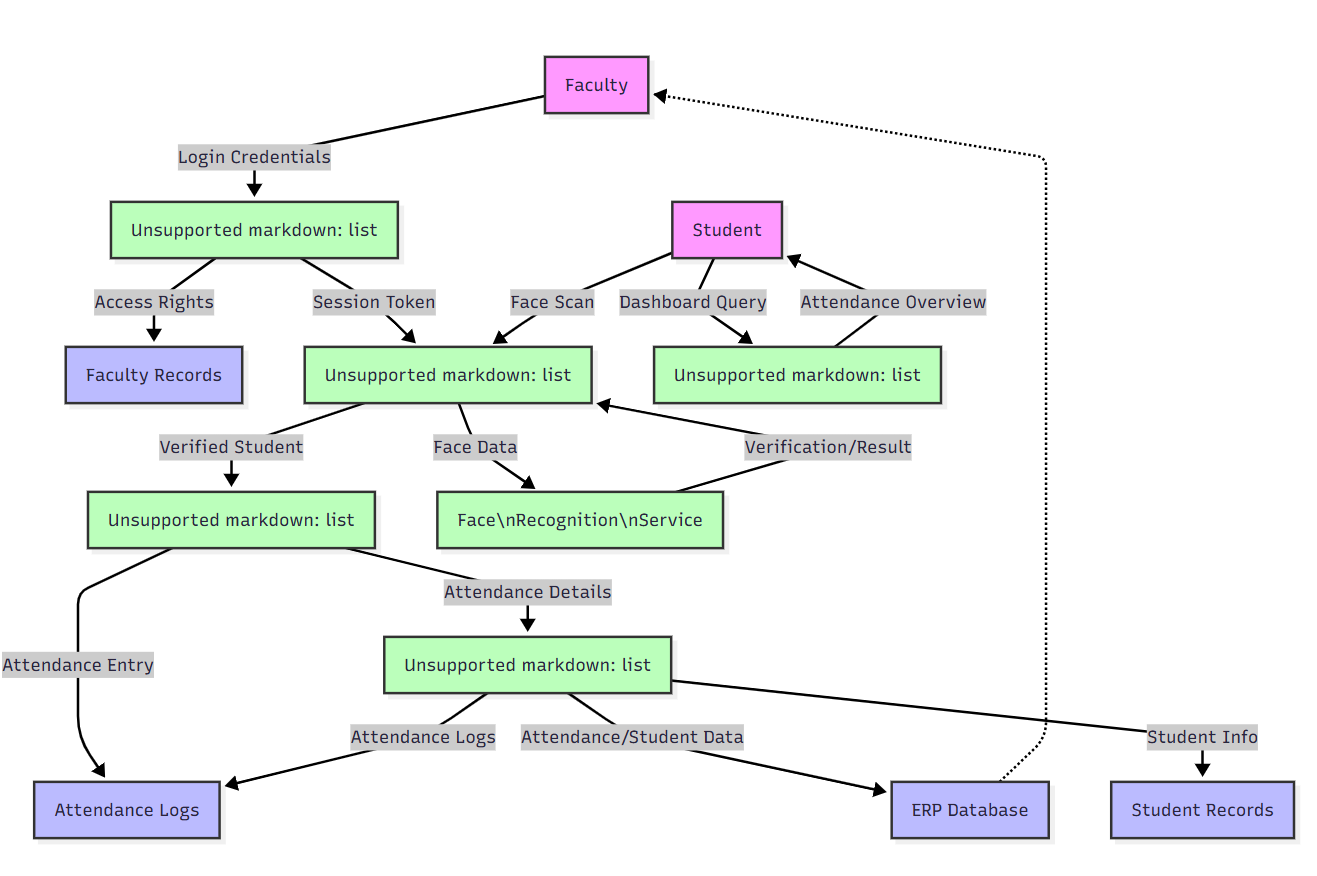
**Attributes:**

* request\_id
* user\_id
* subject
* message
* status

**6. RELATIONSHIP EXPLANATION**

* **Users → Faculty / Students** (1:1)
* **Departments → Subjects** (1:N)
* **Faculty → Faculty\_Subjects → Subjects** (M:N)
* **Students → Enrollments → Subjects** (M:N)
* **Faculty → Attendance\_Sessions → Students** (class attendance)
* **Attendance\_Sessions → Attendance\_Logs** (1:N)
* **Users → Face\_Data** (1:N)
* **Attendance\_Sessions → ERP\_Sync\_Logs** (1:N)
* **Users → Support\_Requests** (1:N)

**Diagram : 6.2**

****

**DATA FLOW DIAGRAM (DFD) EXPLANATION**

*(Level 0, Level 1, and Level 2 – Full Professional Documentation)*

This section explains the **data movement**, **processes**, and **interactions** in **FaceAuth 2.0** using structured DFD levels. A DFD provides a clear visualization of how data flows between users, processes, and data stores.

**LEVEL 0 DFD (CONTEXT DIAGRAM)**

The **Level 0 DFD** represents the entire **FaceAuth 2.0** system as a single process interacting with external entities.

**External Entities:**

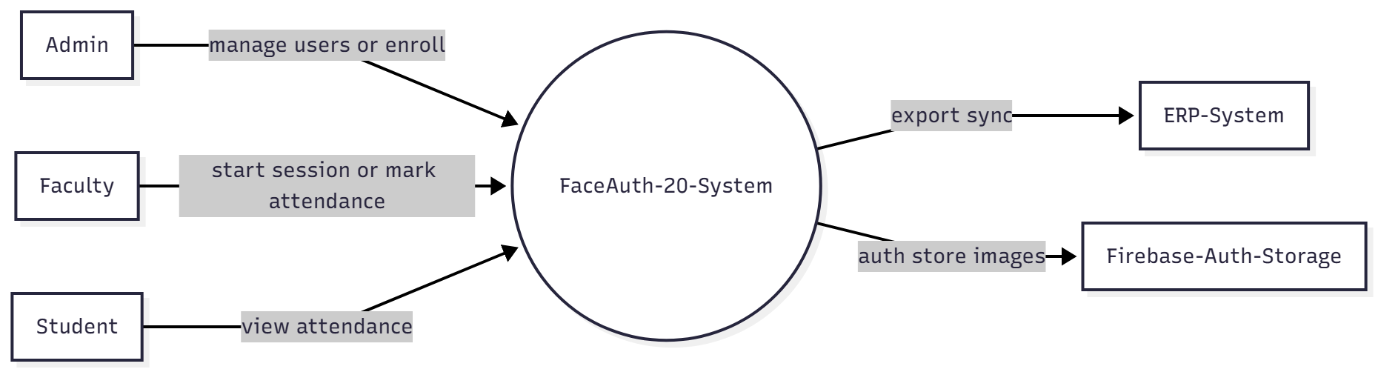
* **Admin**
* **Faculty**
* **Student**
* **ERP System**
* **Firebase Authentication**

**Main Process:**

**FaceAuth 2.0 System**

**Data Stores (Conceptual):**

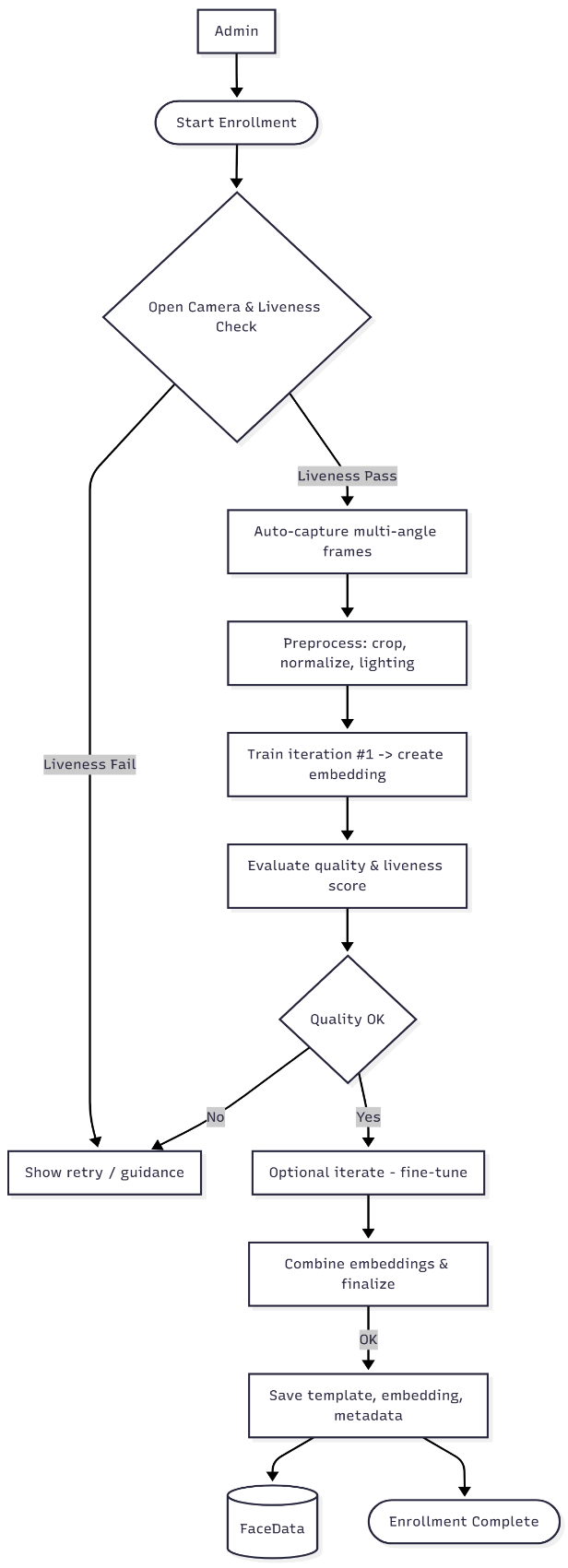
* User Database
* Face Data
* Attendance Logs
* Subjects/Departments

**Diagram:**

**LEVEL 1 DFD (DETAILED DECOMPOSITION)**

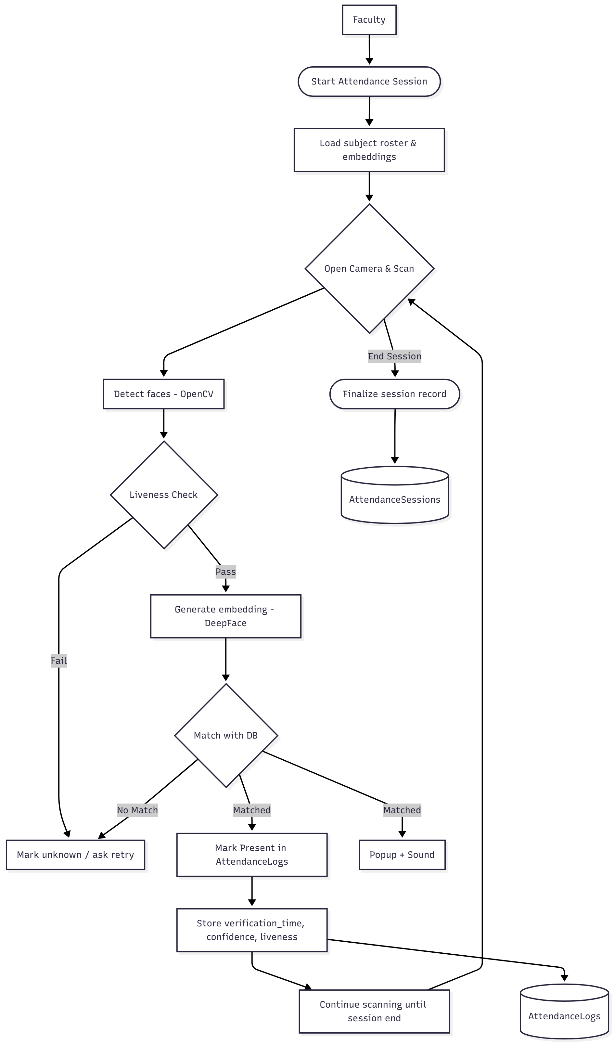
At Level 1, the main system is broken into 6 primary processes:

**Processes:**

1. **User Management**
2. **Face Enrollment & ML Training**
3. **Face Recognition & Attendance Marking**
4. **Attendance Reporting**
5. **ERP Sync**
6. **Support & Notification System**

**Data Stores:**

* **DS1: Users**
* **DS2: FaceData**
* **DS3: Subjects/Departments**
* **DS4: AttendanceSessions**
* **DS5: AttendanceLogs**
* **DS6: SupportRequests**

**Diagram:**

**Level 1 Flow Explanation**

**1. User Management**

* Admin adds/edit/deletes users
* User details stored in **DS1 Users**
* Subjects mapped using **DS3 Subjects**

**2. Face Enrollment**

* Admin initiates enrollment
* System captures liveness (blink/head movement)
* Auto-ML training generates embeddings
* Data stored in **DS2 FaceData**

**3. Face Recognition & Attendance Marking**

* Faculty starts recognition session
* System loads:
  + FaceData (DS2)
  + Users (DS1)
  + Subject rosters (DS3)
* Real-time recognition occurs
* Attendance recorded in **DS5 AttendanceLogs**
* Session stored in **DS4 AttendanceSessions**

**4. Attendance Reporting**

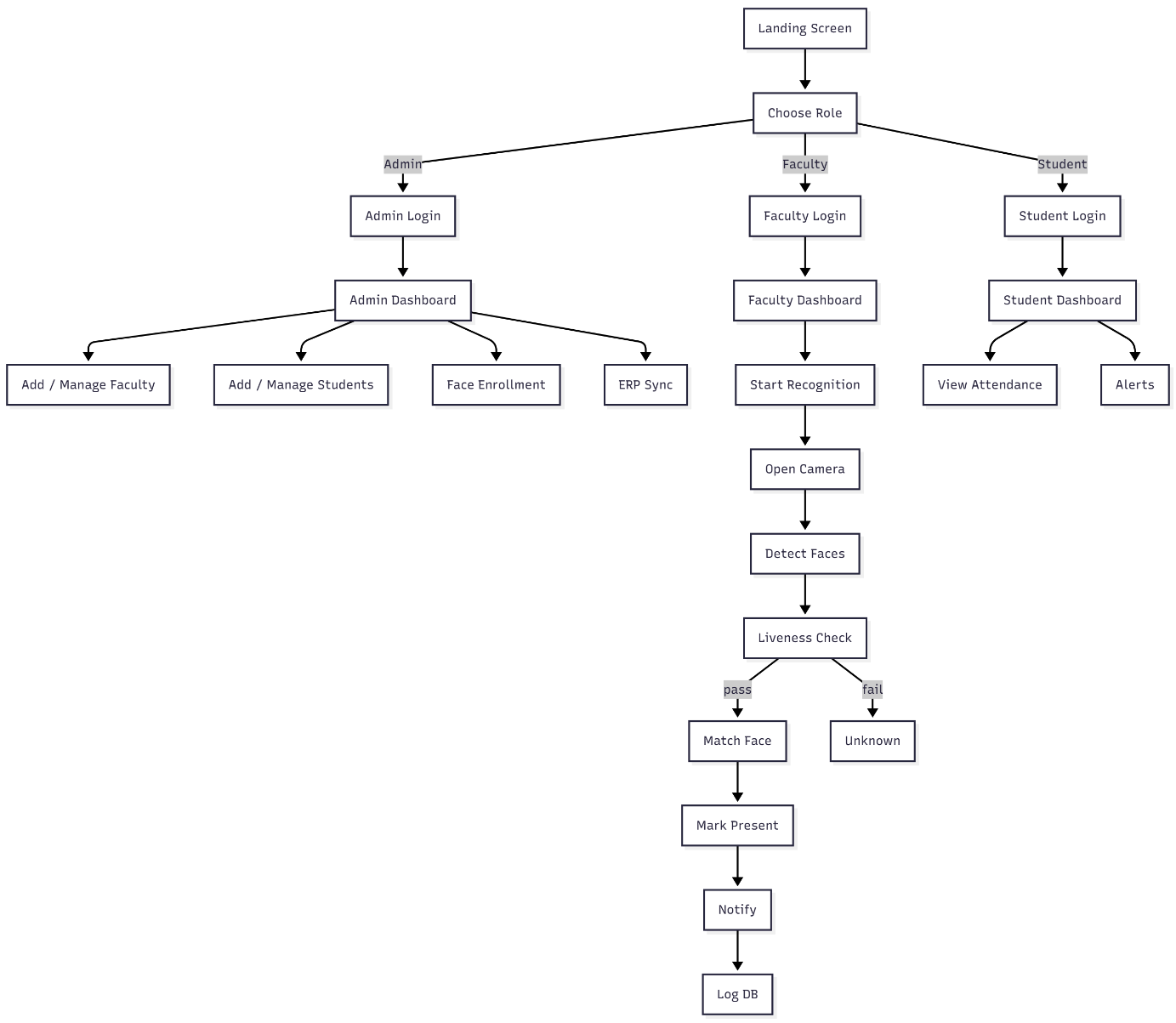
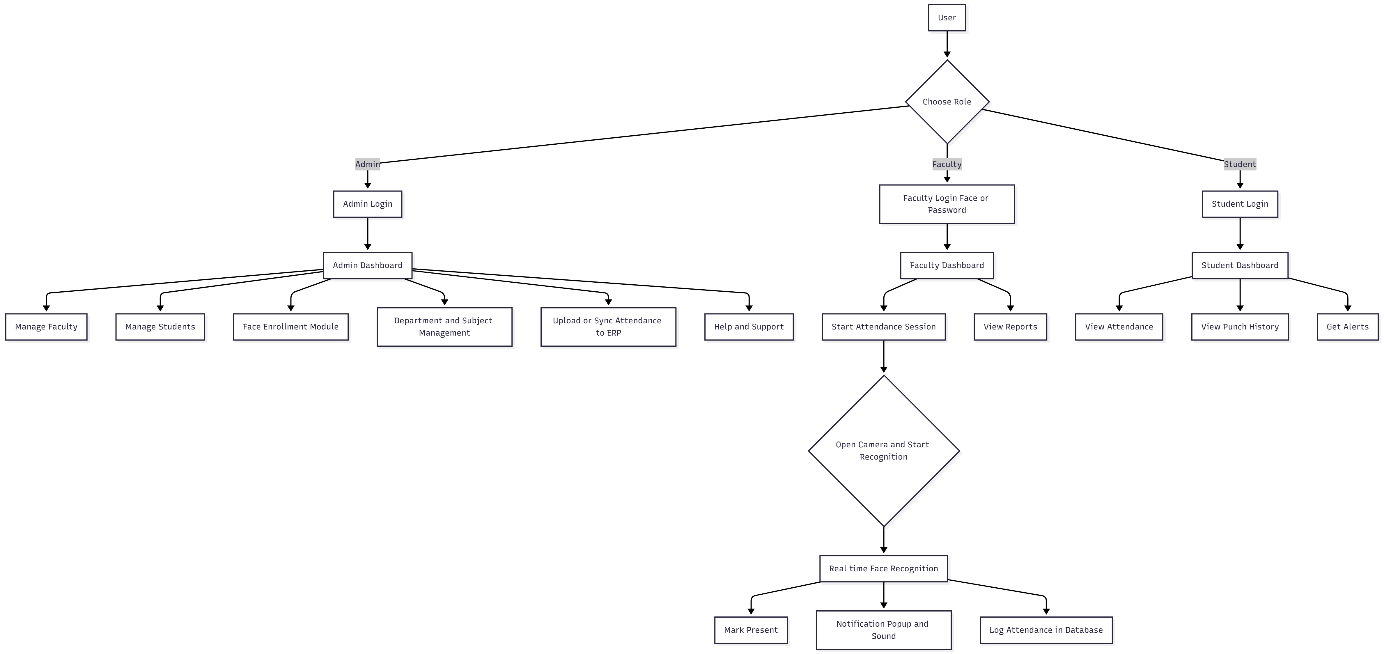
* Faculty/Admin requests report
* System fetches logs from DS5
* Generates PDF/Excel
* Student views report via query to DS5

**5. ERP Sync**

* Admin syncs attendance
* System exports logs
* ERP responds with status
* Status stored in **DS4 Sessions**

**6. Support & Help System**

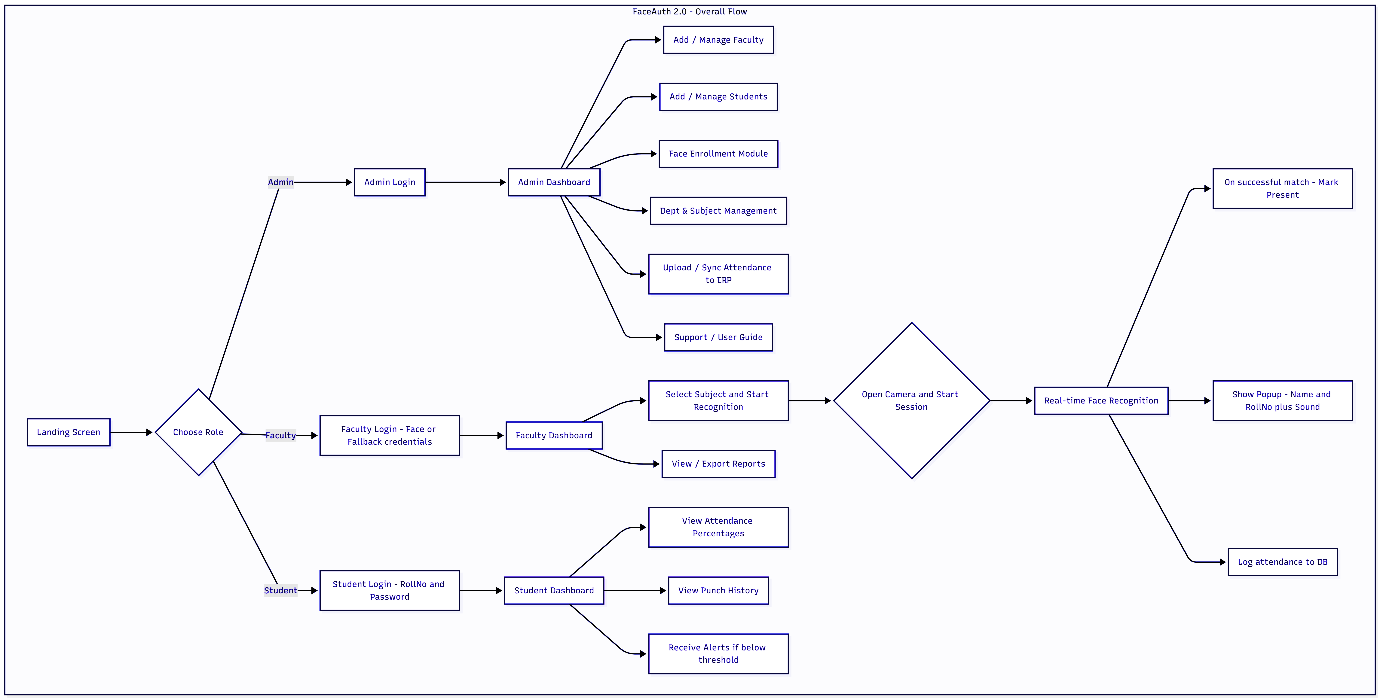
* User submits request
* Stored in **DS6 SupportRequests**
* Notified to Admin

**Diagram:**

**LEVEL 2 DFD (PROCESS-LEVEL DECOMPOSITION)**

Level 2 breaks down the two most complex processes:

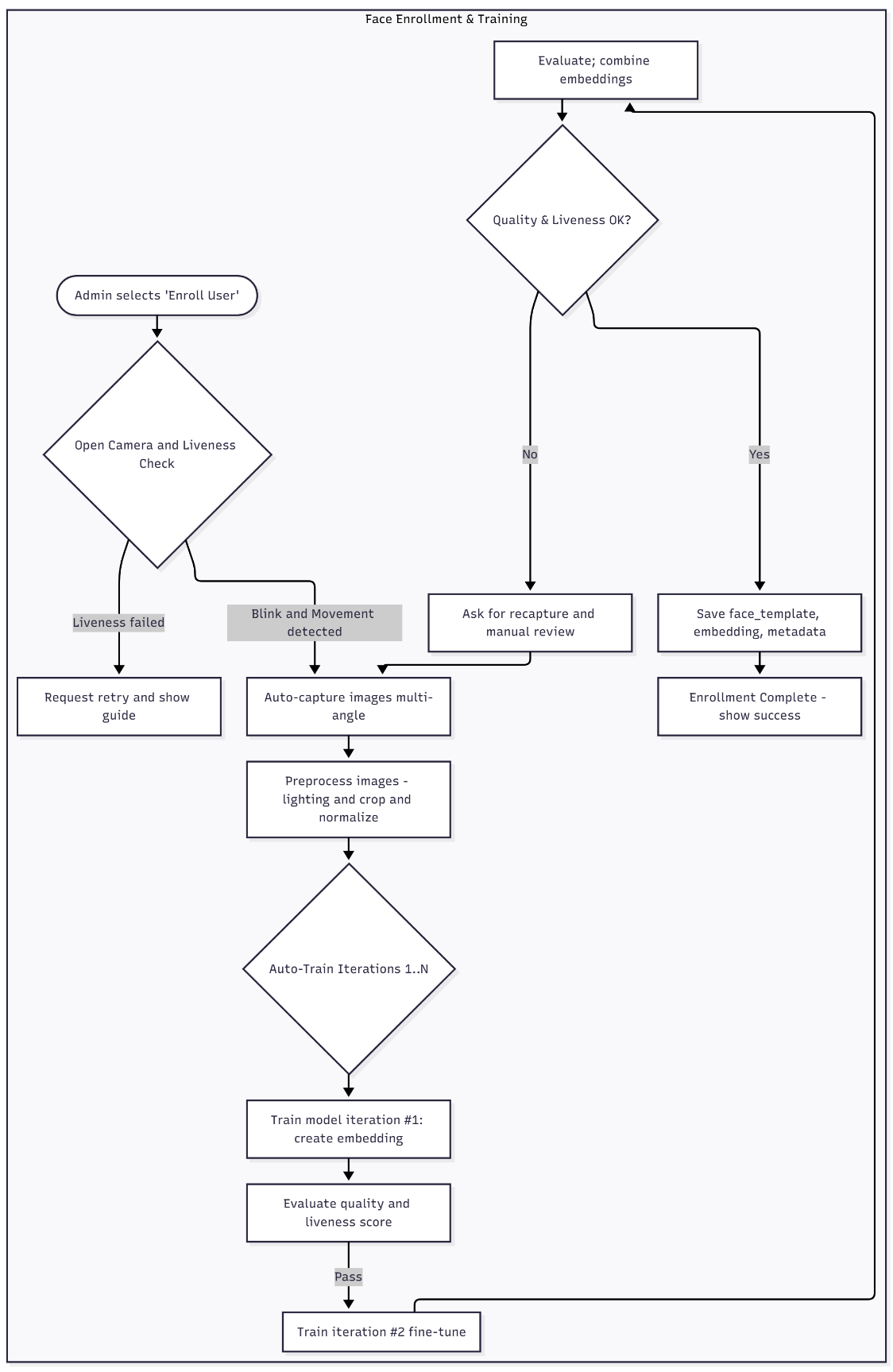
* **Face Enrollment**
* **Face Recognition & Attendance Marking**



**LEVEL 2 – PROCESS 2: Face Enrollment (Detailed)**

**Steps:**

1. Admin adds new user → Users table (DS1)
2. Admin selects “Add Face Data”
3. System opens camera
4. Liveness detection ensures:
   * Blink movement
   * Head tilt
5. If real user:
   * Multiple images captured
6. Preprocessing:
   * Cropping
   * Lighting adjustment
   * Normalization
7. Auto-Training:
   * DeepFace embedding created
   * 2–3 learning iterations
8. Store:
   * Embeddings → DS2
   * Image reference → Firebase Storage
9. Link face ID → user ID

**Output:** Enrollment successful → stored in database and cloud.

**LEVEL 2 – PROCESS 3: Attendance Recognition (Detailed)**

**Steps:**

1. Faculty selects subject → fetch students list
2. System loads:
   * Face embeddings (DS2)
   * Enrolled student list (DS3, DS1)
3. Camera activated
4. Face detection (OpenCV)
5. Liveness verification
6. Embedding generated
7. Matching using DeepFace similarity
8. If match:
   * Mark present in DS5
   * Display popup: Name + Roll No
   * Play a confirmation sound
9. If no match:
   * Display “Unknown Face”
10. Continue real-time scanning
11. End session → DS4 updated

**LEVEL 2 – PROCESS 4: Reporting**

**Detailed:**

* Faculty/Admin requests report
* System queries DS5 logs
* Generates:
  + Daily report
  + Monthly report
  + Subject-wise report
* Export as Excel/PDF

**LEVEL 2 – PROCESS 5: ERP Sync**

**Detailed:**

1. Admin selects “Sync to ERP”
2. System compiles attendance log
3. Data sent to ERP
4. ERP returns success/failure
5. Log saved in DS4

**LEVEL 2 – PROCESS 6: Support System**

**Steps:**

1. User submits support request
2. System stores it in DS6
3. Notification sent to Admin
4. Admin replies manually (email/phone)

**DFD SUMMARY**

|  |  |
| --- | --- |
| **DFD Level** | **Description** |
| **Level 0** | Overall system as a single process |
| **Level 1** | Divides into 6 major functional processes |
| **Level 2** | Explains technical inner working of enrollment & attendance |

**This completes your full DFD explanation section.**

**MODULE-WISE FUNCTIONAL SPECIFICATIONS**

*(Admin Module, Faculty Module, Student Module, Support Module)*

This section defines the **functional behavior**, **features**, and **system responsibilities** of each module in **FaceAuth 2.0**. It is written professionally, suitable for academic documentation.

**1. ADMIN MODULE**

The **Admin Module** provides complete system control and is accessible only by authorized administrators.

**Key Functionalities:**

**1.1 User Management**

* Add new Faculty or Student accounts.
* Edit/update user information.
* Delete or deactivate user profiles.
* Reset credentials for Faculty/Students.
* Assign roles and permissions.

**1.2 Department & Subject Management**

* Create new departments.
* Add/edit/delete subjects.
* Assign subjects to faculty members.
* Manage academic structure (year/section/branch).

**1.3 Face Enrollment Management**

* Initiate face registration for new users.
* Capture images using liveness detection (blink/head movement).
* Execute auto-ML training to generate embeddings.
* Approve or reject face enrollment if quality is low.
* Store embeddings in database and Firebase.

**1.4 Attendance Control & Monitoring**

* View attendance reports by:
  + Date
  + Subject
  + Faculty
  + Student
* Update/Correct attendance records.
* Monitor activity logs for suspicious entries.

**1.5 ERP Integration**

* Export attendance in CSV/PDF.
* Sync attendance to institutional ERP.
* View ERP sync logs and statuses.

**1.6 Support & Documentation**

* View support requests submitted by users.
* Provide help assistance.
* Edit application README/About section.
* Access complete system documentation.

**2. FACULTY MODULE**

This module is used by faculty members to take attendance, monitor class presence, and manage subject-related data.

**Key Functionalities:**

**2.1 Faculty Login**

* Login using:
  + Face Authentication (DeepFace + Liveness Detection)
  + Faculty ID + Password (fallback option)

**2.2 Start Attendance Session**

* Select subject from assigned subject list.
* Set class parameters (session time, section).
* Initiate camera-based scanning.
* Start real-time detection & face matching.

**2.3 Automated Attendance Marking**

* System detects a face → verifies liveness.
* DeepFace compares embedding with stored data.
* If matched:
  + Mark student **Present**
  + Display popup: *Name + Roll No*
  + Play confirmation sound
* System continuously scans until session ends.

**2.4 Attendance Review**

* View today’s attendance.
* Update/correct attendance for their subjects.
* Add remarks for students.

**2.5 Attendance Export**

* Export attendance:
  + Daily
  + Weekly
  + Monthly
* Formats: **PDF** / **Excel**

**3. STUDENT MODULE**

The Student Module allows transparency and visibility into academic attendance.

**Key Functionalities:**

**3.1 Student Login**

* Credential-based login (Roll No + Password).
* Firebase Authentication integration.

**3.2 Attendance Dashboard**

* Display overall attendance percentage.
* Show subject-wise attendance.
* Visualize attendance trends.

**3.3 Punch/Mark History**

* Daily timestamp when student was marked present.
* Class-wise attendance logs.

**3.4 Alerts & Notifications**

* Automatic low-attendance alerts (<75%).
* Faculty/Admin-specific updates.

**3.5 Profile & Support**

* View personal profile information.
* Submit support/help request.

**4. SUPPORT & HELP MODULE**

This module provides assistance, troubleshooting, and feedback collection.

**Key Functionalities:**

**4.1 User Guide**

* Complete guide on using the application.
* Includes:
  + Login instructions
  + Face enrollment steps
  + Attendance process
  + ERP export process
* Available in PDF or in-app screens.

**4.2 Support Request Form**

* Students/Faculty can submit:
  + Issues
  + Errors
  + Complaints
  + Queries
* Automatically sent to:
  + Developer email: **kumarpratipal123@gmail.com**
  + Support database table (SupportRequests)

**4.3 About Application**

* Project description (README format).
* Developer details:
  + **Pratipal Kumar Singh (1323607)**
  + **Sonu Kumar (1323524)**
* University name & logo.

**4.4 Contact Support**

* IT Support Phone: **+91 6207852281**
* Email Support: **kumarpratipal123@gmail.com**

**MODULE INTERDEPENDENCY OVERVIEW**

|  |  |  |
| --- | --- | --- |
| **Module** | **Depends On** | **Explanation** |
| Admin | All data stores | Full system control |
| Faculty | FaceData, AttendanceLogs, Sessions | Required for recognition & attendance |
| Student | AttendanceLogs | Read-only attendance view |
| Support | Users | Log complaints & queries |

**MODULE-WISE SUMMARY**

* **Admin Module** manages the entire system and data operations.
* **Faculty Module** handles real-time face recognition attendance.
* **Student Module** provides attendance transparency and alerts.
* **Support Module** manages feedback, help, and documentation.

**UI/UX GUIDELINES FOR FACEAUTH 2.0**

*(Colors • Layout • Logo Placement • Design Standards)*

This section describes the professional user interface and user experience design guidelines followed for the **FaceAuth 2.0 – Student Attendance System** using **Flutter**, based on usability principles and clean modern UI standards.

**1. COLOR PALETTE (BRAND + SYSTEM COLORS)**

A consistent color palette ensures a professional and university-branded experience.

**Primary Colors**

|  |  |  |
| --- | --- | --- |
| **Purpose** | **Color** | **Code** |
| Brand Accent | Royal Blue | **#1A73E8** |
| Primary Buttons | Deep Blue | **#0B4CAD** |
| Headers/Titles | Navy | **#04337F** |

**Secondary Colors**

|  |  |  |
| --- | --- | --- |
| **Purpose** | **Color** | **Code** |
| Highlights | Orange | **#FF8C42** |
| Status Success | Green | **#28A745** |
| Status Error | Red | **#DC3545** |
| Status Warning | Yellow | **#FFC107** |

**Neutral Colors**

|  |  |  |
| --- | --- | --- |
| **Purpose** | **Color** | **Code** |
| Background | Light Gray | **#F4F6F8** |
| Border/Lines | Gray | **#D1D5DB** |
| Text Primary | Black | **#212529** |
| Text Secondary | Dark Gray | **#555555** |
| Card Background | White | **#FFFFFF** |

**2. TYPOGRAPHY GUIDELINES**

**Fonts Used**

* **Poppins** (Modern, clean – recommended for titles, headings)
* **Roboto** (Standard for app body text)

**Text Sizes**

|  |  |
| --- | --- |
| **Element** | **Size** |
| App Title | 24–32 px |
| Section Headings | 20–22 px |
| Labels | 14–16 px |
| Body Text | 14 px |
| Button Text | 16 px |
| Small Help Text | 12 px |

**Font Weight Recommendations**

* Titles: **Semi-Bold / Bold**
* Buttons: **Medium / Bold**
* Body Text: **Regular**

**3. LOGO & UNIVERSITY BRANDING**

**A. Landing / Splash Screen**

* Place **MM(DU) University Logo** at the top center.
* Below the logo, include:
  + Application Name: **FaceAuth 2.0**
  + Subtitle: *AI-Based Student Attendance System*

**B. Login Screen**

* Small logo top-center.
* Title text below the logo:  
  **Welcome to FaceAuth 2.0**
* Role selection buttons: Admin | Faculty | Student

**C. Dashboard Screens**

* Logo shown in the App Drawer OR on the top left of the AppBar.

**Logo Source:**

Use the official MM(DU) logo provided by the university:  
(*as mentioned in your link*)

**4. LAYOUT GUIDELINES**

**A. General Layout Rules**

* Use **card-based UI** for grouping data.
* Maintain **8/16/24 px padding** for clean spacing.
* Keep maximum white space to avoid clutter.
* Use **rounded corners (12–20 px)** for modern style.
* Components must follow **Material 3 standards**.

**B. Mobile-First Layout**

* All screens optimized for vertical orientation.
* Avoid heavy components at top—recognition camera views should be fullscreen.

**C. Navigation Structure**

* Admin Dashboard → Side Drawer
* Faculty Dashboard → Bottom Navigation OR Cards
* Student Dashboard → Simple Cards + Charts

**5. SCREEN-BY-SCREEN UI DESIGN STANDARDS**

**5.1 Splash Screen**

* MM(DU) Logo centered
* App Name: *FaceAuth 2.0*
* Simple fade animation (2 seconds)

**5.2 Landing Screen**

* Clean background (**#F4F6F8**)
* Role buttons:
  + Admin
  + Faculty
  + Student
* Use uniform rounded buttons with icons.

**5.3 Login Screens**

* **Admin Login:** Email + Password
* **Faculty Login:** Face Scan (primary)
* **Student Login:** Roll No + Password
* “Forgot Password?” and “Help” links at bottom.

**5.4 Face Enrollment**

* Camera preview (fullscreen)
* Frame guide overlay for proper alignment
* Real-time prompts:
  + Move slightly
  + Blink
  + Turn head
* Status badge colors:
  + Green → Verified
  + Yellow → Need More Light
  + Red → Failed

**5.5 Attendance Marking Screen**

* Fullscreen camera
* Live bounding boxes around faces (OpenCV overlay)
* Popup for each recognition:
  + Name
  + Roll No
  + Status: Present
* Bottom controls:
  + Start
  + Pause
  + Stop session

**5.6 Admin Dashboard**

* Cards for:
  + Total Students
  + Total Faculty
  + Attendance Reports
  + Face Enrollment
  + ERP Sync
* Use icons from Material Icons.

**5.7 Student Dashboard**

* Circular progress chart for attendance percentage
* Subject-wise cards
* Alerts for low attendance

**6. UI/UX BEST PRACTICES FOLLOWED**

* **Minimal clicks** to complete core actions.
* **Real-time feedback** (sounds, popups, alerts).
* **Error prevention** via warnings and guidance messages.
* **Accessibility:**
  + High-contrast text
  + Large buttons
  + Readable fonts
* **Consistency:**  
  Common UI patterns across Admin, Faculty, and Student modules.

**7. FIGMA DESIGN COMPLIANCE**

Design follows your provided prototype link:  
🎨 **Figma Prototype:** <https://www.figma.com/design/5iuAFU9rGCtsCRfegNDn1a/>

**IMPLEMENTATION PLAN & MILESTONES**

*(Project Timeline for FaceAuth 2.0 – Student Attendance System)*

This section defines a clear, structured, and professional implementation roadmap for developing **FaceAuth 2.0**. It includes phases, tasks, and timeline milestones that you can include directly in your project documentation.

**1. PROJECT DEVELOPMENT PHASES**

The project is divided into **six major phases**:

1. **Planning & Requirements Gathering**
2. **System Design & Architecture**
3. **Frontend (Flutter) Development**
4. **Backend (Flask + ML Model) Development**
5. **Integration & Testing**
6. **Deployment, Documentation & Review**

**2. DETAILED IMPLEMENTATION PLAN**

Below is a professional, step-by-step plan:

**Phase 1 – Planning & Requirement Analysis (Week 1)**

**Tasks:**

* Understand institution requirements
* Identify necessary data fields (Student, Faculty, Subjects)
* Study DeepFace, OpenCV, and liveness detection models
* Analyze ERP format for attendance sync
* Prepare project scope and objectives

**Deliverables:**

* Requirement Specification Document
* Project Plan Approval

**Phase 2 – System Design (Week 2)**

**Tasks:**

* Create system architecture diagram
* Prepare ER Diagram
* Create Level 0, Level 1, Level 2 DFDs
* Design UI/UX screens using Figma
* Decide database structure (SQLite or MySQL)

**Deliverables:**

* ERD, DFDs
* Figma UI Designs
* Database schema

**Phase 3 – Frontend Development: Flutter (Week 3–4)**

**Tasks:**

* Setup project structure
* Design reusable components (cards, buttons, dialogs)
* Implement:
  + Splash Screen
  + Landing Screen
  + Login screens (Admin, Faculty, Student)
  + Dashboards
  + Face enrollment UI
  + Face recognition camera screen
  + Attendance reporting UI
  + Support & About pages

**Deliverables:**

* Working Flutter UI
* User Navigation Flow completed

**Phase 4 – Backend Development: Flask + ML Models (Week 5–6)**

**Tasks:**

**A. Flask Server**

* Setup Flask project
* Create REST APIs
  + User management APIs
  + Face enrollment APIs
  + Attendance APIs
  + Report APIs
* Connect database (SQLite/MySQL)

**B. ML Model Integration**

* Integrate **DeepFace** for recognition
* Implement **Liveness Detection**
* Implement **Auto-Training Loop** (2–3 iterations)
* Create embeddings & store in FaceData

**Deliverables:**

* Functional Flask backend
* ML-powered recognition working
* API integration completed

**Phase 5 – Integration & Testing (Week 7–8)**

**Tasks:**

* Integrate Flutter with Flask
* Connect Firebase Authentication
* Connect Firebase Storage
* Test:
  + Face enrollment
  + Recognition accuracy
  + Liveness detection
  + Session-based attendance marking
  + ERP export
  + PDF/Excel downloads
* Fix bugs and optimize performance

**Deliverables:**

* Fully integrated system
* Test case reports
* Recognition accuracy ≥ 95%

**Phase 6 – Deployment & Documentation (Week 9)**

**Tasks:**

* Deploy Flask backend on server (PythonAnywhere, Render, AWS, etc.)
* Create APK build of Flutter app
* Prepare:
  + Final Project Documentation
  + User Manual
  + Installation Guide
  + Screenshots
* Final demonstration & viva preparation

**Deliverables:**

* Live deployed backend
* APK for demonstration
* Printed & digital project report
* Final presentation

**3. MILESTONE TIMELINE (GANTT STYLE)**

|  |  |  |
| --- | --- | --- |
| **Phase** | **Duration** | **Milestone** |
| Planning & Requirements | Week 1 | Requirements Confirmed |
| System Design | Week 2 | ERD + DFD + UI Ready |
| Flutter Frontend | Week 3–4 | App UI Complete |
| Flask Backend + ML | Week 5–6 | Core AI & API Done |
| Integration & Testing | Week 7–8 | Full System Functional |
| Deployment & Documentation | Week 9 | Final Submission |

**4. FUTURE ENHANCEMENT ROADMAP**

|  |  |
| --- | --- |
| **Version** | **Enhancement** |
| **2.1** | Automated class scheduling for attendance |
| **2.2** | AI-based attendance predictions |
| **2.3** | Dashboard analytics with charts |
| **2.4** | Integration with complete ERP API |
| **3.0** | Fully cloud-hosted backend + Web Portal |

**TESTING PLAN**

*(Unit Testing • Integration Testing • System Testing • UAT • Sample Test Cases)*

This section provides a complete and professional testing strategy for **FaceAuth 2.0 – Student Attendance System**. It ensures that every module, API, and AI-based feature functions correctly, securely, and reliably.

**1. TESTING APPROACH**

Testing is performed in four levels:

1. **Unit Testing** – Testing individual components (Flutter widgets, APIs, ML functions)
2. **Integration Testing** – Testing how modules talk to each other
3. **System Testing** – Testing the entire app as one system
4. **User Acceptance Testing (UAT)** – Final validation with sample users

**2. UNIT TESTING**

**A. Frontend (Flutter) Unit Tests**

Test each component separately:

|  |  |
| --- | --- |
| **Component** | **Test Description** |
| Login Screens | Validate input (email, password, roll number) |
| Buttons & UI Widgets | Ensure navigation and clicks work |
| Face Enrollment UI | Checks for camera permissions, alignment frames |
| Attendance Camera Screen | Test real-time preview and frame updates |
| Support Form | Validate fields and send actions |

**B. Backend (Flask) Unit Tests**

Test every API endpoint independently:

|  |  |
| --- | --- |
| **API** | **Test Cases** |
| /login | Valid/invalid credentials |
| /register\_face | Save face, reject low-quality face |
| /recognize | Match face, mismatched face, liveness fail |
| /attendance/mark | Correct session mapping |
| /attendance/report | Fetch daily/weekly/monthly reports |

**C. ML Model Unit Testing**

|  |  |
| --- | --- |
| **Model** | **Test Cases** |
| DeepFace Recognition | Accuracy, false positives, false negatives |
| Liveness Detection | Detect blink/movement, reject photo replay |
| Auto-Model Training | Embedding consistency, training time |

**3. INTEGRATION TESTING**

Integration tests ensure combined components work smoothly.

**Areas Tested:**

**A. Flutter ↔ Firebase**

* Authentication success
* Token verification
* User login storage

**B. Flutter ↔ Flask API**

* Image sending from camera
* JSON response handling
* Attendance logs updating

**C. Flask ↔ Database**

* Insert/update operations
* Embedding retrieval
* Attendance session storage

**D. AI Engine ↔ Backend Logic**

* Recognition pipeline
* Liveness → Embedding → Matching
* Error fallback handling

**4. SYSTEM TESTING**

End-to-end testing of the entire application.

**Core Scenarios:**

**4.1 Face Enrollment Workflow**

* Liveness → Capture → Preprocess → ML Training → Store

**4.2 Attendance Marking**

* Real-time recognition
* Matching against DB
* Logging in AttendanceLogs
* Popup + Sound feedback

**4.3 Dashboard Operations**

* Admin, Faculty, and Student dashboards
* Export to PDF/Excel

**4.4 ERP Sync**

* Generate file
* Send to ERP
* Validate response

**5. USER ACCEPTANCE TESTING (UAT)**

Performed with:

* **1 Admin**
* **2 Faculty Members**
* **5 Students**

**Criteria:**

* App usability
* Clarity of UI
* Speed of recognition
* Accuracy of attendance
* Error handling
* Smoothness of navigation

**UAT Outcome Targets:**

* Recognition accuracy: **≥ 95%**
* Liveness detection success: **≥ 98%**
* Response time: **< 1 second per frame**
* Dashboard load time: **< 2 seconds**

**6. SAMPLE TEST CASES (TABLE FORMAT)**

Below is a very professional set of test cases for documentation.

**TEST CASE 1 – Faculty Login via Face Authentication**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_FAC\_01** |
| Module | Faculty Login |
| Input | Live Camera Face |
| Expected Output | Successful login if face matches |
| Result | Pass/Fail |
| Comments | Must verify liveness |

**TEST CASE 2 – Face Enrollment with Liveness Check**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_ADMIN\_02** |
| Module | Face Enrollment |
| Input | User face (blink + movement) |
| Expected Output | Model trained, template stored |
| Result | Pass/Fail |
| Comments | Must reject photos/videos |

**TEST CASE 3 – Start Attendance Session**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_FAC\_05** |
| Module | Attendance Module |
| Input | Faculty selects subject |
| Expected Output | Session created in DS4 |
| Result | Pass/Fail |
| Comments | Sessions must not duplicate |

**TEST CASE 4 – Real-Time Student Recognition**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_ATT\_10** |
| Module | Recognition |
| Input | Student faces during live scan |
| Expected Output | Recognized students marked Present |
| Result | Pass/Fail |
| Comments | Popup + audio must trigger |

**TEST CASE 5 – Export Report (PDF/Excel)**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_REP\_12** |
| Module | Reports |
| Input | Date range |
| Expected Output | Correct file downloaded |
| Result | Pass/Fail |
| Comments | Ensure no missing entries |

**TEST CASE 6 – ERP Sync**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_ERP\_15** |
| Module | ERP Integration |
| Input | Synced data file |
| Expected Output | ERP returns success response |
| Result | Pass/Fail |
| Comments | Must log status in DB |

**TEST CASE 7 – Student Attendance Dashboard**

|  |  |
| --- | --- |
| **Test Case ID** | **TC\_STD\_03** |
| Module | Student Module |
| Input | Student login |
| Expected Output | Correct attendance % displayed |
| Result | Pass/Fail |
| Comments | Must match DB values |

**7. AUTOMATION TESTING (OPTIONAL)**

Possible future automation testing tools:

* Selenium (for web version)
* Flutter Driver / Integration Test
* Python PyTest (for API testing)
* TensorFlow model evaluation script

**DEPLOYMENT GUIDE**

*(Flask Backend • Flutter App • Firebase • Database • ERP Sync)*

This section provides a **complete, step-by-step, professional deployment guide** for launching **FaceAuth 2.0 – Student Attendance System**. It covers backend hosting, mobile app building, ML model configuration, database setup, and ERP integration.

**1. DEPLOYMENT OVERVIEW**

The complete deployment consists of:

1. **Flutter App Deployment (Android APK / iOS Build)**
2. **Flask Backend Deployment (Cloud Hosting)**
3. **Firebase Integration (Authentication + Storage)**
4. **Database Deployment (SQLite / MySQL / Firebase Firestore)**
5. **ERP Sync Deployment (CSV/PDF export + API-ready)**

**2. FLASK BACKEND DEPLOYMENT**

You can deploy Flask using any of the following options:

**Recommended Platforms**

✔ PythonAnywhere  
✔ Render.com  
✔ Railway.app  
✔ AWS EC2 / Lightsail  
✔ Google Cloud VM  
✔ Heroku (if available for Python)

**Step-by-Step Guide (PythonAnywhere / Render.com)**

**Step 1: Prepare Your Backend Project**

Your directory must contain:

/backend

|-- app.py

|-- requirements.txt

|-- models/

|-- face\_recognition/

|-- templates/

|-- static/

|-- database.sqlite3

**Step 2: Create requirements.txt**

Include:

Flask

opencv-python

deepface

numpy

pandas

firebase-admin

requests

python-dotenv

Pillow

scikit-learn

tensorflow

(TensorFlow is optional; DeepFace works without it depending on backend.)

**Step 3: Upload Files to Hosting Platform**

* Open PythonAnywhere or Render dashboard
* Upload your backend folder
* Create a new Web App
* Select **Manual Configuration (Python 3.10+)**

**Step 4: Configure WSGI**

Set the path to:

/home/username/backend/app.py

**Step 5: Set Environment Variables**

Include:

FIREBASE\_CREDENTIALS=json-key-string

DATABASE\_URL=sqlite:///database.sqlite3

SECRET\_KEY=your-secret-key

**Step 6: Test API Endpoints**

Try:

https://faceauth.com/ping

https://faceauth.com/recognize

https://faceauth.com/enroll

**Step 7: Connect Backend URL to Flutter App**

In Flutter:

const String apiURL = "https://faceauth.com";

**3. FLUTTER APP DEPLOYMENT**

**Step 1: Configure API Base URL**

Inside your Flutter app:

lib/config/api.dart

Set:

class APIConfig {

static const baseURL = "https://your-backend-url.com";

}

**Step 2: Build Android APK**

Open terminal inside Flutter folder:

flutter clean

flutter pub get

flutter build apk --release

APK path:

/build/app/outputs/apk/release/app-release.apk

**Step 3: Build App Bundle (Play Store)**

flutter build appbundle

**Step 4: Deploy on Android Device**

Share **APK** to use it on mobile.

**4. FIREBASE SETUP**

**Step 1: Create Firebase Project**

* Go to firebase console
* Click “Add Project”
* Add Android/iOS app

**Step 2: Enable Authentication**

Go to:

Authentication → Sign-in Method → Email/Password

**Step 3: Enable Firebase Storage**

Used for:

* Face images
* Face templates

Set rules:

{

"rules": {

".read": "auth != null",

".write": "auth != null"

}

}

**Step 4: Add google-services.json to Flutter**

Place in:

android/app/

**Step 5: Setup Firebase Admin in Flask**

Download service key JSON:

import firebase\_admin

from firebase\_admin import credentials

cred = credentials.Certificate("service\_key.json")

firebase\_admin.initialize\_app(cred)

**5. DATABASE DEPLOYMENT**

**Option 1: SQLite (Local)**

* Upload SQLite file to backend environment.

**Option 2: MySQL Cloud**

Use:

* Aiven
* Railway
* ClearDB
* PlanetScale (recommended)

**Create Tables**

Use your SQL DDL script to create all tables.

**6. FACE RECOGNITION ENGINE DEPLOYMENT**

**Required folders**

/face\_data

/embeddings

**Initial Setup**

* Create directories
* Preload ML models
* Cache embeddings

**7. ERP SYNC DEPLOYMENT**

Your app must export:

* CSV
* Excel
* PDF

**Steps:**

1. Admin clicks **Sync to ERP**
2. API generates file
3. File sent to ERP endpoint (manual upload or API)
4. ERP returns status
5. Status logged in erp\_sync\_logs table

**ERP Format Example**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Roll No** | **Name** | **Subject** | **Date** | **Status** |
| 1323607 | Pratipal Singh | BCA-308 | 07-06-2025 | Present |

**8. FINAL DEPLOYMENT & TESTING**

**Checklist**

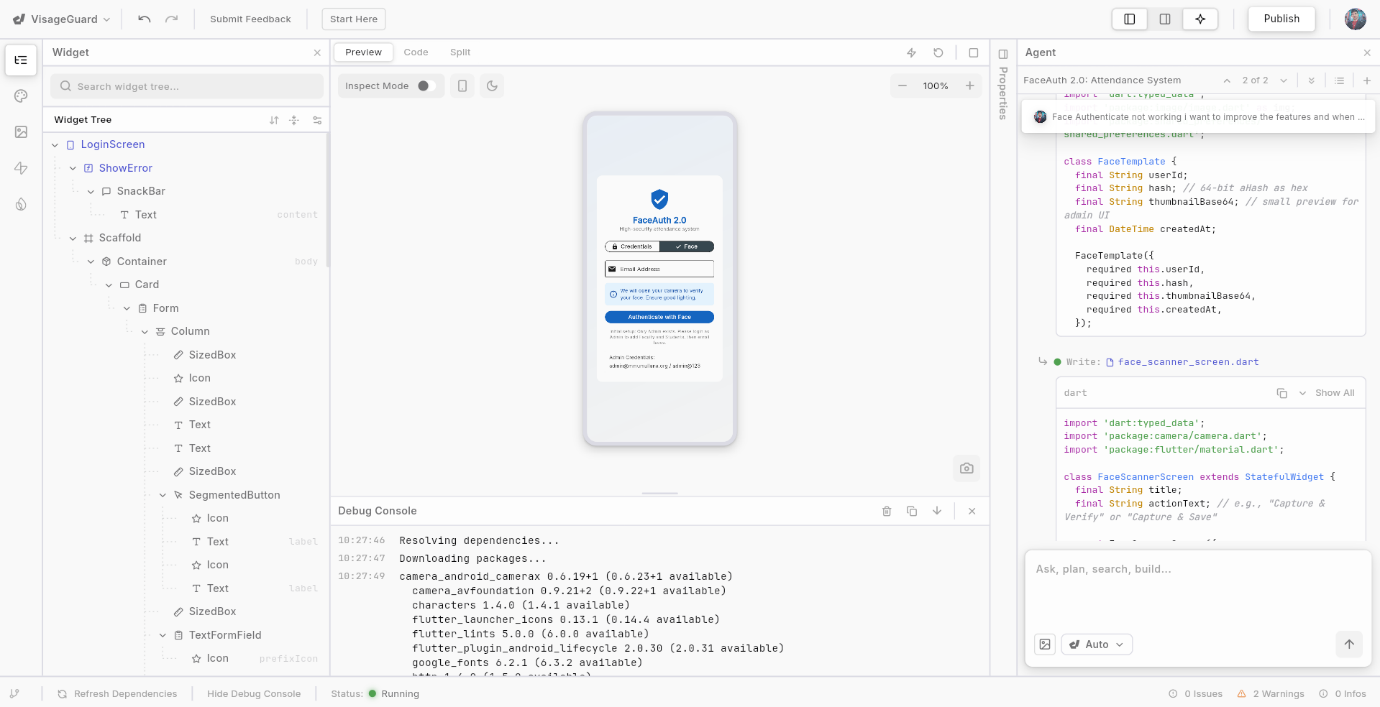
* Backend running on cloud
* Flutter APK installed and tested
* Firebase Auth working
* Face Enrollment working
* Face Recognition operational
* Reports generate correctly
* ERP sync verified
* Support form connected to:
  + Database
  + Email

**OUTPUT & SCREENS**

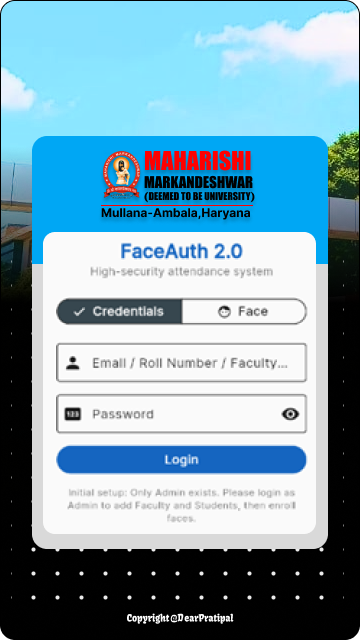
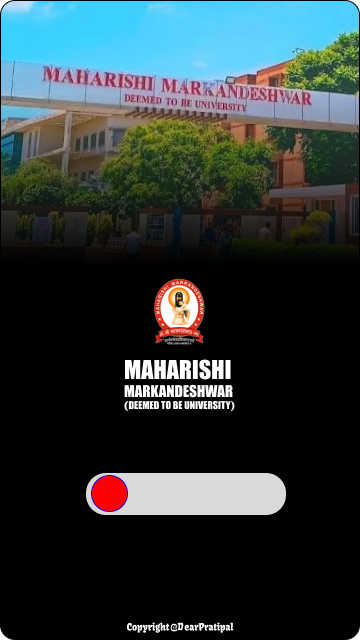
This chapter presents the **actual outputs** generated by the *FaceAuth 2.0 – Student Attendance System*. The outputs validate the correct working of each module and demonstrate how the system behaves during real operations such as login, face enrollment, real-time recognition, attendance marking, and reporting.

When the application is launched, the splash screen displays:

* University Logo (MM(DU))
* Application Title: *FaceAuth 2.0*
* Subtitle: *AI-Based Student Attendance System*

This confirms that the app is loading correctly and initializing backend services.

**FaceAuth 2.0**



|  |  |  |
| --- | --- | --- |
| Admin Dashboard Output | Faculty Face Recognition Login Output | Student Login Output |
|  |  |  |
| Login Page | | |
|  |  |  |
| Admin Dashboard | **Faculty Dashboard** | **Student Dashboard** |

|  |  |
| --- | --- |
| **Dashboard** | |
| **Admin -> Faculty Management** | **Admin -> Faculty FaceEnrollment** |
|  |  |

|  |  |
| --- | --- |
| **Admin -> Student Management** | **Admin -> Student FaceEnrollment** |
|  |  |

|  |
| --- |
| **Faculty Dashboard** |
|  |

|  |
| --- |
| **Student Dashboard** |
|  |

**MAINTENANCE & FUTURE ENHANCEMENTS**

This section outlines the long-term maintenance strategy and potential future improvements for **FaceAuth 2.0 – Student Attendance System**. It demonstrates foresight, professionalism, and scalability planning in your project documentation.

**1. MAINTENANCE PLAN**

Maintenance ensures the system remains stable, secure, fast, and up-to-date after deployment.

**1.1 Corrective Maintenance**

Fixes bugs and issues reported after deployment.

**Activities:**

* Debug incorrect attendance logs
* Fix recognition errors in low light
* Correct subject mapping errors
* Patch API and Flutter crashes
* Fix UI alignment or navigation glitches

**1.2 Adaptive Maintenance**

Modify the system to adapt to new technologies or institutional requirements.

**Activities:**

* Update DeepFace or liveness detection algorithms
* Update Flutter SDK and plugins
* Support new Android / iOS versions
* Modify ERP sync logic based on updated ERP format
* Add new subjects, branches, or departments as needed

**1.3 Perfective Maintenance**

Enhance performance, usability, or reliability based on user feedback.

**Activities:**

* Improve camera frame rate
* Speed up recognition algorithm
* Optimize database queries
* Improve UI/UX based on user testing
* Add new reporting features

**1.4 Preventive Maintenance**

Proactively detect issues before they happen.

**Activities:**

* Regular database backup
* Monitoring backend logs
* Checking Firebase quota usage
* Updating security certificates
* Cleaning unused embeddings or face data

**2. SYSTEM BACKUP STRATEGY**

To avoid data loss:

**Daily Backup**

* Attendance logs
* User data
* Face embeddings

**Weekly Backup**

* Entire database dump
* Firebase Storage folder backup

**Monthly Backup**

* Full system backup (backend + ML models + database)

**3. SECURITY MAINTENANCE**

Security is critical for a face-recognition system.

**Actions:**

* Rotate Firebase Admin keys
* Encrypt stored embeddings
* Update Flask dependencies regularly
* Enable HTTPS on backend
* Prevent brute-force login attacks
* Role-based access control enforcement

**4. FUTURE ENHANCEMENTS**

These improvements can enhance scalability, performance, and user experience in future versions of FaceAuth.

**4.1 Auto-Timetable-Based Attendance**

* App auto-detects scheduled class
* Automatically opens attendance session
* Faculty just taps **Start Recognition**

**4.2 Web Dashboard for Admin & Faculty**

* Add web portal using Flutter Web or React
* Cloud analytics dashboards
* Manage faculty/students via browser

**4.3 AI-Based Attendance Analytics**

* Predict absenteeism
* Trend charts
* Monthly performance visualization
* Identify students with low engagement

**4.4 Offline Attendance Mode**

* Local data caching
* Sync when internet reconnects

**4.5 Multi-Angle 3D Face Recognition**

* Better recognition accuracy
* More secure against spoof attacks

**4.6 OTP/QR Backup Authentication**

* If face scan fails
* Use secondary QR/OTP login method

**4.7 Push Notifications (Firebase Cloud Messaging)**

* Attendance alerts
* Admin announcements
* Subject reminders

**4.8 Dedicated Faculty & Student Portal (Web Version)**

* Attendance summary
* Face-enrollment verification
* Settings & profile

**4.9 AI Fraud Detection System**

* Detect suspicious attendance patterns
* Flag identical face embeddings
* Identify multiple attendance attempts from same person

**4.10 Fully Automated ERP API Integration**

* Direct sync without manual uploads
* Real-time API updates with campus ERP system

**5. TECHNOLOGY UPGRADE PLAN**

**Yearly Updates Suggested:**

* Update Firebase SDK
* Upgrade Flutter version
* Migrate Python to latest stable release
* Replace deprecated Flutter plugins
* Upgrade ML models to latest versions

**6. LONG-TERM SUSTAINABILITY PLAN**

**Ensure:**

* Clear documentation for future developers
* Open-source-friendly coding standards
* Modular architecture for easy updates
* Dedicated monitoring for:
  + Server uptime
  + Database size
  + Storage usage
  + API performance

This maintenance & enhancement plan ensures that **FaceAuth 2.0** remains powerful, scalable, secure, and adaptable for future institutional needs.

**CONCLUSION**

The development of **FaceAuth 2.0 – Student Attendance System** demonstrates how modern educational institutions can leverage **Artificial Intelligence, Machine Learning, and Cloud Technologies** to automate and secure attendance processes. Traditional methods such as manual roll calls, ID card scanning, and fingerprint systems often face issues such as proxy attendance, time consumption, accuracy problems, and hygiene concerns. FaceAuth 2.0 successfully addresses these limitations through **DeepFace-based face recognition**, **liveness detection**, and **auto-ML training**, ensuring high security and reliability.

The system integrates **Flutter** for building a modern, responsive, mobile-first user interface and **Flask** for handling backend logic, data processing, and AI pipelines. **Firebase Authentication and Storage** ensure secure login and cloud compatibility. The user roles of **Admin**, **Faculty**, and **Student** are designed with clear responsibilities and access levels, enabling streamlined operations.

Through real-time recognition, automated attendance marking, instant success notifications, and centralized reporting, FaceAuth 2.0 significantly reduces administrative workload, improves transparency, and enhances the overall efficiency of institutional attendance management. The system also supports ERP integration, making it ready for large-scale deployment.

Furthermore, a future-proof architecture and modular design ensure easy maintenance, scalability, and extension. With clear opportunities for enhancements—such as web dashboards, analytics, push notifications, and advanced biometric features—the application has strong potential for continued growth and real-world adoption.

In conclusion, **FaceAuth 2.0** is an innovative, AI-powered, secure, and user-friendly attendance management solution that brings institutions one step closer to a smart, digital, and automated academic environment.

**REFERENCES / BIBLIOGRAPHY**

This section lists all the sources, tools, libraries, frameworks, and research papers consulted during the development of **FaceAuth 2.0 – Student Attendance System**. All references follow a clean academic format suitable for project documentation.

**1. BOOKS & RESEARCH PAPERS**

1. **Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014).**  
   *DeepFace: Closing the Gap to Human-Level Performance in Face Verification.*  
   IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
2. **Zhang, K., Zhang, Z., et al. (2016).**  
   *Joint Face Detection and Alignment Using Multi-task Cascaded Convolutional Networks (MTCNN).* IEEE Signal Processing Letters.
3. **IEEE Research Articles (2020–2023)**  
   Various research papers on face recognition, liveness detection, and computer vision.

**2. WEBSITES & ONLINE DOCUMENTATION**

**Face Recognition & Computer Vision**

1. OpenCV Documentation – <https://docs.opencv.org>
2. DeepFace Library – <https://github.com/serengil/deepface>
3. Python face\_recognition Library – <https://github.com/ageitgey/face_recognition>

**Backend Development**

1. Flask Official Documentation – <https://flask.palletsprojects.com>
2. Python Official Documentation – <https://docs.python.org>

**Frontend & Mobile App**

1. Flutter Official Documentation – <https://flutter.dev>
2. Dart Programming Language – <https://dart.dev>
3. Flutter Liveness Detection Plugin – <https://pub.dev/packages/flutter_liveness_detection_randomized_plugin>
4. Flutter Camera Plugin – <https://pub.dev/packages/camera>

**Firebase & Cloud Services**

1. Firebase Authentication – <https://firebase.google.com/docs/auth>
2. Firebase Storage – <https://firebase.google.com/docs/storage>
3. Firebase Cloud Firestore – <https://firebase.google.com/docs/firestore>

**Database Systems**

1. SQLite Documentation – <https://sqlite.org>
2. MySQL Reference Manual – <https://dev.mysql.com/doc>

**ERP Integration Concepts**

1. CSV Format Standards – <https://www.w3.org/TR/tabular-data-model/>
2. PDF Generation (Python) – <https://pypi.org/project/reportlab/>

**3. DEVELOPMENT TOOLS**

1. Git & GitHub – <https://github.com>
2. Visual Studio Code – <https://code.visualstudio.com>
3. Android Studio – <https://developer.android.com/studio>
4. Postman API Tool – <https://www.postman.com>

**4. DESIGN TOOLS & RESOURCES**

1. Figma UI/UX Design – <https://www.figma.com>
2. Material Design Guidelines – <https://m3.material.io>
3. MM(DU) University Logo Reference – MM(DU) Official Website

**5. ADDITIONAL BLOGS & ARTICLES**

1. “Building Real-Time Face Recognition in Python” – Medium
2. “Deep Learning for Face Spoof Detection” – Towards Data Science
3. “Implementing Face Recognition with DeepFace” – Analytics Vidhya
4. “Optimizing Deep Learning Models for Mobile Devices” – Google ML Blog

**REFERENCING STYLE USED:**

* IEEE Citation Style
* APA-style formatting where suitable

This ensures your project report follows academic standards.

**APPENDIX**

*(Technical Add-ons • API Endpoints • SQL Tables • Sample Inputs/Outputs • User Manual Snippets • README Extract)*

This section provides additional technical content, examples, and resources that support the full documentation of **FaceAuth 2.0 – Student Attendance System**. These materials are optional for evaluation but extremely helpful for understanding implementation.

**APPENDIX – A**

API ENDPOINTS (FLASK BACKEND)

Below is a clean list of RESTful APIs used in the project.

**1. Authentication APIs**

**POST /login**

**Purpose:** Login user (Admin/Faculty/Student).  
**Input:** Email, Password  
**Output:** Token, Profile Details

**POST /verify-face-login**

**Purpose:** Faculty face login.  
**Input:** Image Frame (Base64)  
**Output:** Success/Failed + User ID

**2. User Management APIs (Admin Only)**

**POST /users/add**

Add new faculty/student.

**PUT /users/update/<id>**

Update user details.

**DELETE /users/delete/<id>**

Delete user.

**3. Face Enrollment APIs**

**POST /enroll-face**

**Purpose:**

* Liveness detection
* Auto training
* Store embeddings

**Input:** Multiple face frames  
**Output:** Face ID + Training status

**4. Attendance Session APIs**

**POST /session/start**

Start attendance session.  
Input: Faculty ID, Subject ID

**POST /session/recognize**

Continuous real-time face recognition.

Input: Live frame  
Output: Student ID + Present status

**POST /session/stop**

Closes attendance session.

**5. Attendance Logs APIs**

**GET /attendance/today/<subject\_id>**

Get today's attendance.

**GET /attendance/student/<student\_id>**

Fetch history.

**GET /attendance/report/export**

Download PDF/Excel.

**6. Support APIs**

**POST /support/create**

Send support request.

**GET /support/list**

View all support tickets.

**APPENDIX – B**

DATABASE TABLE STRUCTURE (DDL SQL)

These SQL structures match your ER diagram.

**Users Table**

CREATE TABLE Users (

user\_id INTEGER PRIMARY KEY AUTOINCREMENT,

full\_name TEXT NOT NULL,

email TEXT UNIQUE,

password\_hash TEXT,

role TEXT CHECK(role IN ('admin', 'faculty', 'student')),

phone TEXT,

is\_active INTEGER DEFAULT 1

);

**Students Table**

CREATE TABLE Students (

student\_id INTEGER PRIMARY KEY,

roll\_no TEXT UNIQUE,

branch TEXT,

section TEXT,

year TEXT,

batch TEXT,

FOREIGN KEY(student\_id) REFERENCES Users(user\_id)

);

**Faculty Table**

CREATE TABLE Faculty (

faculty\_id INTEGER PRIMARY KEY,

faculty\_code TEXT UNIQUE,

department\_id INTEGER,

designation TEXT,

FOREIGN KEY(faculty\_id) REFERENCES Users(user\_id)

);

**Face Data Table**

CREATE TABLE FaceData (

face\_id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id INTEGER,

storage\_ref TEXT,

embedding BLOB,

embedding\_version INTEGER,

liveness\_score REAL,

training\_iterations INTEGER,

FOREIGN KEY(user\_id) REFERENCES Users(user\_id)

);

**Attendance Logs**

CREATE TABLE AttendanceLogs (

attendance\_id INTEGER PRIMARY KEY AUTOINCREMENT,

session\_id INTEGER,

student\_id INTEGER,

status TEXT,

verification\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

confidence REAL,

liveness\_pass INTEGER,

FOREIGN KEY(session\_id) REFERENCES AttendanceSessions(session\_id),

FOREIGN KEY(student\_id) REFERENCES Students(student\_id)

);

**APPENDIX – C**

SAMPLE INPUT / OUTPUT

**Example Request – Face Recognition**

**Client Sends (Base64 Image):**

{

"frame": "/9j/4AAQSkZJRgABA..."

}

**Server Responds:**

{

"match": true,

"student\_id": 1323607,

"name": "Pratipal Kumar Singh",

"confidence": 0.97,

"liveness": 1,

"status": "present"

}

**Example Attendance Export Response**

Name, Roll No, Subject, Date, Status

Pratipal Singh, 1323607, BCA-308, 2025-04-02, Present

Sonu Kumar, 1323524, BCA-308, 2025-04-02, Present

**APPENDIX – D**

USER MANUAL SNIPPETS

**1. How to Login**

**Admin**

* Enter Email & Password
* Press **Login**

**Faculty**

* Tap **Face Login**
* Look into camera
* Blink or move head slightly

**Student**

* Enter Roll No + Password

**2. How to Enroll Face (Admin)**

1. Go to **Manage Users**
2. Select user → **Enroll Face**
3. Hold device at eye level
4. Follow instructions:
   * "Look straight"
   * "Blink"
   * "Turn head slightly"
5. Wait for 2–3 training cycles
6. Success message appears

**3. How Faculty Takes Attendance**

1. Go to **Start Attendance**
2. Select subject
3. Camera opens
4. Keep students in frame
5. Recognition happens automatically
6. Popup shows:
   * Name
   * Roll No
   * Marked Present
7. Stop session when done

**APPENDIX – E**

README.md EXTRACT (Professional)

# FaceAuth 2.0 – Student Attendance System

FaceAuth 2.0 is an AI-powered mobile attendance system that uses

DeepFace recognition and liveness detection to automate attendance

for educational institutions.

**## Features**

- Real-time face recognition

- Liveness detection (anti-spoof)

- Auto ML training during face enrollment

- Admin, Faculty, and Student modules

- PDF/Excel attendance reports

- Firebase authentication & storage

- ERP-ready export support

**## Tech Stack**

- Flutter (Frontend)

- Flask (Backend)

- DeepFace + OpenCV (AI Engine)

- Firebase (Cloud Auth & Storage)

- SQLite/MySQL (Database)

Developed by:

- Pratipal Kumar Singh (1323607)

- Sonu Kumar (1323524)

MMICT & BM (MCA), MM(DU) Mullana

**APPENDIX – F**

SAMPLE SYSTEM DIRECTORY STRUCTURE

FaceAuth2.0/

├── backend/

│ ├── app.py

│ ├── models/

│ ├── services/

│ ├── database.sqlite3

│ ├── static/

│ └── templates/

├── flutter\_app/

│ ├── lib/

│ ├── android/

│ ├── ios/

│ └── assets/

├── docs/

│ ├── ERD.pdf

│ ├── DFD.pdf

│ ├── UI\_Designs/

│ └── Project\_Documentation.pdf

└── README.md