**Functional Requirements (FR)**

**1. User Roles**

* **FR1**: The system shall support two roles:
* **Admin/Invigilator**: Create/manage exam sessions, validate enrollments, upload class lists, view reports.
* **Student**: Pre-enroll with documents/biometrics, authenticate before exams.

**2. Pre-Enrollment**

* **FR2**: Students shall upload scanned copies of:
* Form B (registered courses).
* ID card (school/NIC).
* Fee receipt (proof of payment).
* **FR3**: Students shall enroll their **biometric data** (modality-agnostic) via a physical kiosk.
* **FR4**: Admins shall manually approve/reject enrollments after verifying document validity.

**3. Class List & CA Marks Management**

* **FR5**: Admins shall upload **class lists** (CSV/Excel) for each course, including:
* Student IDs (matriculation numbers).
* **CA marks**.
* **FR6**: The system shall link CA marks to student enrollments (via student matriculation number).

**4. Exam Session Management**

* **FR7**: Admins shall create **authentication sessions** for:
* **Faculty/school**
* Department,
* Level,
* Course (e.g., "COME6101: Advanced Algorithms").
* Date/time window (e.g., 8:00 AM – 10:00 AM, 20 Nov 2024).
* Venue (e.g., "NAHPI Hall A").
* **FR8**: The system shall auto-generate a list of enrolled students for the course (from Form B and class list data).

**5. Authentication Workflow**

* **FR9**: During authentication, students shall:

1. Provide biometric data (e.g., fingerprint scan).
2. Receive real-time validation of:  
   * Biometric match.
   * Document validity (Form B, fee receipt).
   * CA mark eligibility (CA > 0 or "Eligible" status).

* **FR10**: The system shall display:
* Student name, ID (matriculation number), photo.
* Course eligibility status.
* CA mark (if available).
* **FR11**: Grant/deny access based on:
* Valid biometric match.
* Approved documents.
* CA mark eligibility.

**6. Reporting**

* **FR12**: Admins shall view/download:
* Real-time list of authenticated students (with CA marks).
* List of "Question Marked" students with reasons (e.g., "CA Missing", "Fee Unpaid").
* Final attendance report (PDF/Excel).

**7. Data Management**

* **FR13**: The system shall operate **offline** (no dependency on central university DB).
* **FR14**: Store data locally (students, enrollments, CA marks, sessions, logs).

**Non-Functional Requirements (NFR)**

**1. Performance**

* **NFR1**: Authentication response time ≤ 3 seconds per student.
* **NFR2**: Support **50+ concurrent authentications** (scalable to 500+ with optimizations).

**2. Security**

* **NFR3**: Biometric templates (not raw data) stored with **AES-256 encryption**.
* **NFR4**: Student documents and CA marks secured with **role-based access control** (admins only).

**3. Usability**

* **NFR5**: Admin interface requires ≤ 30 minutes of training (simple CSV uploads, intuitive dashboards).
* **NFR6**: Authentication kiosk has a 3-step interface (e.g., "Scan → Verify → Result").

**4. Scalability**

* **NFR7**: Modular design to scale from **1 department (NAHPI)** to the entire university (15k+ students).
* **NFR8**: Database indexing for fast queries (e.g., student ID, course code).

**5. Reliability**

* **NFR9**: System uptime ≥ 95% during exam sessions.
* **NFR10**: Daily backups of critical data (enrollments, CA marks, logs).

**6. Cost**

* **NFR11**: Total hardware/software cost ≤ $200 (MVP phase).
* Example: Refurbished laptop ($150) + fingerprint scanner ($50).

**7. Interoperability**

* **NFR12**: Support common file formats:
* Documents: PDF, JPG, PNG.
* Class lists: CSV, Excel.

**Out-of-Scope (For Future Iterations)**

1. Integration with the university’s central database.
2. Automated CA mark generation (admins will upload manually).
3. Multi-modal biometrics (e.g., fingerprint + facial recognition).
4. Mobile app support for students/admins.

**Key Use Case Example**

**Scenario**: A student attempts to authenticate for the "CE500: Advanced Algorithms" exam.

1. Scans fingerprint at the kiosk.
2. System checks:

* Biometric match → Confirms identity.
* Form B → Registered for CE500.
* Fee receipt → Paid.
* CA mark → 25/30 (eligible).

1. Result:

✅ Authenticated

Name: John Doe

ID: S001

Course: CE500

CA Mark: 25/30

**Why These Requirements Work**

* **Comprehensive Validation**: Combines biometrics, document checks, **and academic eligibility (CA marks)**.
* **MVP Focus**: Manual processes (document/CA uploads) avoid central DB dependency.
* **Scalability**: Modular design allows future expansion (e.g., facial recognition, auto-CA sync).

Start with these specs, and iterate as you build! 🚀

Here’s a **concise, unambiguous list of use cases** for your MVP, structured by user roles (Admin/Invigilator and Student). These cover onboarding, core workflows, and edge cases.

### ****Use Case Diagram Overview****

### ****1. Admin/Invigilator Use Cases****

#### **UC1: Manage Student Enrollment**

* **Description**: Approve/reject student enrollment after verifying documents.
* **Trigger**: Student submits enrollment documents.
* **Preconditions**:
* System is set up.
* Student has uploaded Form B, ID, fee receipt, and biometric data.
* **Flow**:

1. Admin logs into the system.
2. Navigates to "Pending Enrollments".
3. Reviews student documents.
4. Approves or rejects enrollment.

* **Postconditions**: Student is enrolled (or rejected) and notified via email/SMS.

#### **UC2: Create Exam Session**

* **Description**: Set up an authentication session for a specific course/exam.
* **Trigger**: Exam timetable is published.
* **Preconditions**:
* Admin is logged in.
* Class list with CA marks is uploaded.
* **Flow**:

1. Admin selects "Create New Session".
2. Inputs faculty/school, department, level, course, date, start and end times and venue.
3. System auto-generates the list of eligible students (with CA marks).
4. Session is saved.

* **Postconditions**: Session is active and ready for authentication.

#### **UC3: Upload Class List with CA Marks**

* **Description**: Upload a CSV/Excel file of students and CA marks for a course.
* **Trigger**: Course coordinator provides the class list.
* **Preconditions**:
* Admin is logged in.
* CSV/Excel file is formatted correctly.
* **Flow**:

1. Admin selects "Upload Class List".
2. Uploads the file.
3. System validates and links CA marks to enrolled students.

* **Postconditions**: CA marks are visible in the session dashboard.

#### **UC4: Monitor Authentication Session**

* **Description**: Real-time view of authenticated/flagged students during an exam.
* **Trigger**: Exam session starts.
* **Preconditions**:
* Exam session is active.
* **Flow**:

1. Admin opens the session dashboard.
2. Views real-time list of:  
   * Authenticated students (green).
   * Question Marked students (red) with reasons (e.g., "CA Missing").

* **Postconditions**: Admin can intervene for flagged cases.

#### **UC5: Generate Attendance Report**

* **Description**: Export post-exam attendance report.
* **Trigger**: Exam session ends.
* **Preconditions**:
* Session is closed.
* **Flow**:

1. Admin selects "Generate Report".
2. Chooses format (PDF/Excel).
3. Downloads and shares with the exam office.

* **Postconditions**: Report is archived for future audits.

### ****2. Student Use Cases****

#### **UC6: Submit Enrollment Documents**

* **Description**: Upload Form B, ID, and fee receipt to the system.
* **Trigger**: Student wants to enroll for exams.
* **Preconditions**:
* Student has valid documents.
* **Flow**:

1. Student logs into the portal.
2. Uploads scanned documents.
3. Submits for approval.

* **Postconditions**: Documents are pending admin review.

#### **UC7: Enroll Biometric Data**

* **Description**: Register fingerprint/face at a physical kiosk.
* **Trigger**: Enrollment documents are approved.
* **Preconditions**:
* Enrollment is approved (UC1).
* Biometric hardware is available.
* **Flow**:

1. Student visits enrollment kiosk.
2. Scans biometric (e.g., fingerprint) 3 times for accuracy.
3. System confirms enrollment.

* **Postconditions**: Biometric template is stored securely.

#### **UC8: Authenticate for Exam**

* **Description**: Verify identity and eligibility before entering the exam hall.
* **Trigger**: Student arrives for an exam.
* **Preconditions**:
* Exam session is active (UC2).
* Student is enrolled (UC7).
* **Flow**:

1. Student scans biometric at the exam kiosk.
2. System checks:  
   * Biometric match.
   * Valid documents.
   * CA mark eligibility.
3. Displays:  
   * ✅ **Authenticated** (grant access).
   * ❌ **Question Marked** (deny access + reason).

* **Postconditions**: Student is allowed/denied entry; log is saved.

### ****Edge Cases****

1. **UC9: Handle Biometric Mismatch**

* **Flow**:  
  1. Student fails biometric scan 3 times.
  2. System prompts for manual ID input.
  3. Admin verifies ID manually.

1. **UC10: Resolve Missing CA Mark**

* **Flow**:  
  1. Student is flagged for "CA Missing".
  2. Admin checks with the course coordinator.
  3. Updates CA mark manually (if error).

### ****Key Workflows****

#### **Student Flow**

1. Submit documents (UC6) → Enroll biometrics (UC7) → Authenticate (UC8).

#### **Admin Flow**

1. Approve enrollments (UC1) → Upload CA marks (UC3) → Create session (UC2) → Monitor (UC4) → Generate report (UC5).

### ****Next Steps****

1. Map these use cases to **system architecture** (e.g., Django for UC1-UC5, Python for UC7-UC8).
2. Define **database schema** (Students, Courses, Sessions, CA\_Marks).
3. Prioritize UC6-UC8 (student-facing) and UC2-UC4 (admin-critical).

This covers all MVP interactions while leaving room for scalability

Let’s refine your **object-oriented analysis (OOA)** and transition into a **high-level design** for your system. We’ll structure it around the university’s hierarchy, courses, students, and authentication workflows.

### ****Key Entities (Domain Model)****

#### **1. University**

* **Attributes**:
* name: String (e.g., "University of Bamenda").
* schools: List of School or Faculty.

#### **2. School/Faculty**

* **Attributes**:
* schoolType: Enum (e.g., SchoolOfEngineering, FacultyOfScience).
* departments: List of Department.

#### **3. Department**

* **Attributes**:
* name: String (e.g., "Computer Engineering").
* levels: List of Level.

#### **4. Level**

* **Attributes**:
* code: String (e.g., "500" for Year 3).
* courses: List of Course.

#### **5. Course**

* **Attributes**:
* code: String (e.g., "COME5101").
* name: String (e.g., "Embedded Systems Design").
* enrolledStudents: List of Student.
* caMarks: Dictionary<Student, CAMark>.

#### **6. Student**

* **Attributes**:
* studentId: String.
* name: String.
* formB: List of Course (registered courses).
* feeReceipts: List of FeeReceipt.
* biometricData: BiometricData.

#### **7. CAMark**

* **Attributes**:
* student: Student.
* course: Course.
* score: Double (e.g., 25/30).

#### **8. ExamSession**

* **Attributes**:
* course: Course.
* startTime: DateTime.
* endTime: DateTime.
* venue: String.
* authenticatedStudents: List of Student.
* questionMarkedStudents: List<Student, Reason>.

#### **9. BiometricData**

* **Attributes**:
* student: Student.
* template: Encrypted binary (fingerprint/facial template).

#### **10. AuthenticationLog**

* **Attributes**:
* student: Student.
* timestamp: DateTime.
* status: Enum (Authenticated, QuestionMarked).
* reason: String (e.g., "CA Missing").

### ****Relationships (Associations)****

1. **University** has-many **Schools/Faculties**.
2. **School** has-many **Departments**.
3. **Department** has-many **Levels**.
4. **Level** has-many **Courses**.
5. **Course** has-many **Students** (via enrolledStudents).
6. **Student** has-a **FormB**, **FeeReceipts**, **BiometricData**.
7. **Student** has-many **CAMarks** (one per course).
8. **ExamSession** belongs-to **Course**.
9. **BiometricData** linked-to **Student**.

### ****Class Diagram (Simplified)****

+----------------+ +----------------+ +----------------+

| University |<>-----| School |<>-----| Department |

+----------------+ +----------------+ +----------------+

| - name | | - schoolType | | - name |

+----------------+ +----------------+ +----------------+

|

| +----------------+ +----------------+

+-----------| Level |<>-----| Course |

+----------------+ +----------------+

| - code | | - code |

+----------------+ | - name |

+----------------+

|

|

+----------------+ +----------------+ +----------------+

| Student |------| CAMark |<-----| ExamSession |

+----------------+ +----------------+ +----------------+

| - studentId | | - score | | - startTime |

| - name | +----------------+ | - endTime |

| - formB | | - venue |

| - biometricData| +----------------+

+----------------+

### ****Key Design Decisions****

1. **Aggregation Over Inheritance**:

* Use **composition** for university hierarchy (e.g., University → School → Department → Level → Course). Avoid deep inheritance hierarchies.

1. **Biometric Data Handling**:

* Store **encrypted templates** (not raw images) in BiometricData.
* Use a dedicated class to abstract biometric operations (e.g., FingerprintScanner).

1. **Course Enrollment**:

* Link Student to Course via formB (List of Course).
* Validate enrollment during authentication using formB and CAMark.

1. **Exam Sessions**:

* ExamSession is tied to a Course and tracks real-time authentication status.

1. **Security**:

* Encrypt sensitive data (e.g., BiometricData.template, FeeReceipts).

### ****Core Workflows (Sequence Diagrams)****

#### **1. Student Authentication**

Student -> Kiosk: Scan biometric

Kiosk -> BiometricSystem: Match biometric

BiometricSystem -> Database: Fetch Student

Database --> BiometricSystem: Student data

BiometricSystem -> Validator: Check FormB, FeeReceipt, CAMark

Validator --> Kiosk: Authentication Result

Kiosk --> Student: Display Status (✅/❌)

#### **2. Admin Creates Exam Session**

Admin -> System: Create ExamSession (Course, Time, Venue)

System -> Database: Fetch enrolledStudents (from FormB)

System -> Database: Fetch CAMarks (for Course)

System --> Admin: Session Created

### ****Design Patterns****

1. **Factory Pattern**:

* Use BiometricFactory to create modality-specific handlers (e.g., FingerprintHandler, FacialHandler).

1. **Observer Pattern**:

* ExamSession notifies admins in real-time when students are flagged.

1. **Singleton Pattern**:

* Use for DatabaseConnection to ensure a single pool of DB connections.

### ****Database Schema (SQL Sketch)****

CREATE TABLE Student (

student\_id VARCHAR(20) PRIMARY KEY,

name VARCHAR(100),

biometric\_template BYTEA -- Encrypted

);

CREATE TABLE Course (

course\_code VARCHAR(10) PRIMARY KEY,

name VARCHAR(100),

level\_code VARCHAR(10),

department\_id INT REFERENCES Department(id)

);

CREATE TABLE CAMark (

student\_id VARCHAR(20) REFERENCES Student(student\_id),

course\_code VARCHAR(10) REFERENCES Course(course\_code),

score DOUBLE,

PRIMARY KEY (student\_id, course\_code)

);

CREATE TABLE ExamSession (

session\_id SERIAL PRIMARY KEY,

course\_code VARCHAR(10) REFERENCES Course(course\_code),

start\_time TIMESTAMP,

end\_time TIMESTAMP,

venue VARCHAR(100)

);

### ****Next Steps (Design Phase)****

1. **Prototype Biometric Integration**:

* Use Python’s pyfingerprint library for fingerprint scanning.

1. **Build Admin Dashboard**:

* Use Django for CRUD operations on courses/sessions.

1. **Test Core Workflows**:

* Simulate authentication for 50 students in a pilot course.

This OOAD gives you a blueprint to start coding. Let me know if you need help with specific modules! 🚀