REPUBLIC OF CAMEROON
****
Peace-Work-Fatherland
****
MINISTER OF HIGHER EDUCATION
****
UNIVERSITY OF BUEA

REPUBLIQUE DU CAMEROON
****
PAIX-Travail-Patrie
****
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
****
UNIVERSITE DE BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY

COURSE TITLE:
INTERNET PROGRAMMING (J2EE) AND MOBILE PROGRAMMING
COURSE CODE:
CEF440

TASK 6 REPORT

Database design and implementation

Course Intructor: Dr. Nkemeni Valery

GROUP IV

S/N	Names	Matricules
1	ARREY ABUNAW REGINA EBAI	FE22A152
2	AWA ERIC ANGELO JUNIOR	FE22A162
3	FAVOUR OZIOMA	FE22A217
4	OBI OBI ETCHU JUNIOR	FE22A291
5	VERBURINYUY JERVIS NYAH	FE22A324

Table of Contents

ABSTRACT

This report presents the database design and implementation phase of the *Mobile-Based Attendance Management System Based on Geofencing and Facial Recognition*. The system aims to provide a reliable, secure, and automated solution for recording student attendance in real time by combining facial biometric verification with geolocation validation.

The database design is structured to support multiple user roles students, instructors, and administrators while capturing complex interactions such as course scheduling, GPS-based geofencing, attendance check-ins, and audit logging. Key entities such as User, Course, ClassSession, Attendance, Geofence, Notification, and AuditLog have been carefully modeled to reflect real-world academic operations and ensure system scalability and maintainability.

The implementation includes the creation of relational schemas, ER diagrams, and secure backend data access mechanisms. The database is designed to handle sensitive data such as facial biometric hashes and location coordinates, with an emphasis on performance, privacy compliance, and integrity. This task lays the foundation for seamless integration with the mobile front-end and cloud-based backend APIs, enabling real-time attendance monitoring and enhanced institutional oversight.

I. Introduction

1.1 Purpose of the Task

The purpose of this task is to define, design, and implement a robust and secure database system that serves as the backbone of the **Mobile-Based Attendance Management System Based on Geofencing and Facial Recognition**. This system aims to automate the process of student attendance recording by ensuring that only physically present and verified students can check in. To achieve this, the database must efficiently support:

- Facial biometric data for secure identity verification
- Geolocation data for validating physical presence within classroom boundaries
- Real-time attendance recording, management, and retrieval
- Role-based user management (students, instructors, administrators)
- Secure storage of sensitive personal and biometric data
- Audit logs and notification tracking for transparency and accountability

This task is critical to ensuring that the entire application infrastructure operates seamlessly, securely, and in compliance with academic and data privacy standards.

1.2 Scope of Database Design

The scope of the database design encompasses the definition and management of the following key entities and relationships that reflect the system's core functionality:

- **User**: Contains details of all system users (students, instructors, admins) along with secure login credentials and facial biometric hashes for identity verification.
- **Course**: Represents the academic subjects taught by instructors, each associated with multiple class sessions.
- **ClassSession**: Represents scheduled sessions for a course, each uniquely identified and associated with a geofence boundary.
- **Attendance**: Tracks student check-ins per session, including the method used (facial recognition, GPS, or manual override) and attendance status.
- **Geofence**: Defines virtual boundaries using latitude, longitude, and radius for validating physical presence at check-in.
- **Notification**: Stores messages sent to users, such as absence alerts, check-in confirmations, or admin notices.
- **AuditLog**: Keeps records of sensitive or critical system actions (e.g., manual overrides, deletions), improving traceability and accountability.

This database will serve as the central component that links all functional modules of the system and ensures accurate, consistent, and secure data flow across the mobile application, backend APIs, and user dashboards.

2.1 Identification of Core Data Entities

The Mobile-Based Attendance Management System is designed around several essential data entities that represent real-world objects and processes. These include:

- **Users**: All participants in the system, including students, instructors, and administrators.
- **Attendance Records**: Logs that track student check-ins for each class session.
- **Courses/Classes**: Represent academic subjects managed by instructors and their associated scheduled sessions.
- **Locations/Geofences**: Define spatial boundaries for validating physical presence during check-in.
- **Facial Recognition Data**: Stores biometric features used to verify student identity.
- **Notifications**: Event-triggered messages sent to users (e.g., absence alerts, check-in confirmations).
- **AuditLog**: Tracks sensitive system actions such as manual overrides, deletions, or configuration changes.

2.2 Data Attributes for Each Entity

Below is a detailed breakdown of fields for each core entity, including data types and constraints:

1. User

Attribute	Type	Constraints	Description
user_id	UUID / INT	Primary Key	Unique user identifier
full_name	VARCHAR	NOT NULL	User's full name
email	VARCHAR	UNIQUE, NOT NULL	Login and communication identifier
password_hash	TEXT	NOT NULL	Encrypted password
role	ENUM ('student', 'instructor', 'admin')	NOT NULL	Role determines access level
profile_picture_url	TEXT	OPTIONAL	Link to profile image
facial_ biometric_hash	TEXT / JSON	NOT NULL	Biometric facial vector for recognition

2. Course

Attribute	Type	Constraints	Description
course_id	UUID / INT	Primary Key	Unique course identifier
course_code	VARCHAR	UNIQUE	Official course code (e.g., CEF440)
course_name	VARCHAR	NOT NULL	Full name of the course
instructor_id	UUID / INT	Foreign Key (User)	Linked instructor

3. ClassSession

Attribute	Type	Constraints	Description
session_id	UUID / INT	Primary Key	Unique identifier for session
course_id	UUID / INT	Foreign Key (Course)	Associated course
scheduled_time	DATETIME	NOT NULL	Start time of the class
geofence_id	UUID / INT	Foreign Key (Geofence)	Location validation link
status	ENUM	NOT NULL	e.g., 'active', 'completed', 'cancelled'

4. Attendance

Attribute	Type	Constraints	Description
attendance_id	UUID / INT	Primary Key	Unique check-in record
student_id	UUID / INT	Foreign Key (User)	Attending student
session_id	UUID / INT	Foreign Key (ClassSession)	Related session
check_in_time	DATETIME	NOT NULL	Time of check-in
check_in_method	ENUM ('facial', 'gps', 'manual')	NOT NULL	Mode of validation
check_in_status	ENUM ('present', 'late', 'absent', 'override')	NOT NULL	Attendance state

5. Geofence

Attribute	Type	Constraints	Description
geofence_id	UUID / INT	Primary Key	Unique boundary ID
session_id	UUID / INT	Foreign Key (ClassSession)	Session it applies to
latitude	DECIMAL	NOT NULL	Center coordinate
longitude	DECIMAL	NOT NULL	Center coordinate
radius_meters	FLOAT	NOT NULL	Allowed distance from center (meters)

6. Notification

Attribute	Type	Constraints	Description
notification_id	UUID / INT	Primary Key	Unique identifier
user_id	UUID / INT	Foreign Key (User)	Recipient
message	TEXT	NOT NULL	Content of notification
type	ENUM ('info', 'warning', 'alert')	NOT NULL	Message classification
created_at	DATETIME	NOT NULL	Time sent

7. AuditLog

Attribute	Data Type	Description
log_id	UUID / INT	Unique ID of log entry
user_id	UUID / INT	User who performed the action
action_type	VARCHAR	Type of action (e.g., "check-in", "override", "delete")
timestamp	DATETIME	Time action was performed
details	TEXT / JSON	Context or parameters of the action

2.3 Data Relationships

The relationships between entities are defined as follows:

Relationship	Type	Description
User ↔ Course	1:N	One instructor teaches many courses
Course ↔ ClassSession	1:N	A course has multiple sessions
ClassSession ↔ Attendance	1:N	One session can have many attendance records
User (student) ↔ Attendance	1:N	A student can have many attendance records
ClassSession ↔ Geofence	1:1	Each session has one geofence boundary
User ↔ Notification	1:N	A user can receive many notifications

2.4 Data Security Requirements

To ensure privacy and system integrity, the following data protection measures must be implemented:

Sensitive Data Identification

- **Biometric Data** (facial_ biometric_hash): High-sensitivity, requires encryption and restricted access.
- **Location Data** (latitude, longitude): Considered sensitive under privacy laws.
- **User Credentials** (email, password_hash): Must be securely stored using cryptographic hashing (e.g., bcrypt).

Compliance Requirements

- Comply with **GDPR** and local data protection laws.
- Provide data deletion and export options for users.
- Ensure all transmissions use **HTTPS** with **TLS encryption**.

Access Control

- Use **role-based access control (RBAC)**:
    - Students can only view their own records.
    - Instructors can view/edit their course attendance.
    - Admins can manage all records.
- Use **authentication tokens** to validate sessions.
- Log all sensitive actions in the **AuditLog** table for traceability.

3.1 Database Requirements Analysis

The relational database is designed to support the core functionalities of the **Mobile-Based Attendance Management System** and seamlessly integrate with the **FastAPI back-end** and mobile frontend. PostgreSQL is selected due to its robust support for constraints, indexing, spatial/geographic data, and JSON fields (for biometrics), making it ideal for handling both structured attendance data and embedded biometric vectors.

Key Functional Requirements Impacting Design

- User authentication and role-based access.
- Real-time recording of student attendance with face and GPS verification.
- Session scheduling and geofence-based location validation.
- Notifications to users and audit logs of sensitive events.

Non-Functional Requirements

- **Performance**: PostgreSQL indexes and query optimization enable check-in within 5 seconds.
- **Scalability**: Relational schema design supports up to 1000 concurrent users and horizontal scaling with read replicas.
- **Security**: PostgreSQL allows for encrypted connections and role-based access at the database level.

3.2 Data Modeling Approach

Chosen Technique: Relational (PostgreSQL)

The database design uses a **normalized relational model**, mapping entities to SQL tables and enforcing data integrity via primary/foreign keys. PostgreSQL's strengths in constraints, JSON support, and indexing make it suitable for secure, structured attendance tracking.

Justification for Relational + FastAPI Approach

- FastAPI integrates well with **SQLAlchemy ORM** or **Tortoise ORM** for managing PostgreSQL schemas.
- Role-based APIs map directly to relational queries e.g., fetching attendance for a student, sessions per course.
- Spatial and biometric data are efficiently handled via PostGIS (for location) and PostgreSQL JSONB fields for embeddings.

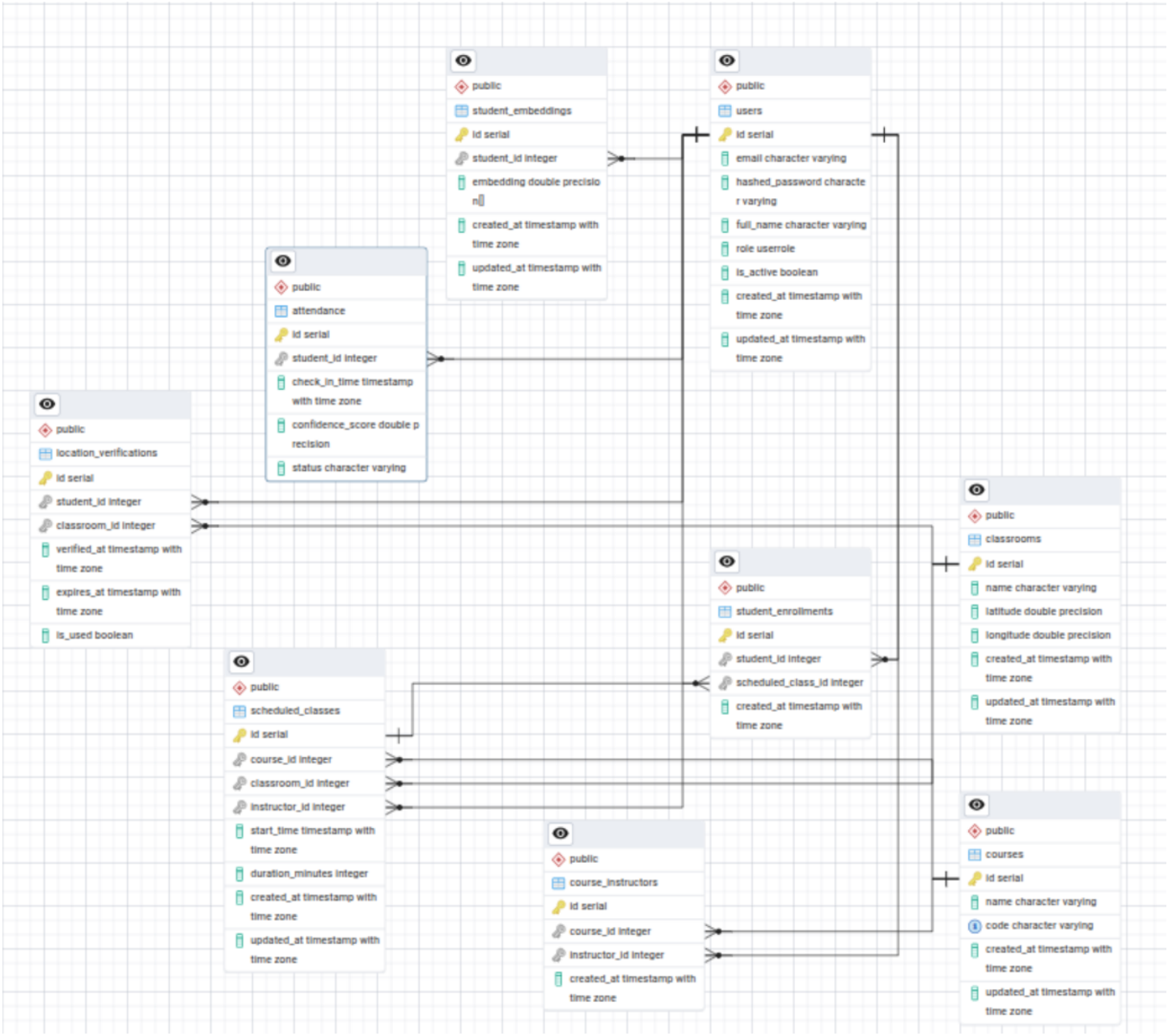
3.3 Entity-Relationship (ER) Model

ER Diagram Summary

Entities and relationships in the PostgreSQL schema:

Entity	Key Attributes	Relationships
User	`user_id (PK)`, `email`, `role`, `facial_embedding`	Has many: Attendance, Notifications
Course	`course_id (PK)`, `course_code`, `instructor_id (FK)`	Instructor teaches many courses
ClassSession	`session_id (PK)`, `course_id (FK)`, `geofence_id (FK)`	Each course has many sessions
Geofence	`geofence_id (PK)`, `latitude`, `longitude`, `radius_meters`	1:1 with ClassSession
Attendance	`attendance_id (PK)`, `student_id (FK)`, `session_id (FK)`	1:1 per student-session
Notification	`notification_id (PK)`, `user_id (FK)`	Sent to users
AuditLog	`log_id (PK)`, `user_id (FK)`	Tracks critical actions

ER Diagram



3.4 Business Rules

Below are rules that govern how the system operates, ensuring integrity, compliance, and usability.

1. Authentication & Roles

- Each user must register with a unique email and password.
- Students must provide facial data at registration.
- Only authorized roles may perform actions (e.g., only instructors can override attendance).

2. Attendance & Check-In

- Students can only check in to active sessions.
- Valid check-in requires successful facial recognition and being within the session's geofence.
- A student can only check in once per session.

3. Data Access

- Students can access only their attendance records.
- Instructors can view data for their courses.
- Admins have full access to manage users, sessions, and configurations.

4. Security & Privacy

- Facial and geolocation data must be encrypted and securely stored.
- All sensitive operations must be logged in the AuditLog.
- Notifications are role-specific and triggered by defined events (e.g., check-in failure, overrides).

3.5 Assumptions and Constraints

Assumptions

- All users (students/instructors) have smartphones with GPS and a front-facing camera.
- Internet connectivity is available during check-in.
- Users consent to facial and location data being used for attendance.
- Institutional support exists for backend deployment and maintenance.

Constraints

- Must comply with GDPR/local data privacy regulations.
- System must perform check-in verification in ≤5 seconds.
- Location accuracy is dependent on GPS signal strength and device hardware.
- The database must handle at least 1000 concurrent users without performance degradation.
- Device cameras must be ≥5MP for reliable facial detection.

3.6 Normalization Process

The database is normalized to **Third Normal Form (3NF)** for consistency, flexibility, and efficient updates.

Normalization Level	Action Taken
1NF	All fields store atomic values only.
2NF	Partial dependencies removed; all non-key attributes depend on whole primary key.
3NF	Removed transitive dependencies (e.g., moved instructor info from session to course).

Justification for Using 3NF

- Maintains data integrity and avoids duplication.
- Supports relational joins in FastAPI endpoints with SQLAlchemy.
- Allows for optimized indexing and query tuning in PostgreSQL.

Trade-Offs

Trade-Off	Mitigation
Joins may slow down frequent queries (e.g., student attendance dashboard)	Use PostgreSQL indexes and ORM-level eager-loading
Slight complexity in API-to-database mapping	Handled with clear Pydantic schemas and FastAPI dependency injection

IV. Database Implementation

The database for the mobile-based attendance system was implemented using PostgreSQL. It is structured to support secure, role-based user management, facial recognition-based check-in, geolocation verification, and scheduled class attendance logging. The design is normalized for consistency and scalability, with clearly defined relationships between entities.

4.1 Key Tables and Thier Roles

1. **users:** Stores all user accounts, including students and instructors. Key fields include; id, email, hashed_password, full_name, role, is_active.
2. **courses:** Holds metadata for each course. Key fields include id, code, name.
3. **course_instructors:** Links instructors to the courses they teach (many-to-many). Key fields include id, course_id, instructor_id.
4. **classrooms:** Defines physical or virtual classrooms with geolocation data for geofencing. Key fields include id, name, latitude, longitude.
5. **scheduled_classes:** Represents specific class sessions with time, location, and instructor. Key fields include id, course_id, classroom_id, instructor_id, start_time, duration_minutes.
6. **student_enrollments:** Associates students with scheduled classes (many-to-many). Key fields include id, student_id, scheduled_class_id.
7. **student_embeddings:** Stores biometric embeddings (e.g., facial vectors) used for recognition. Key fields include id, student_id, embedding, created_at, updated_at.
8. **attendance:** Logs attendance records based on facial recognition and optional geolocation. Key fields include id, student_id, check_in_time, confidence_score, status.
9. **location_verifications:** Stores location-based validation attempts using GPS data. Key fields include id, student_id, classroom_id, verified_at, expires_at, is_used.

4.2 Entity Relationships

- Users may be students or instructors, distinguished by their role.
- Each student can enroll in multiple scheduled_classes via student_enrollments.
- Each course can have multiple instructors through course_instructors.
- Each scheduled_class has one associated classroom and instructor.
- Attendance is linked directly to students and reflects facial verification results.
- Location_verifications tie a student to a classroom at a specific time for GPS-based validation.
- Student_embeddings are uniquely linked to students for facial recognition during check-in.

V. Back-End Implementation

The backend of the mobile-based attendance system was developed using FastAPI, providing a lightweight and high-performance RESTful API for handling facial recognition, geofencing validation, and attendance tracking. The backend is tightly integrated with the PostgreSQL database and includes secure authentication and role-based access control.

5.1 Technology Stack

- Framework: FastAPI (Python 3.12)
- Database ORM: SQLAlchemy with Alembic for migrations
- Facial Recognition: DeepFace
- Geofencing Logic: Haversine formula for distance calculation
- Authentication: JWT-based, with role-specific route protection
- Others: Pydantic (data validation), Passlib (password hashing), Uvicorn (ASGI server)

5.2 Core functionalities

1. User Authentication and Role Management

- JWT tokens are issued on login and used to protect all API endpoints.
- Roles (student, instructor, admin) are embedded in the token for access control.
- Passwords are hashed securely using Passlib (bcrypt).

2. Facial Recognition Check-in

- Student uploads a selfie at check-in time.
- The backend uses DeepFace to:
  - Extract a facial embedding from the uploaded image.
  - Compare it with stored embeddings in the student_embeddings table.
  - Return a confidence score.
- If the confidence score exceeds a set threshold, the student is considered verified.

3. Geolocation Based Validation

- The student's current coordinates (latitude, longitude) are sent along with check-in.
- The backend:
  - Fetches the geolocation of the classroom for the scheduled session.
  - Uses the Haversine formula to calculate the distance between the user and classroom.
  - Validates attendance if the user is within a defined threshold (20 meters).
- Result is logged in the location_verifications table.

4. **Attendance Recording**
   - If both facial recognition and location validation succeed:
   - A new record is created in the attendance table.
   - Includes timestamp, confidence score, and attendance status (present, invalid location, etc.).
   - Attendance is tied to scheduled_class_id via the student's enrollment.
5. **Intructor and Admin APIs**
   - Instructors:
     - View enrolled students and attendance records for their scheduled classes.
   - Admins:
     - Create/update users, courses, classrooms, and scheduled classes.
     - Assign instructors to courses and manage embeddings.

5.3 API Structure and Overview

The API is organized under versioning (/api/v1/) and grouped by functionality: authentication, user management, attendance, and classroom geolocation services.

1. **Authentication**

- POST /api/v1/auth/register: Register a new user (student or instructor).
- POST /api/v1/auth/login: Authenticate user credentials and return a JWT token.

2. **Users**

- GET /api/v1/users/: Retrieve a list of all users (admin only).
- GET /api/v1/users/me: Retrieve details of the currently authenticated user.
- GET /api/v1/users/{user_id}: Retrieve a specific user by ID.
- PUT /api/v1/users/{user_id}: Update a user's details.
- DELETE /api/v1/users/{user_id}: Delete a user from the system.

3. **Attendance (Facial Recognition)**

- POST /api/v1/attendance/admin/upload-face: Upload a student's facial embedding to the system (admin functionality, used for enrollment or re-enrollment).
- POST /api/v1/attendance/student/check-in: Student check-in using facial recognition and optional geolocation.

4. **Classroom (Geolocation)**

- POST /api/v1/classroom/admin/set-class-boundary: Define or update geofence boundaries (latitude, longitude, radius) for a classroom.
- POST /api/v1/classroom/student/verify-location: Student submits GPS coordinates for location validation during check-in.

5.4 Error Handling and Validation

- All APIs use structured error responses with HTTP status codes.
- Pydantic models ensure strict input validation.
- Invalid facial matches or out-of-range geolocations return 403 Forbidden or 422 Unprocessable Entity responses with clear messages.

This API structure ensures separation of concerns, clear role-based access, and scalability. Each route includes proper validation, response formatting, and secure access control using JWT authentication middleware.

VI. Connecting Database to Back-End

The connection between the FastAPI backend and the PostgreSQL database is handled using SQLAlchemy with a configuration defined in config.py and initialized through database.py. This setup ensures reusable, maintainable, and environment-specific connections using dependency injection and robust session management.

6.1 Configuration (config.py)

All environment-specific and connection-related settings are centralized in config.py using Pydantic's BaseSettings, which supports .env file loading for secrets and database credentials.

```
# config.py (excerpt)

class Settings(BaseSettings):
    POSTGRES_SERVER: str = "localhost"
    POSTGRES_USER: str = "postgres"
    POSTGRES_PASSWORD: str = "angeloboy"
    POSTGRES_DB: str = "attendance_db"

    SQLALCHEMY_DATABASE_URI: str = f"postgresql://{POSTGRES_USER}:{POSTGRES_PASSWORD}@{POSTGRES_SERVER}/{POSTGRES_DB}"
```

This allows for seamless modification of credentials or connection targets by simply updating environment variables or the .env file without touching code.

6.2 Database Initialization (database.py)

The database.py file sets up the database engine and session handling:

```
# database.py

engine = create_engine(settings.SQLALCHEMY_DATABASE_URI, pool_pre_ping=True)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()
```

- **engine:** Initializes the SQLAlchemy engine using the configured connection URI.
- **SessionLocal:** A session factory to handle transactions for each request.
- **Base:** The declarative base used for defining ORM models.

6.3 Database Injection

To ensure each API request gets a clean and isolated database session, FastAPI's dependency injection system is used via the get_db() function:

```
# Dependency
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

In FastAPI routes, this dependency is injected like so:

```
from fastapi import Depends
from app.db.database import get_db
from sqlalchemy.orm import Session

@app.get("/api/v1/users/")
def read_users(db: Session = Depends(get_db)):
    return db.query(User).all()
```

This pattern ensures:

- Each request gets its own transactional session.
- Sessions are properly closed even if an exception occurs.
- Business logic stays decoupled from session management.

6.4 Model Integration

Each SQLAlchemy model (e.g., User, Attendance, Classroom) inherits from Base, enabling automatic schema mapping when migrations are run (typically using Alembic).

Example model declaration:

```
class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    role = Column(String, nullable=False)
```

This modular, dependency-injected architecture enables efficient, scalable, and clean database interactions across the entire FastAPI application.

VII. Validation and Testing

To ensure the reliability, correctness, and security of the mobile-based attendance system, both input validation and automated testing were integral parts of the backend development lifecycle. This section outlines how validation and testing were implemented across the system

7.1 Input Validation

Input validation was handled using Pydantic, which is tightly integrated with FastAPI. It ensures all incoming data (from users, GPS, or face uploads) is strictly validated before any database operations occur.

❖ Examples of Input Validation

1. User Registration:
   - ➢ Validates email format, password length, and role type.
   - ➢ Rejects missing or malformed fields automatically.
2. Check-In Endpoint:
   - ➢ Verifies that an image is uploaded.
   - ➢ Confirms latitude and longitude are valid float values within acceptable range.
3. Classroom Boundary Setup:
   - ➢ Validates geofence radius and coordinates.

❖ Benefits

- ➢ Prevents SQL injection, type mismatches, and malformed payloads.
- ➢ Reduces logic bugs and ensures API robustness.

7.2 Authentication and Authorization Testing

Role-based access control was enforced using JWT tokens, and extensively tested to ensure:

- ➢ Unauthorized users cannot access protected endpoints.
- ➢ Students cannot access admin or instructor endpoints.
- ➢ Expired or invalid tokens return appropriate 401/403 errors.

Unit tests were written to simulate token generation, expiry, and misuse scenarios.

7.3 Automated Testing

The backend supports automated testing using pytest and httpx (or FastAPI's TestClient). Tests were grouped by module and covered the following:

1. User and Auth Tests:
   - Register new user
   - Login and token issuance
   - Invalid credentials and duplicate registration
2. Attendance Tests:
   - Valid face image check-in
   - Rejecting unrecognized faces
   - Location check validation using mocked GPS coordinates
3. Database Integrity:
   - Setup/teardown using in-memory SQLite or test PostgreSQL DB
   - Verifying cascading deletes (e.g., when a user is deleted, related embeddings are handled)

7.4 Manual and Functional Testing

- Manual testing was done using Postman to simulate real-world scenarios (e.g., student checking in late or outside geofence).
- Edge cases like uploading blurry images or checking in from a different location were tested to ensure graceful error handling.

VIII. Challenges and Resolutions

During the development of the mobile-based attendance system, several technical and design challenges emerged. This section highlights those obstacles and how they were effectively addressed.

1) Facial Recognition Accuracy in Diverse Conditions

Challenge:

- DeepFace sometimes failed to identify students due to:
  - Low-quality or poorly lit images.
  - Variations in facial appearance (e.g., glasses, masks, hairstyles).

Resolution:

- Implemented preprocessing constraints (e.g., enforcing front-facing photos).
- Increased matching threshold tolerance slightly for improved reliability.
- Suggested storing multiple facial embeddings per student for better accuracy (future improvement).

2) Geolocation Drift and Inaccuracy

Challenge:

- GPS readings from mobile devices occasionally showed slight drift (±10–30 meters), causing legitimate students to be marked as out-of-bound.

Resolution:

- Applied the Haversine formula with a buffer radius (20 meters).
- Added an endpoint (/classroom/student/verify-location) to allow students to test their GPS location before check-in.
- Educated users (via mobile frontend) to enable high-accuracy location mode.

3) Combining Facial Recognition and Geofencing in Realtime

Challenge:

- Ensuring that both facial recognition and geolocation validation occurred atomically during the check-in process, without race conditions or partial success.

Resolution:

- Implemented a combined /attendance/student/check-in endpoint to handle both verifications sequentially with unified response messages.
- Used transaction handling in SQLAlchemy to roll back attendance records if either check failed

4) Data Consistency across linked models

Challenge:

- Maintaining referential integrity between users, scheduled classes, facial embeddings, and attendance records, especially when modifying or deleting users.

Resolution:

- Enforced foreign key constraints in the PostgreSQL schema.
- Added cascading delete or nullification logic using SQLAlchemy relationships.
- Used Alembic migrations to ensure DB schema remained in sync with models.

5) Handling Large Image Uploads in API

Challenge:

- Mobile selfies exceeded default request size limits or caused memory issues on the backend.

Resolution:

- Configured FastAPI and Uvicorn to accept multipart form data efficiently.
- Added size and format validation for uploaded images.
- Compressed or resized images server-side before processing with DeepFace.

6) Security: JWT Token Management and Role Abuse

Challenge:

- Preventing unauthorized access to admin and instructor functionalities.

Resolution:

- Embedded user roles directly in JWT payload.
- Used FastAPI's dependency injection to enforce route-level role checks.
- Added test cases to simulate unauthorized access and ensure proper 403 responses.

IX. Conclusion

This task successfully achieved the design and implementation of a relational database system tailored for the **Mobile-Based Attendance Management System Based on Geofencing and Facial Recognition**. The database structure was carefully modeled to reflect real-world academic processes and meet the specific functional and non-functional requirements outlined in the Software Requirements Specification (SRS).

Summary of Achievements

- **Identified and defined** all core entities including Users, Courses, ClassSessions, Attendance, Geofences, Notifications, and AuditLogs.
- **Developed a normalized relational model** using PostgreSQL, structured for efficient data storage, fast retrieval, and integrity.
- **Constructed a detailed Entity-Relationship Diagram (ERD)** illustrating all entities, attributes, relationships, and cardinalities.
- **Integrated data security measures** for sensitive information such as facial embeddings and geolocation data, in compliance with GDPR.
- **Prepared for back-end integration** using FastAPI and PostgreSQL-compatible ORMs, enabling seamless connectivity between APIs and the database layer.

Support for SRS Requirements

The database design fully supports the SRS by:

- Enabling **real-time attendance recording** through facial recognition and GPS validation.
- Ensuring **role-based access** and record filtering (by student, course, session).
- Maintaining **audit trails and logs** for all sensitive or override operations.
- Supporting **notification delivery** and attendance status monitoring.

Summary of Key Design Decisions

- Chose **PostgreSQL** for its robust support for indexing, constraints, and JSON handling (biometric data).
- Applied **3NF normalization** to minimize redundancy and enforce data consistency.
- Incorporated **foreign key constraints** and **enumerated types** to encode business logic.
- Structured the schema to optimize for **FastAPI-based access**, including scalable query patterns and ORM mapping.

Next Steps in Project Development

1. **Back-End API Implementation**
    - Build RESTful endpoints using FastAPI (e.g., /check-in, /get-attendance, /sessions/today).
    - Implement role-based access control and token-based authentication (JWT or Firebase Auth).
2. **Front-End Integration**
    - Connect the mobile app to the API endpoints for registration, login, check-in, and dashboards.
3. **Testing and Validation**
    - Conduct unit, integration, and performance testing on database operations and APIs.
4. **Deployment and Hosting**
    - Deploy PostgreSQL on a cloud platform (e.g., Heroku, Supabase, or AWS RDS) and integrate with cloud-based FastAPI back end.
5. **Monitoring and Analytics**
    - Set up monitoring tools (e.g., pgAdmin, Prometheus) and prepare attendance analytics dashboards.

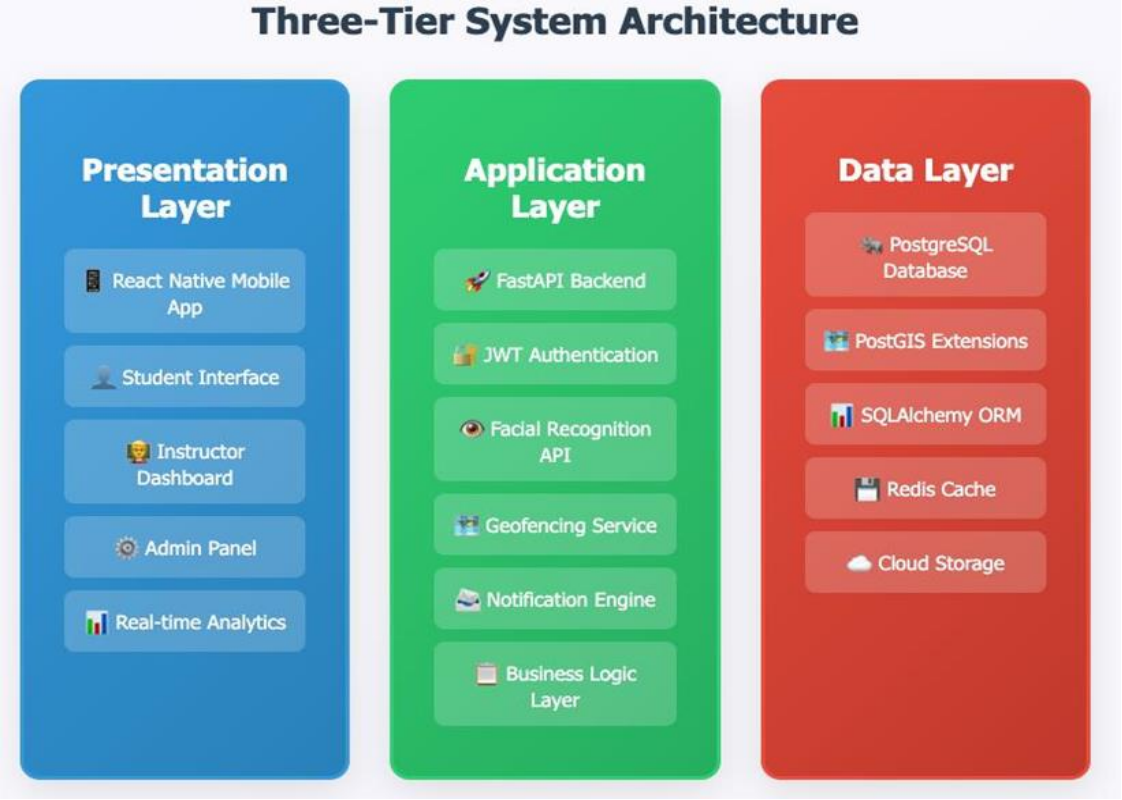
11.1 Glossary

Term	*Definition*
ERD	Entity-Relationship Diagram – a visual representation of data entities and their relationships.
DBMS	Database Management System – software that enables the creation, querying, updating, and administration of databases.
CRUD	Create, Read, Update, Delete – the four basic operations of persistent storage.
UUID	Universally Unique Identifier – a 128-bit number used to uniquely identify information.
Geofence	A virtual boundary defined by geographic coordinates and a radius, used to trigger location-based actions.
Facial Biometric Hash	A secure, machine-readable representation (e.g., vector or embedding) of a user's facial features.
Foreign Key	A field in one table that uniquely identifies a row in another table, establishing a relationship between the two.
Normalization	A process of organizing data to reduce redundancy and improve integrity in relational databases.

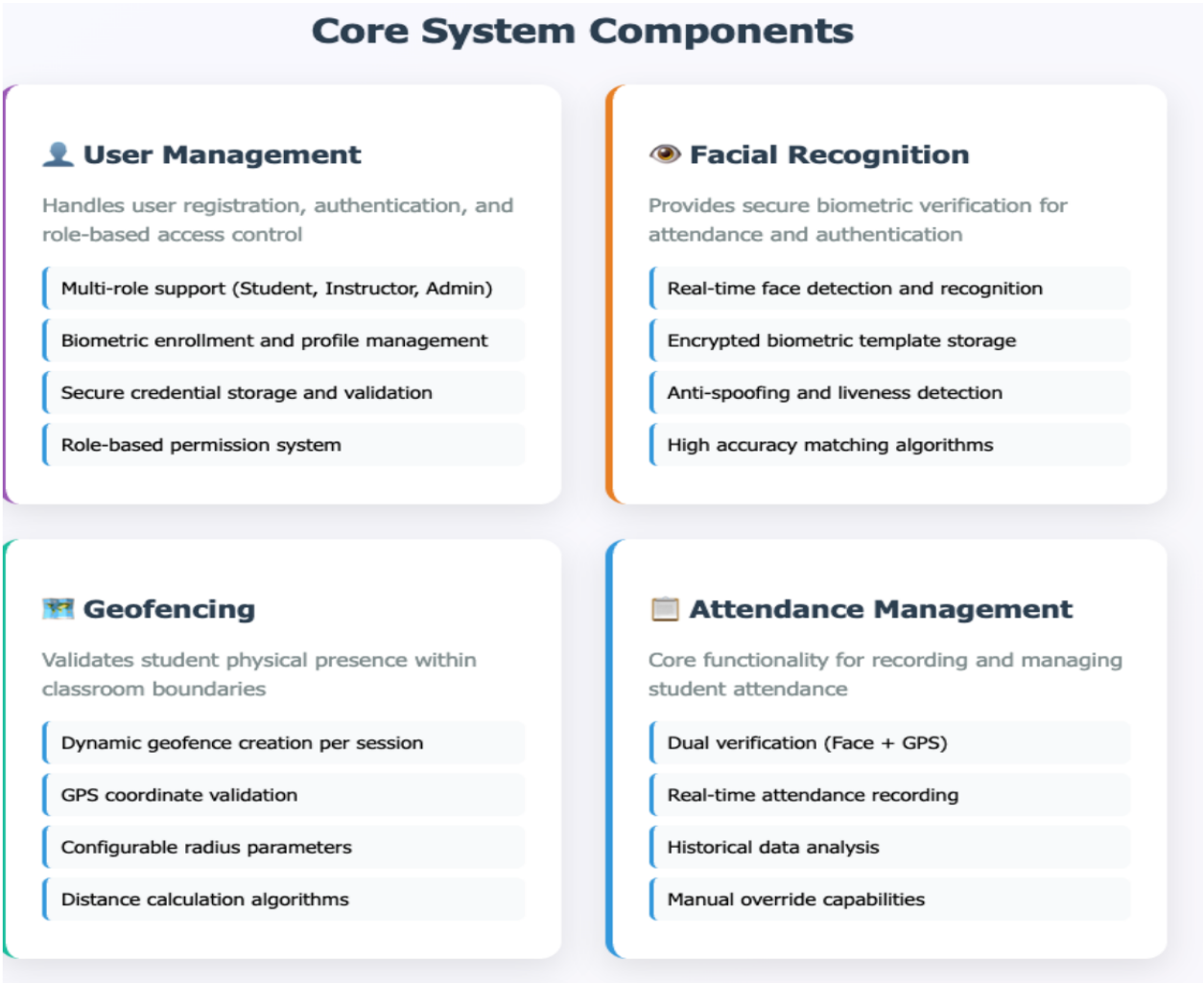
11.2 Tools and Technologies Used

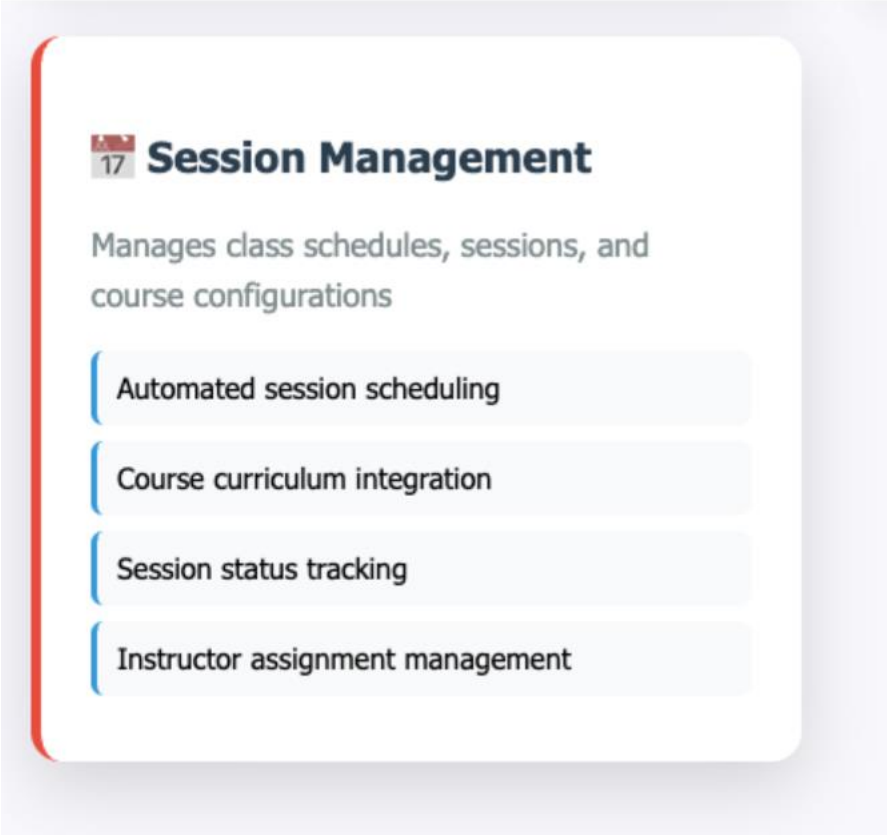
Tool/Technology	*Purpose*
PostgreSQL	Chosen DBMS for implementing the database schema.
FastAPI	Backend framework to integrate with the database. Security protocols for user authentication and API access. Used for user sign-in and token-based authentication. Used to send push notifications to user devices. Used to define and validate geofence boundaries. API testing tool for validating backend integration with the database.

11.3 Visual Representation of System Architecture



11.4 Visual Core Components




Session Management
Manages class schedules, sessions, and course configurations
Automated session scheduling
Course curriculum integration
Session status tracking
Instructor assignment management