UNIVERSITY OF BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY

DERPARTMENT OF COMPUTER ENGINEERING

TASK 6: Database and Backend Implementation

COURSE CODE: CEF 440

COURSE TITLE: INTERNET PROGRAMMING and MOBILE PROGRAMMING

COURSE INSTUCTOR: DR NKEMENI Valery

BY GROUP 17

DATE: June 2025

SN	NAME	MATRICULE
1	AKENJI FAITH SIRRI	FE22A142
2	DYL PADARAN AMBE MUNJO	FE22A193
3	KONGNYU DESCHANEL	FE22A234
4	NDI BERTRAND	FE22A252
5	NDOUKIE EBOKE BLANDINE	FE22A254

Table of Contents

Database Design and Implementation

1. Data Elements Analysis

1.1 Theoretical Foundation of Data Element Identification

Data elements represent the fundamental building blocks of any database system. In the context of our mobile-based attendance management system, data elements are derived from the functional requirements specified in the SRS document. According to Elmasri and Navathe's database theory, data elements should be atomic, relevant, and directly support the system's business logic.

1.2 User Management Data Elements

The system requires comprehensive user management to support role-based access control (RBAC). The theoretical foundation for this approach stems from the principle of least privilege in information security. Each user entity must contain sufficient data to:

- **Identity Authentication**: Username and password fields support the authentication process, following standard security practices where passwords are stored as hashed values using algorithms like bcrypt or Argon2.
- **Role-Based Authorization**: The role field (Student, Educator, Administrator) implements the RBAC model, ensuring users can only access functionalities appropriate to their organizational position.
- **Audit Trail**: Registration date and status fields provide temporal tracking and account lifecycle management.

1.3 Academic Domain-Specific Data Elements

Student Profile Data

The student-specific data elements reflect the academic context of the system. The matriculation number serves as a unique business identifier separate from the technical primary key, following database normalization principles. This separation allows for system flexibility while maintaining academic record integrity.

Data Element	Data Type	Description
`user_id`	UUID / String	Unique identifier for each user
`full_name`	String	User’s official name
`email`	String	Used for authentication and communication
`password_hash`	String (Hashed)	Encrypted password for secure login
`role`	Enum (`student`, `educator`, `admin`)	Determines user access level
`profile_picture`	Image (Base64/URL)	Optional profile photo used for identification

Course Management Data

Course data elements support the educational workflow defined in FR13. The course code serves as a natural key for academic identification, while the technical primary key ensures database efficiency. Credit hours and semester information support academic calendar integration and workload calculation.

Data Element	Data Type	Description
`course_id`	String	Unique course identifier
`course_name`	String	Full name of the course
`course_code`	String	Short course code (e.g., CEF 440)
`lecturer_id`	String	Links course to assigned lecturer
`schedule`	JSON / Object	Contains course days and time slots

Session Management Data

Class session data elements enable temporal tracking of educational activities. The distinction between scheduled time and actual start time addresses the real-world scenario identified in the feasibility study (Section 6.1) where lectures often begin late. This dual-time approach ensures accurate attendance calculation based on actual class duration rather than scheduled duration.

Data Element	Data Type	Description
`attendance_id`	UUID / String	Unique ID for the attendance record
`user_id`	String	Links the record to a student
`course_id`	String	Links attendance to a course
`date`	Date	Date of the class session
`check_in_time`	Time	Actual check-in time
`status`	Enum (`Present`, `Absent`, `Excused`, `Revoked`)	Attendance status
`recorded_by`	String	Lecturer ID who validated the entry

1.4 Biometric and Location Data Elements

Facial Recognition Data

The facial recognition data elements implement biometric authentication as specified in FR2. The face encoding field stores mathematical representations of facial features rather than actual images, addressing privacy concerns while enabling identification. The confidence threshold provides a tunable parameter for balancing security and usability.

Data Element	Data Type	Description
`face_id`	String / Hash	Encrypted representation of facial data
`face_image`	Image	Captured facial image during check-in
`verification_status`	Boolean	Indicates if the face matched a registered user
`timestamp`	DateTime	Time the face was scanned

Geofencing Data

Geofencing data elements implement location-based verification (FR4). The latitude, longitude, and radius fields define circular geofences using the World Geodetic System (WGS84) coordinate system. This approach provides adequate accuracy for classroom-level location verification while maintaining computational efficiency.

Data Element	Data Type	Description
`geofence_id`	String	Unique ID for geofence area
`name`	String	Friendly name of location
`center_latitude`	Float	Latitude of geofence center
`center_longitude`	Float	Longitude of geofence center
`altitude`	float	Height of venue (geofence) above sea level

2. Conceptual Design

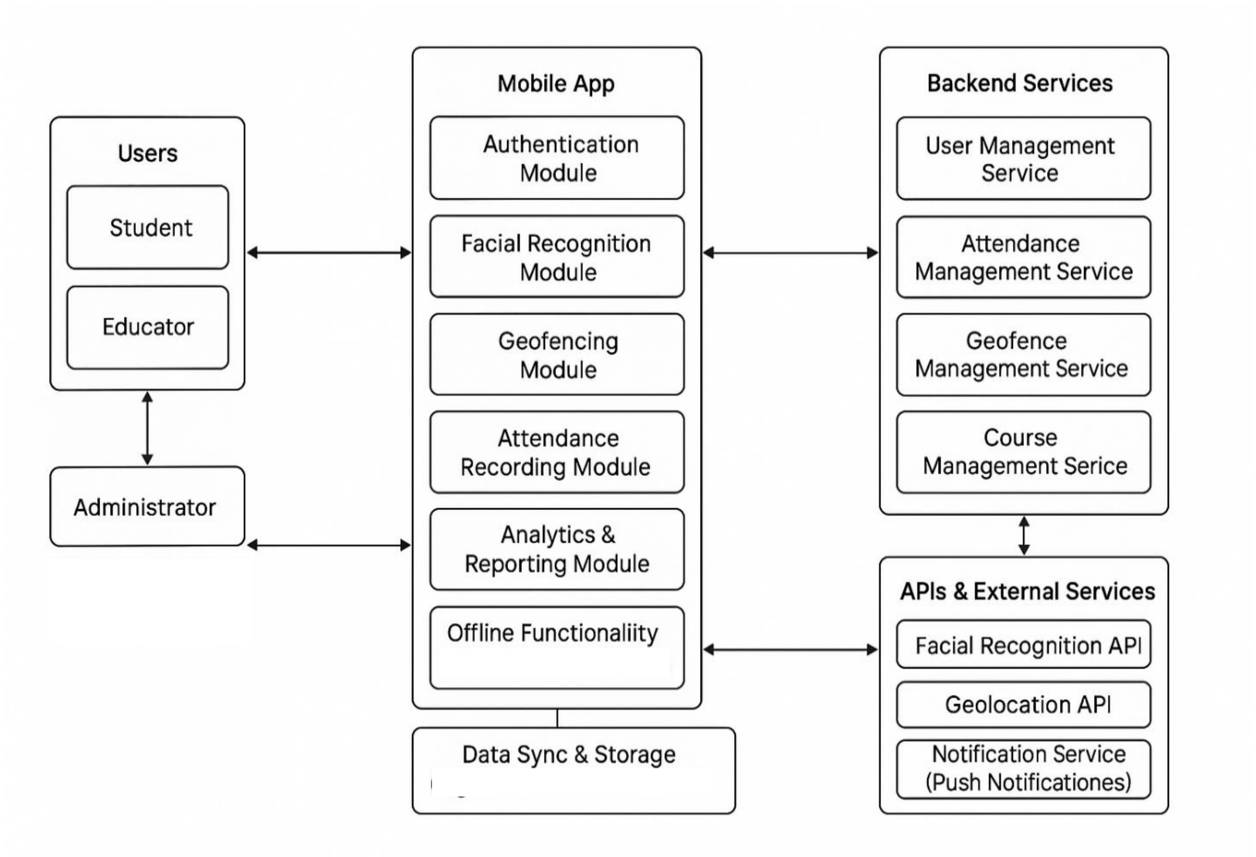


Figure 0-1: System Conceptual Design

The conceptual design provides a high-level view of data relationships without implementation details. This abstraction level is crucial for stakeholder communication and system understanding.

2.1 Data Flow Architecture

The conceptual design implements a layered data flow architecture:

1. Users

This layer represents the primary system actors:

- **Students**: Use the mobile app to register their attendance. Their location and facial identity are verified during attendance.

- **Educators (Lecturers)**: Use the app to monitor attendance, initiate facial recognition, and manage classes.
- **Administrators**: Manage users, geofences, courses, and system configurations.

2. Mobile App (Client-Side)

The mobile application is the core interface through which all users interact with the system. It is composed of the following functional modules:

Authentication Module: Verifies user credentials for login and role-based access (student, lecturer, admin).

Facial Recognition Module: Activates the device's camera to capture student images.

Geofencing Module: Uses GPS to detect whether the student is within the predefined classroom boundary, works with the Geolocation API and syncs location data in real-time or offline.

Attendance Recording Module: Combines facial recognition and geofence verification results, records and timestamps successful check-ins/check-outs, supports manual overrides by lecturers.

Analytics & Reporting Module: Provides users with attendance statistics.

Data Sync & Storage: Ensures local data is synchronized with the backend when internet connectivity is restored.

3. Backend Services

These services provide the logic, processing, and data management needed for the app to function:

User Management Service: Enforces role-based permissions.

Attendance Management Service: Maintains attendance logs, supports validation, analytics, and manual overrides.

Geofence Management Service: Enables administrators to define and update geofenced locations and stores geofence metadata (coordinates, radius, classroom ID).

Course Management Service: Supports course creation, registration, and lecturer assignment, syncs course schedules with attendance tracking modules.

4. APIs & External Services

These are third-party tools and APIs that enhance the system's capabilities:

Facial Recognition API: Processes captured facial data and returns identity verification results.

Geolocation API: Converts raw GPS data into location coordinates for geofencing logic, enables real-time movement tracking and validation.

Notification Service: Sends push alerts for class start/end times, attendance confirmations, or missed check-ins.

3. ER Diagram Theoretical Analysis

The ERD serves as a blueprint for the database, illustrating the various entities (tables), their attributes (columns), and the relationships between them. This design is foundational for developing a system that effectively manages user information, course enrollment, attendance tracking, and communication.

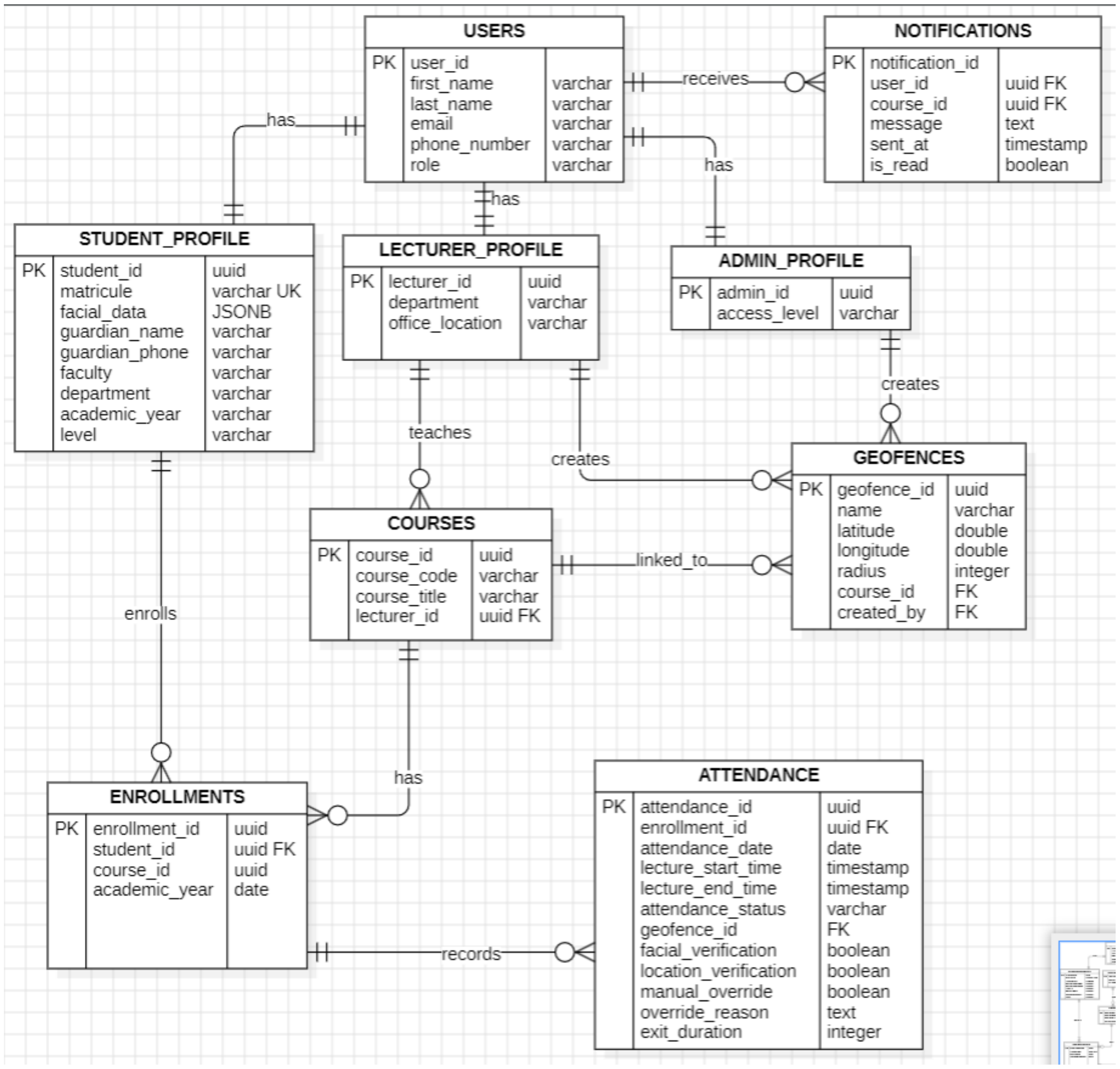


Figure 0-2: ER Diagram

3.1 Relationship Cardinality Analysis

The ER diagram implements specific cardinality relationships based on business rules:

One-to-Many Relationships

- **Users to Student Profiles**: Each user can have at most one student profile, but the system allows for future expansion to support multiple roles per user

- **Courses to Class Sessions**: Each course can have multiple sessions across different time periods
- **Class Sessions to Attendance Records**: Each session generates multiple attendance records, one per enrolled student

Many-to-Many Relationships

- **Students to Courses**: Implemented through the Course Enrollments associative entity, supporting student registration in multiple courses and courses having multiple enrolled students
- **Educators to Courses**: Implemented through Course Assignments, allowing team teaching and multiple course assignments per educator

3.2 Entity Classification

- **Strong Entities**: Users, Courses, Geofences exist independently and have their own primary keys
- **Weak Entities**: Student Profiles, Educator Profiles depend on Users for existence
- **Associative Entities**: Course Enrollments, Attendance Records represent many-to-many relationships

3.3 Integrity Constraints

Referential Integrity

Foreign key relationships ensure data consistency across related tables. Cascading delete operations maintain referential integrity when primary entities are removed.

Domain Constraints

Enumerated types (ENUM) restrict field values to valid options, preventing data inconsistency. For example, attendance status is limited to 'present', 'absent', 'late', or 'excused'.

Business Rule Constraints

Unique constraints on business identifiers (matriculation numbers, course codes) prevent duplicate academic records while allowing technical primary keys for database optimization.

4. Database Implementation

Before the implementation of the database, it is important to enhance our ER diagram and to make sure normalization has been done.

4.1 Database Normalization

4.1 Why We Normalize Databases

Database normalization is a systematic approach to organizing data that serves several critical purposes in the attendance management system:

Eliminate Data Redundancy

Problem: Without normalization, the same data is stored in multiple places, leading to storage waste and maintenance challenges.

Example: In an unnormalized attendance system, course information (course name, instructor, credits) might be repeated in every attendance record, enrollment record, and grade record.

Solution: Store course information once in a dedicated courses table, referenced by other tables through foreign keys.

Prevent Data Anomalies

Normalization prevents three types of destructive anomalies:

Update Anomalies

- **Problem**: When data is duplicated, updating one instance might leave others unchanged, creating inconsistencies
- **Example**: If a course name changes and it's stored in 100 attendance records, failure to update all instances results in inconsistent course names across the system
- **Impact**: Reports showing different names for the same course, confusion for users, and loss of data integrity

Insert Anomalies

- **Problem**: Unable to add certain data without having other unrelated data
- **Example**: Cannot add a new course to the system until at least one student enrolls and has an attendance record
- **Impact**: Prevents proactive course setup and planning

Delete Anomalies

- **Problem**: Deleting a record removes more information than intended
- **Example**: Removing the last attendance record for a course might delete all course information
- **Impact**: Unintentional loss of valuable academic data

Ensure Data Integrity

- **Referential Integrity**: Foreign key relationships ensure related data remains consistent across tables
- **Domain Constraints**: Each attribute has a defined set of valid values, preventing invalid data entry
- **Business Rules**: Database constraints enforce organizational policies automatically

Improve System Maintainability

- **Single Source of Truth**: Each piece of information exists in exactly one location
- **Simplified Updates**: Changes to course information, user details, or business rules require modification in only one place
- **Reduced Human Error**: Elimination of redundancy reduces opportunities for data entry mistakes

4.2 Transformation from Basic ER to EER Model

The Enhanced Entity-Relationship (EER) model extends the basic ER diagram by incorporating advanced modeling concepts that better represent the complexity of the attendance management system. This transformation addresses limitations in the basic ER model and provides a more comprehensive representation of the system's data requirements.

4.2.1 Significance of EER Model Implementation

Enhanced Semantic Representation

The EER model provides superior semantic clarity by explicitly representing:

- **Role-based Access Control**: Different user types have distinct database permissions and application functionalities
- **Inheritance Relationships**: Common attributes are inherited rather than duplicated, reducing redundancy
- **Constraint Specification**: Business rules are more precisely defined through specialization constraints

Database Design Optimization

The EER model directly influences physical database implementation through:

- **Table Inheritance Strategies**: Three implementation approaches are considered:
    1. **Single Table Inheritance**: All user types in one table with discriminator column
    2. **Class Table Inheritance**: Separate tables for each subclass with foreign key relationships
    3. **Concrete Table Inheritance**: Complete separate tables for each user type
- **Query Optimization**: Specialized queries can target specific user types without unnecessary joins
- **Maintenance Efficiency**: Schema modifications affect only relevant user types rather than all users

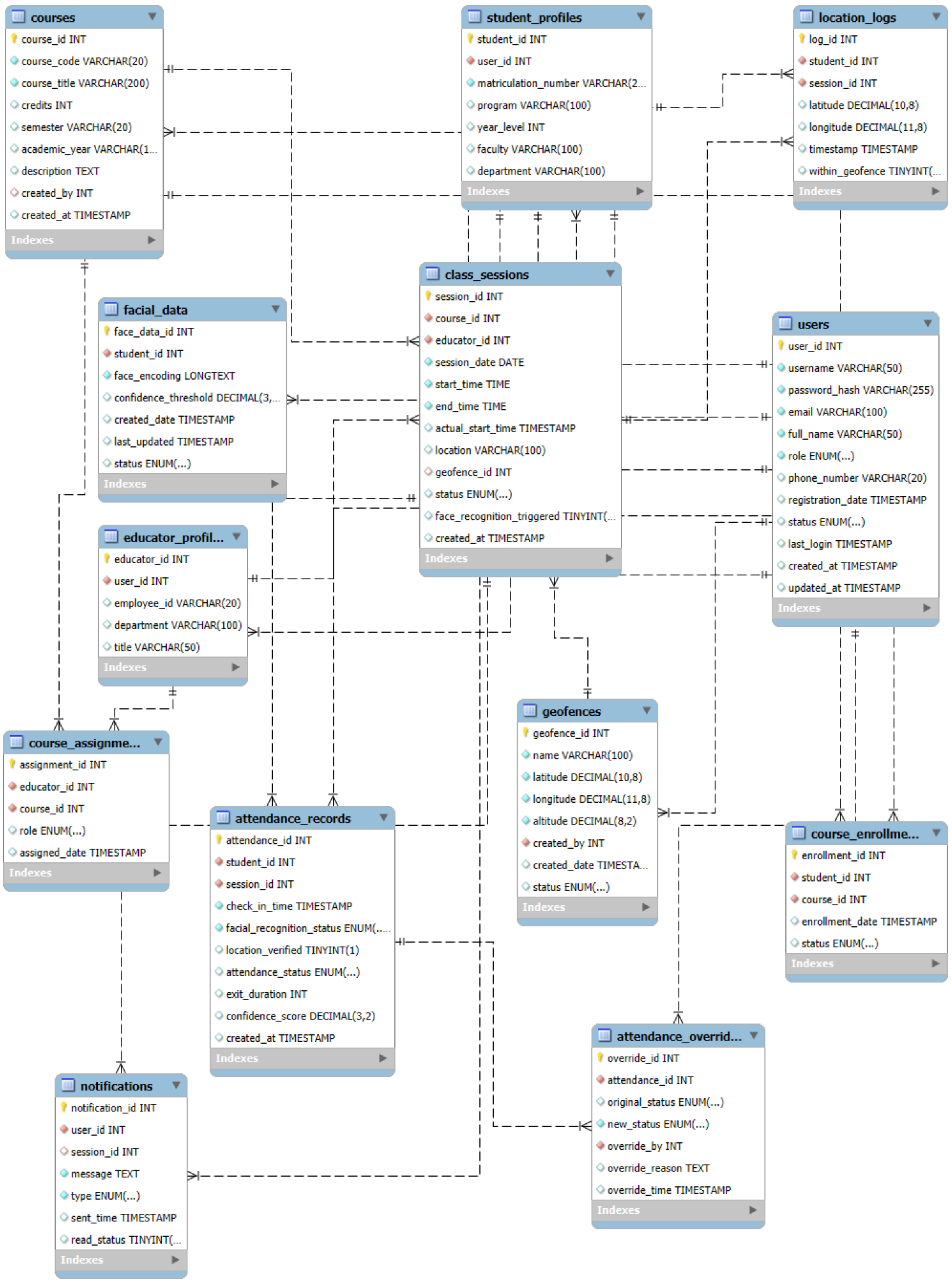


Figure 0-1: Enhanced ER Diagram

4.3 Hybrid Database Architecture

We can now go ahead and implement our database. The implementation employs a polyglot persistence approach, utilizing both SQL and NoSQL databases for optimal performance and functionality:

SQL Database Selection (MySQL)

Relational databases excel at handling structured data with complex relationships. The attendance management system's academic data (courses, enrollments, grades) exhibits strong relational characteristics requiring ACID properties:

- **Atomicity**: Attendance recording operations must complete entirely or not at all
- **Consistency**: Academic records must maintain referential integrity
- **Isolation**: Concurrent attendance submissions must not interfere with each other
- **Durability**: Academic records must persist despite system failures

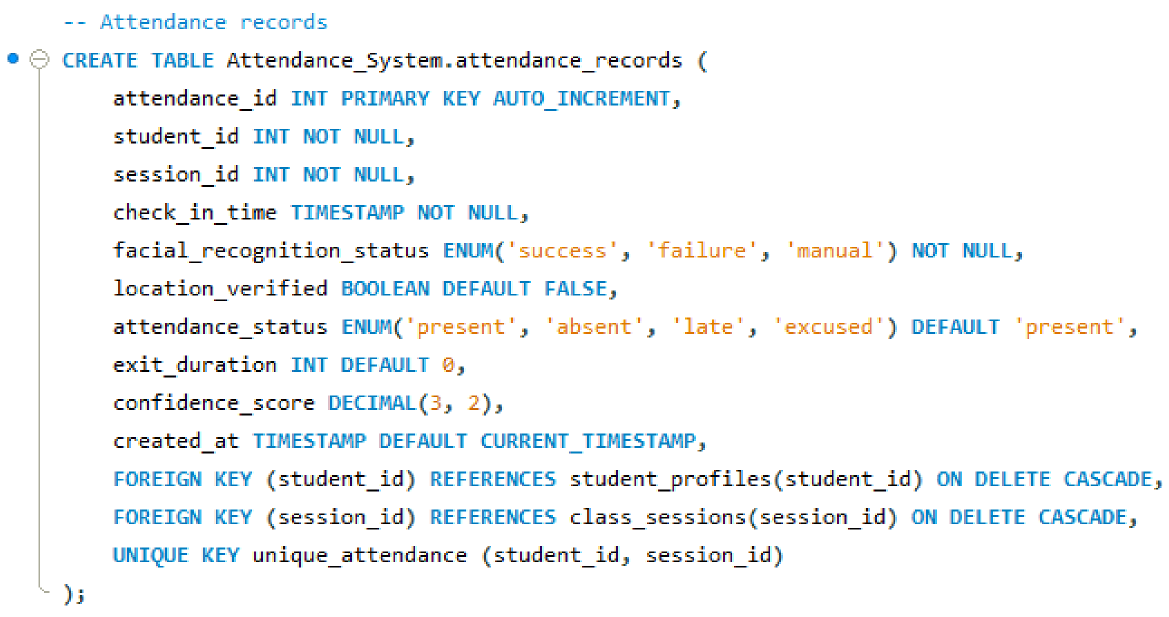
4.4 Database Schema Design Principles

Logical Schema Design

The logical schema translates the conceptual ER model into database-specific structures while maintaining platform independence. This involves:

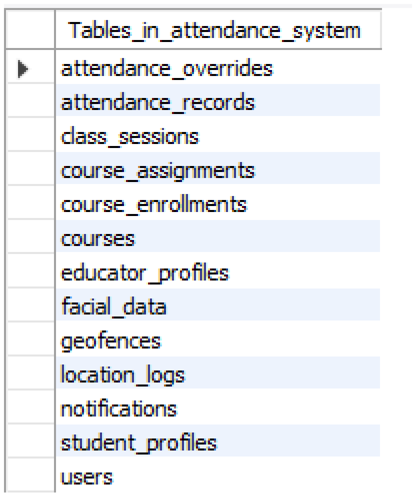
- **Table Structure Definition**: Each entity becomes a table with appropriate column definitions
- **Relationship Implementation**: Foreign keys implement entity relationships
- **Constraint Definition**: Business rules become database constraints

An example of a table created is the attendance record table as shown below. Similar SQL commands were used to create all other tables of the database.



```sql
-- Attendance records
CREATE TABLE Attendance_System.attendance_records (
    attendance_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT NOT NULL,
    session_id INT NOT NULL,
    check_in_time TIMESTAMP NOT NULL,
    facial_recognition_status ENUM('success', 'failure', 'manual') NOT NULL,
    location_verified BOOLEAN DEFAULT FALSE,
    attendance_status ENUM('present', 'absent', 'late', 'excused') DEFAULT 'present',
    exit_duration INT DEFAULT 0,
    confidence_score DECIMAL(3, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (student_id) REFERENCES student_profiles(student_id) ON DELETE CASCADE,
    FOREIGN KEY (session_id) REFERENCES class_sessions(session_id) ON DELETE CASCADE,
    UNIQUE KEY unique_attendance (student_id, session_id)
);
```

Figure 0-2: Attendance Record Table creation



Tables_in_attendance_system
attendance_overrides
attendance_records
class_sessions
course_assignments
course_enrollments
courses
educator_profiles
facial_data
geofences
location_logs
notifications
student_profiles
users

Figure 0-3: All Database Tables

Physical Schema Design

Physical design optimizes database performance through:

- **Index Strategy**: Strategic indexing on frequently queried columns (username, course codes, session dates)

- **Partitioning**: Large tables (attendance records) can be partitioned by academic year or semester
- **Storage Optimization**: Appropriate data types minimize storage requirements while maintaining precision

4.5 Security Implementation Theory

Data Encryption

Sensitive data fields implement field-level encryption:

- **Password Security**: Bcrypt hashing with salt provides one-way encryption for authentication credentials
- **Biometric Data Protection**: Facial encodings are encrypted at rest using AES-256 encryption
- **Personal Information**: Student identification numbers and contact information require encryption for privacy compliance

5. Backend Implementation

5.1 API Design Architecture

The backend implements FASTAPI principles. FastAPI is a modern, high-performance web framework for building APIs with Python 3.7+ based on standard Python type hints. It is designed to create RESTful APIs quickly and efficiently, with a focus on performance, ease of use, and developer productivity.

- **Resource-Based URLs**: Each endpoint represents a specific resource (users, courses, attendance)
- **HTTP Method Semantics**: GET for retrieval, POST for creation, PUT for updates, DELETE for removal
- **Stateless Communication**: Each request contains all necessary information for processing
- **Standard Status Codes**: HTTP status codes communicate operation results consistently

Microservices Considerations

While the current implementation uses a monolithic architecture, the modular design supports future microservices migration:

- **Authentication Service**: User management and token generation
- **Attendance Service**: Facial recognition and location verification
- **Course Management Service**: Academic data and enrollment handling
- **Notification Service**: Communication and alert distribution

5.2 Data Processing Algorithms

Facial Recognition Processing

The system implements facial recognition using established computer vision algorithms:

- **Face Detection**: Haar cascades or deep learning models identify facial regions in images
- **Feature Extraction**: Algorithms like Local Binary Patterns (LBP) or Convolutional Neural Networks extract facial features
- **Face Matching**: Euclidean distance calculations compare extracted features with stored encodings
- **Confidence Scoring**: Statistical measures provide recognition reliability metrics

Geofencing Calculations

Location verification employs geographical computation:

- **Coordinate System**: GPS coordinates use the WGS84 datum for global compatibility
- **Distance Calculation**: Haversine formula calculates spherical distances between GPS points
- **Boundary Detection**: Circular geofences provide simple yet effective location verification
- **Accuracy Considerations**: GPS accuracy limitations (±3-5 meters) influence geofence radius selection

5.3 Real-Time Processing Requirements

Synchronous Operations

Critical operations require immediate processing:

- **Authentication**: Login requests need immediate response for user experience
- **Attendance Submission**: Real-time verification ensures timely feedback to users
- **Location Verification**: Immediate geofence checking prevents attendance fraud

Asynchronous Processing

Non-critical operations can be processed in background:

- **Facial Recognition**: Complex image processing can occur asynchronously with status updates
- **Notification Distribution**: Bulk notifications are queued for efficient delivery
- **Analytics Generation**: Attendance statistics and reports are computed during low-usage periods

6. Database-Backend Integration Theory

6.1 Connection Management Strategies

Connection Pooling

Database connection pooling optimizes resource utilization:

- **Pool Size Configuration**: Balance between resource consumption and request handling capacity
- **Connection Lifecycle**: Automatic connection creation, validation, and cleanup
- **Timeout Management**: Idle connection cleanup prevents resource exhaustion
- **Load Balancing**: Multiple database servers can share connection load

Transaction Management

Database transactions ensure data consistency:

- **Transaction Boundaries**: Clear definition of atomic operation units
- **Rollback Strategies**: Error handling that maintains database consistency
- **Isolation Levels**: Appropriate concurrency control for multi-user environments
- **Deadlock Prevention**: Query ordering and timeout strategies prevent system locks

6.2 Data Synchronization Patterns

Online-First Approach

Primary operations occur with immediate database interaction:

- **Real-Time Validation**: Attendance submissions are immediately verified against database rules
- **Immediate Feedback**: Users receive instant confirmation of successful operations
- **Consistency Guarantee**: Database state remains consistent across all user interactions

Offline Capability

Mobile environments require offline functionality:

- **Local Storage**: Critical data cached on mobile devices for offline access
- **Synchronization Queue**: Offline operations queued for later database update
- **Conflict Resolution**: Strategies for handling conflicting updates when reconnecting
- **Data Freshness**: Mechanisms to ensure cached data remains reasonably current

6.3 Performance Optimization Theory

Query Optimization

Database query performance directly affects user experience:

- **Index Strategy**: Strategic indexing on frequently filtered and joined columns
- **Query Plan Analysis**: Regular examination of execution plans for optimization opportunities
- **Batch Operations**: Grouping multiple database operations for efficiency
- **Result Caching**: Frequently accessed data cached to reduce database load

Scalability Considerations

System design must accommodate growth:

- **Horizontal Scaling**: Database sharding strategies for large user populations
- **Vertical Scaling**: Hardware improvements to handle increased load
- **Read Replicas**: Separate read-only database instances for query distribution
- **Data Archiving**: Historical data management to maintain performance with growing datasets

7. Conclusion

The database design and implementation for the mobile-based attendance management system represents a comprehensive solution addressing the functional and non-functional requirements specified in the SRS document. The theoretical framework underlying this implementation draws from established database design principles, security best practices, and mobile application development patterns.

The hybrid database approach leverages the strengths of both relational and document-oriented storage systems, providing optimal performance for different data types and access patterns. The careful consideration of academic environment requirements ensures the system can effectively serve educational institutions while maintaining compliance with privacy regulations and integration capabilities with existing systems.

The modular design approach facilitates future enhancements and scalability improvements, ensuring the system can evolve with changing technological requirements and institutional needs.

The comprehensive theoretical foundation provides a solid basis for implementation decisions and ongoing system maintenance.

This detailed analysis demonstrates how database design theory translates into practical solutions for real-world applications, specifically addressing the unique requirements of educational attendance management in mobile environments. The integration of biometric authentication, location verification, and traditional database management principles creates a robust and secure system suitable for modern educational institutions.