Campus ERP – Smart College Management System

Software Requirement Specification (SRS)

Document Control

Item	Description
Document Title	Software Requirement Specification (SRS)
System Name	CampusERP – Smart College Management System
Module	Academic Module
Version	1.0
Prepared For	Institutional Use
Prepared By	CampusERP Development Team

---

1. Introduction

1.1 Purpose

This document presents the **updated Software Requirement Specification (SRS)** for the **CampusERP – Smart College Management System**, with a **clear and structured separation of the Academic Module as an independent sub-module**.

The primary objectives of this document are to:

- Establish clear system modularity and responsibility separation
- Improve maintainability, scalability, and extensibility
- Support future academic enhancements and regulatory frameworks
- Serve as a standardized reference for development, testing, and deployment

---

1.2 Scope

CampusERP is a **centralized, web-based ERP platform** designed to automate and integrate all academic and administrative processes within an educational institution.

1.3 Intended Users

The CampusERP system is designed to support the following user groups:

- College Management / Executives
- Academic Administrators
- Faculty Members
- Examination Controller
- Students
- Parents / Guardians
- ERP System Administrator

Access to system functionality is strictly governed using **Role-Based Access Control (RBAC)**.

1.4 Definitions, Acronyms, and Abbreviations

Term	Description
ERP	Enterprise Resource Planning
RBAC	Role-Based Access Control
API	Application Programming Interface
UI	User Interface
SRS	Software Requirement Specification

---

2. Overall System Description

2.1 System Perspective

CampusERP is a **modular, secure, and scalable ERP system** built on a centralized database and service-oriented API architecture.

Key System Characteristics:

- Independent and loosely coupled functional modules
- Centralized and consistent data repository
- Secure, role-based user authentication
- Seamless communication between modules

The **Academic Module** functions as a **core reference module**, supplying academic structure and configuration data required by multiple dependent subsystems.

2.2 High-Level System Modules

No.	Module Name	Description
1	**Dashboard**	Displays system overview, analytics, and KPIs
2	**Alerts**	Generates alerts for important events
3	**Reports**	Produces academic and operational reports
4	**Notifications**	Sends SMS, email, and in-app notifications
5	**Admissions**	Manages student admission and enrolment
6	**Students**	Maintains student profiles and lifecycle
7	**Academic**	Manages academic structure and curriculum
8	**Staff**	Manages faculty and staff information
9	**Attendance**	Tracks student attendance
10	**Timetable**	Manages class and faculty schedules
11	**Exams**	Handles examination processes
12	**Library**	Manages library resources
13	**Hostel**	Manages hostel facilities
14	**Transport**	Manages transport operations
15	**Finance**	Manages fees and financial transactions

Phase 1 – Core Setup

1. Academic
2. Students
3. Staff

Phase 2 – Academic Operations

4. Timetable
5. Attendance

6. **Exams**

Phase 3 – Administration & Support

7. **Admissions**
8. **Finance**
9. **Library**
10. **Hostel**
11. **Transport**

Phase 4 – System Intelligence

12. **Dashboard**
13. **Reports**
14. **Alerts**
15. **Notifications**

Phase 1 – Core Setup

1. Academic
2. Students
3. Staff

Phase 2 – Academic Operations

4. Timetable
5. Attendance
6. Exams

Academic

- Departments
- Courses
- Classes/ Sections
- Semesters/Subjects
- Regulation

[package com.erp.common.audit;]

BaseAuditEntity

```
package com.erp.common.audit;

import jakarta.persistence.*;

import lombok.*;

import java.time.LocalDateTime;

/**

 * BaseAuditEntity

 * ----------------
```

```
 * Provides automatic audit fields for all entities.
 * Every entity should extend this class.
 */
@MappedSuperclass
@Getter
@Setter
public abstract class BaseAuditEntity {

    /**
     * Timestamp when entity was created
     */
    @Column(nullable = false, updatable = false)
    private LocalDateTime createdAt;

    /**
     * Timestamp when entity was last updated
     */
    @Column(nullable = false)
    private LocalDateTime updatedAt;

    /**
     * User who created the record
     */
    @Column(updatable = false)
    private String createdBy;

    /**
     * User who last updated the record
     */
    private String updatedBy;
```

```
    /**
     * Automatically set values before insert
     */
    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
        this.updatedAt = LocalDateTime.now();

        // TEMP: replace later with logged-in user
        this.createdBy = "SYSTEM";
        this.updatedBy = "SYSTEM";
    }

    /**
     * Automatically update values before update
     */
    @PreUpdate
    protected void onUpdate() {
        this.updatedAt = LocalDateTime.now();

        // TEMP: replace later with logged-in user
        this.updatedBy = "SYSTEM";
    }
}
```

🔹 **STEP 0: Regulation Entity**

Purpose

- Defines **batch-wise academic rules & syllabus**
- Handles **NEP changes**, credit changes, grading rules
- Ensures **old batches remain unaffected**

- Independent from AcademicYear & Course

📌 **Real-life meaning:**
University publishes **R2019**, **R2022**, **R2024** regulations.

Regulation – Form Fields

🔸 **Add Regulation Form**

Field	Type	Notes
Regulation Code*	Text	R2019, R2022
Regulation Name*	Text	Regulation 2022 – NEP
Effective From Year*	Number	Admission year
Description	Textarea	Optional
Status*	Active / Inactive	

```
@Entity
@Table(
  name = "regulation",
  uniqueConstraints = {
    @UniqueConstraint(name = "uk_regulation_code", columnNames = "regulationCode")
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Regulation {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long regulationId;

  @NotBlank(message = "Regulation code is required")
```

```
    @Column(nullable = false, length = 20)
    private String regulationCode; // R2019, R2022

    @NotBlank(message = "Regulation name is required")
    private String regulationName;

    @NotNull(message = "Effective year is required")
    private Integer effectiveFromYear;

    private String description;

    @NotNull(message = "Status is required")
    @Enumerated(EnumType.STRING)
    private Status status;
}
```

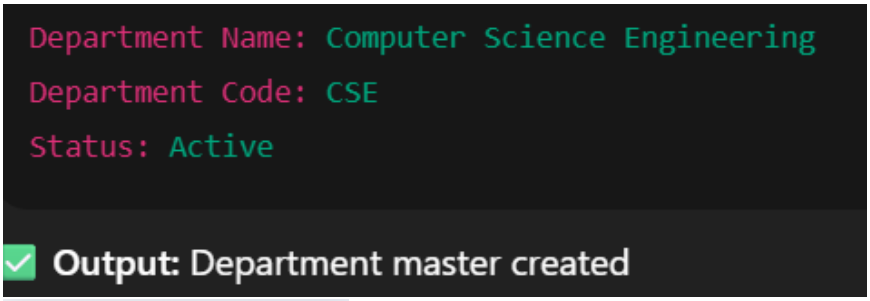
Entity	Regulation FK	Reason
Class	✅ YES	Batch follows one regulation
Semester	✅ YES	Semester structure differs
Subject	✅ YES	Credits & grading differ
AcademicYear	❌ NO	Year ≠ syllabus
Course	❌ NO	One course → many regulations
Department	❌ NO	Independent
Section	❌ NO	Inherits from Class

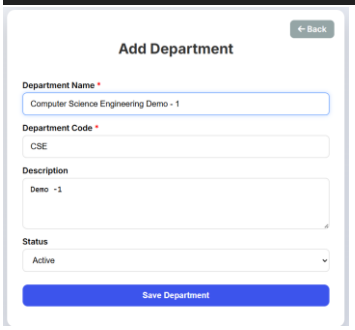
Department

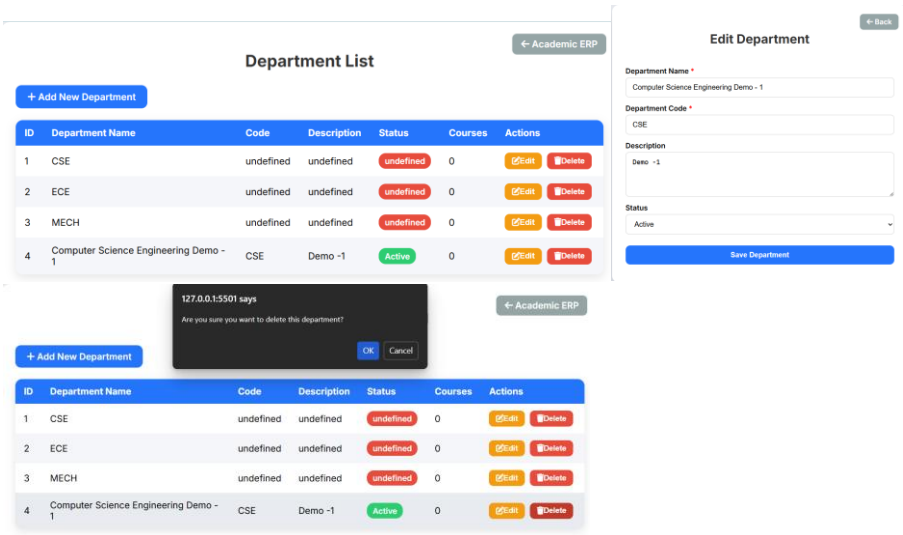
Form Fields

- **Department Name***
- **Department Code***

- Description
- Status (Active / Inactive)*
-







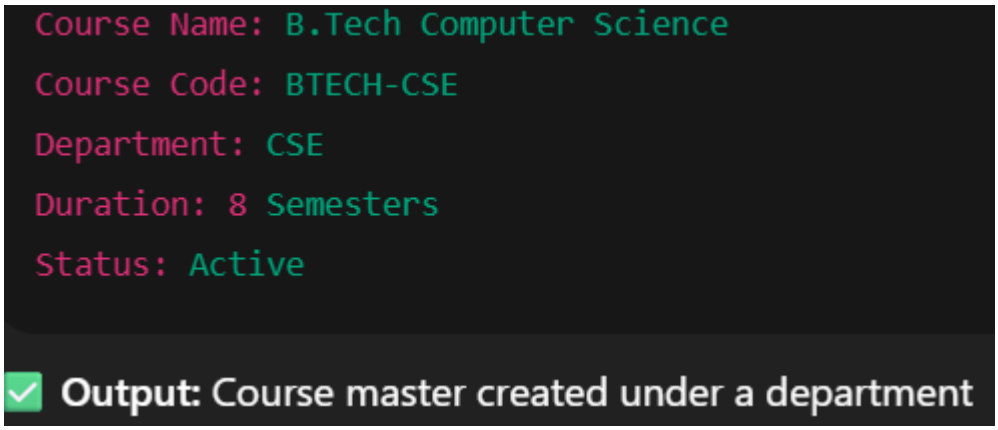
Course

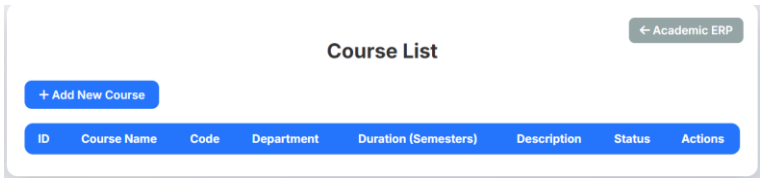
Description

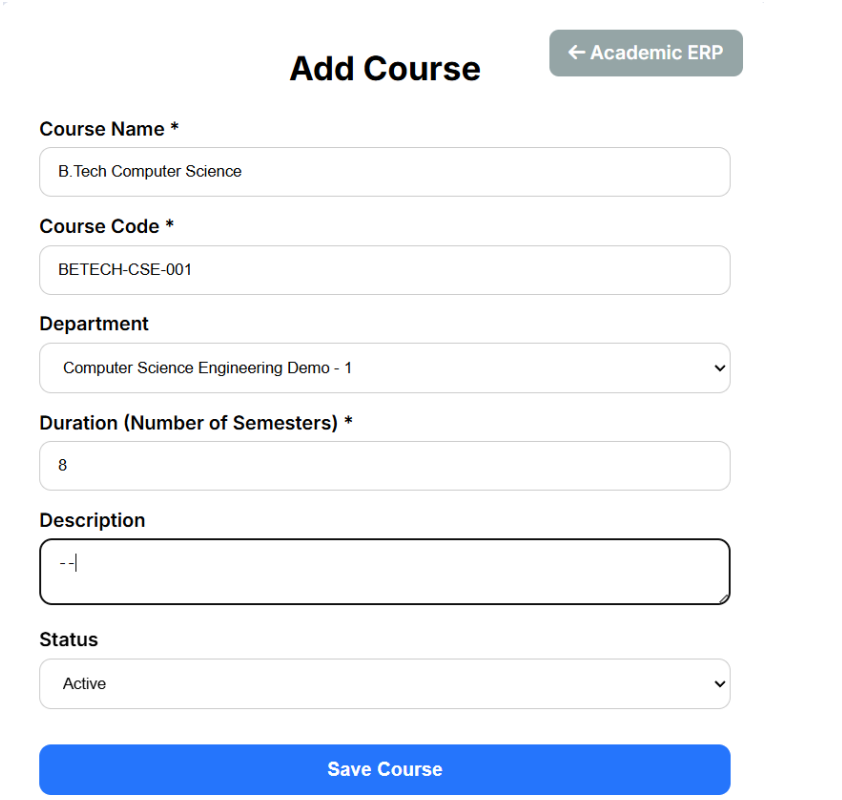
This step is used to define academic courses offered by each department. A course represents a degree program such as B.Tech, BCA, or MCA.

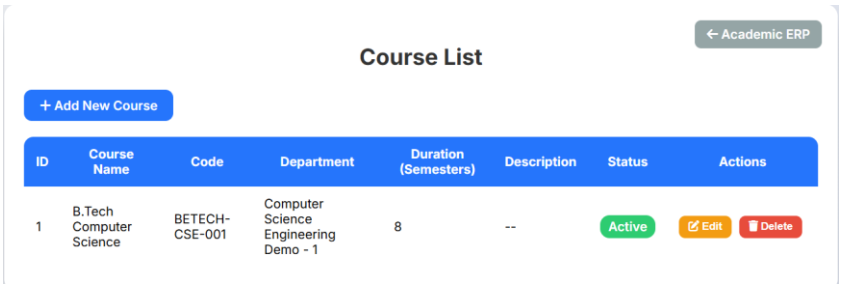
Form Fields

- **Course Name***
- **Course Code***
- **Department (Select Department)**
  (List of departments fetched from Department Master)
- **Degree Type** (B.Tech / BCA / MCA)
- **Duration (Number of Semesters)***
- Description
- Status (Active / Inactive)
-









Class & Section

Description

This step is used to create classes for a specific course and divide them into multiple sections. Classes represent a **course-year grouping**, while sections represent **sub-divisions of a class** to manage students efficiently.

Classes and sections are essential for attendance tracking, timetable creation, and examination management.

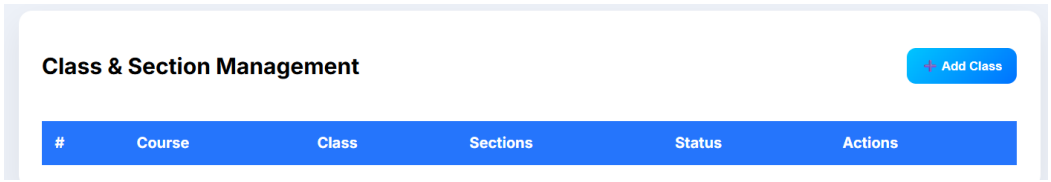
Form Fields – Class Details

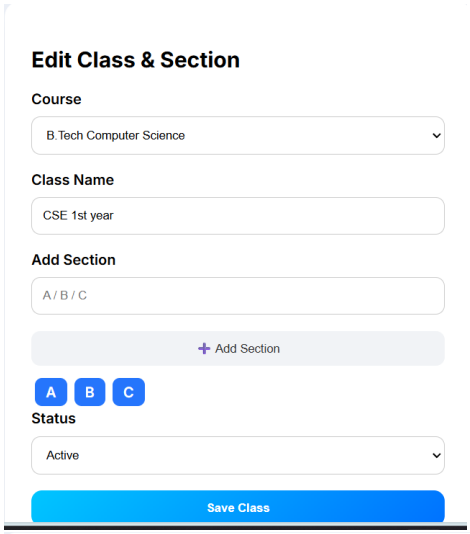
- **Course (Select Course)**
- **Class Name***
  (Example: CSE 1st Year)
- Status (Active / Inactive)

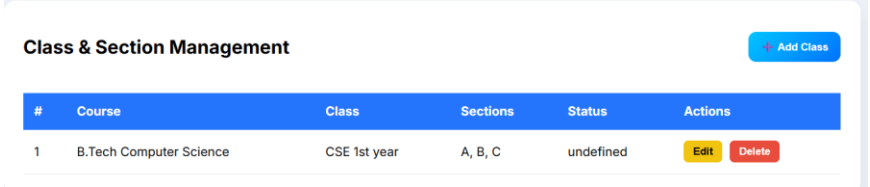
Form Fields – Section Details

- **Section Name** (A / B / C)
- **+ Add Section** (Add multiple sections dynamically)
- Status (Active / Inactive)

```
Course: B.Tech CSE
Class Name: CSE 1st Year
Sections: A, B, C
Status: Active
```







STEP 4: Semesters & Subjects (Academic Module)

Add Subject –

Form Fields

Department

- Auto-selected

- Read-only (derived from course)

Course

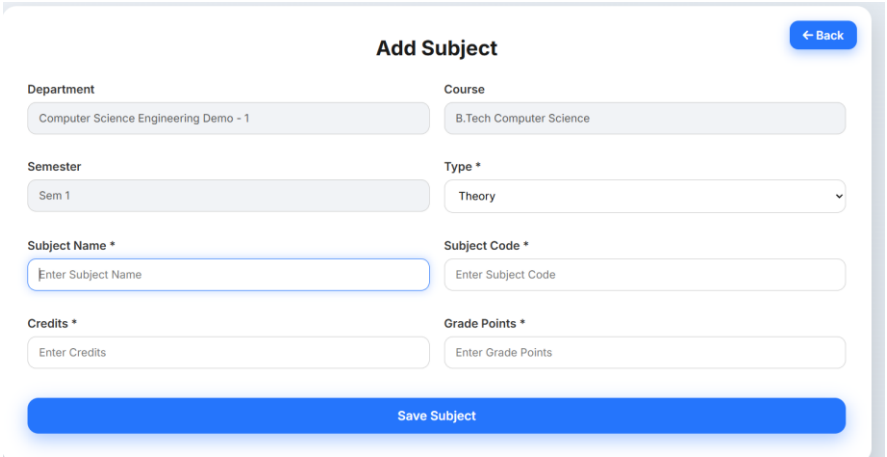
- Auto-selected
- Read-only

Semester

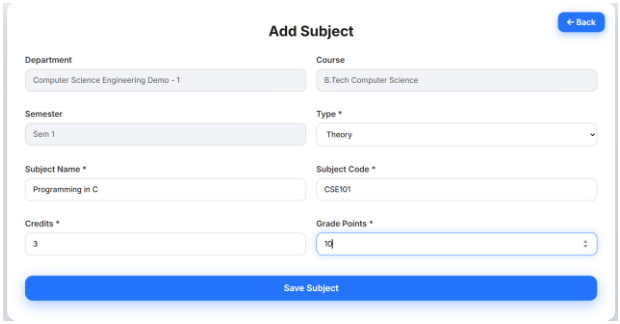
- Selected semester (Sem 1 / Sem 2 / etc.)

Subject Details Fields

- **Subject Name***
- **Subject Code***
- **Type***
    - Theory
    - Practical
    - Lab
- **Credits***
- **Grade Points***
- .
- .



```
Department: Computer Science Engineering
Course: B.Tech Computer Science
Semester: Sem 1
Subject Name: Programming in C
Subject Code: CSE101
Type: Theory
Credits: 4
Grade Points: 10
```



🔹 STEP 0: Main Core Academic Entities

main entities are:

1. Department
2. Course
3. Class
4. Section
5. Semester
6. Subject

📘 Academic Module – Entity Classes with Relationships & Comments

---

🔹 STEP 0: AcademicYear Entity (MASTER)

Purpose

Represents an academic session (batch) such as **2024–2025**.
Used to separate data year-wise and preserve history.

Entity: AcademicYear

Field	Type	Purpose
academic_year_id (PK)	Long	Unique identifier for academic year
academic_year_name	String	Display name (e.g., 2024–2025)
start_date	Date	Academic year start
end_date	Date	Academic year end
status	Enum	Active / Closed

Relationships

- One Academic Year → Many Classes
- One Academic Year → Many Semesters

🔹 **STEP 1: Department Entity (MASTER)**

Purpose

Top-level academic unit (CSE, ECE, MECH).
Shared across courses, staff, and subjects.

Entity: Department

Field	Type	Purpose
department_id (PK)	Long	Unique department identifier
department_name	String	Full name of department
department_code	String	Short code (CSE, ECE)
description	String	Optional department details
status	Enum	Active / Inactive

Relationships

- One Department → Many Courses
- One Department → Many Subjects

🔹 **STEP 2: Course Entity (Depends on Department)**

Purpose

Represents degree programs (B.Tech CSE, BCA).

Entity: Course

Field	Type	Purpose
course_id (PK)	Long	Unique course identifier
course_name	String	Course name
course_code	String	Short course code
degree_type	Enum	B.Tech / BCA / MCA
duration_semesters	Integer	Total number of semesters
department_id (FK)	Long	Links course to department
description	String	Optional description
status	Enum	Active / Inactive

Relationships

- Many Courses → One Department
- One Course → Many Classes
- One Course → Many Semesters
- One Course → Many Subjects

🔹 **STEP 3: Class Entity (Depends on Course + AcademicYear)**

Purpose

Represents **course + academic year grouping**
(e.g., *CSE 1st Year – 2024–25*).

Entity: Class

Field	Type	Purpose
class_id (PK)	Long	Unique class identifier
class_name	String	Class display name
course_id (FK)	Long	Which course the class belongs to
academic_year_id (FK)	Long	Batch/year identification
status	Enum	Active / Inactive

Relationships

- Many Classes → One Course

- Many Classes → One Academic Year
- One Class → Many Sections
- Added `regulation_id (FK)`
-

🔹 STEP 4: Section Entity (Depends on Class)

Purpose

Sub-division of a class (A, B, C)
Used for **attendance, timetable, exams**.

Entity: Section

Field	Type	Purpose
section_id (PK)	Long	Unique section identifier
section_name	String	Section label (A, B, C)
class_id (FK)	Long	Parent class
status	Enum	Active / Inactive

Relationships

- Many Sections → One Class

🔹 STEP 5: Semester Entity (Depends on Course + AcademicYear)

Purpose

Represents academic terms such as Sem 1, Sem 2.
Semesters are **course-based**, not class-based.

Entity: Semester

Field	Type	Purpose
semester_id (PK)	Long	Unique semester identifier
semester_number	Integer	Semester number (1–8)
course_id (FK)	Long	Course to which semester belongs
academic_year_id (FK)	Long	Academic year mapping
status	Enum	Active / Inactive

Relationships

- Many Semesters → One Course
- Many Semesters → One Academic Year
- One Semester → Many Subjects

🔹 STEP 6: Subject Entity (Depends on Semester + Course + Department)

Purpose

Represents subjects taught in a semester.

Entity: Subject

Field	Type	Purpose
subject_id (PK)	Long	Unique subject identifier
subject_name	String	Subject title
subject_code	String	Subject code
subject_type	Enum	Theory / Lab / Practical
credits	Integer	Credit value
grade_points	Integer	Grade calculation
semester_id (FK)	Long	Semester mapping
course_id (FK)	Long	Course mapping
department_id (FK)	Long	Department mapping
status	Enum	Active / Inactive

Relationships

- Many Subjects → One Semester
- Many Subjects → One Course
- Many Subjects → One Department

🔗 Overall Relationship Summary (One View)

AcademicYear

├── Class

└── Semester

Department —▶ Course —▶ Class —▶ Section

│

└— Semester —▶ Subject

Part 1.

Common Enums

✅ **Status.java**

```
package com.erp.academic.enums;
/**
 * Represents current lifecycle state of an entity
 */
public enum Status {
   ACTIVE,    // Entity is active and usable
   INACTIVE,  // Entity is temporarily disabled
   CLOSED     // Entity is permanently closed (mainly for Academic Year)
}
```

✅ **DegreeType.java**

```
package com.erp.academic.enums;
public enum DegreeType {
/**
 * Degree types offered by the institution
 */
    // Undergraduate Degrees
   BA,         // Bachelor of Arts
   BSC,        // Bachelor of Science
   BCOM,       // Bachelor of Commerce
   BBA,        // Bachelor of Business Administration
   BCA,        // Bachelor of Computer Applications
   BTECH,      // Bachelor of Technology
```

```
    BE,          // Bachelor of Engineering
    BPHARM,      // Bachelor of Pharmacy
    BARCH,       // Bachelor of Architecture
    BED,         // Bachelor of Education
    BSW,         // Bachelor of Social Work

    // Postgraduate Degrees
    MA,          // Master of Arts
    MSC,         // Master of Science
    MCOM,        // Master of Commerce
    MBA,         // Master of Business Administration
    MCA,         // Master of Computer Applications
    MTECH,       // Master of Technology
    ME,          // Master of Engineering
    MPHARM,      // Master of Pharmacy
    MED,         // Master of Education
    MSW,         // Master of Social Work

    // Doctoral Degrees
    PHD,         // Doctor of Philosophy

    // Diploma / Certificate
    DIPLOMA,
    PG_DIPLOMA,
    CERTIFICATION
}
```

✅ **SubjectType.java**

```
package com.erp.academic.enums;
public enum SubjectType {
/**
```

```
 * Subject classification
 */
// Core academic types
    THEORY,              // Classroom-based theory
    PRACTICAL,           // Practical sessions
    LAB,                 // Laboratory work

    // Skill / application based
    PROJECT,             // Final / mini project
    SEMINAR,             // Presentation / seminar
    WORKSHOP,            // Hands-on workshops
    INTERNSHIP,          // Industry internship
    FIELD_WORK,          // Field visits / surveys
    CASE_STUDY,          // Case-based learning

    // Evaluation / assessment
    TUTORIAL,            // Tutorial sessions
    ASSIGNMENT,          // Assignment-based subject
    VIVA,                // Oral examination
    CONTINUOUS_EVALUATION,  // Internal assessment

    // Research / advanced
    RESEARCH,            // Research-oriented subject
    DISSERTATION,        // Dissertation / thesis

    // Non-credit / support
    ELECTIVE,            // Elective subject
    AUDIT,               // Audit course (no credits)
    NON_CREDIT,          // No credit course
    VALUE_ADDED,         // Value added course
    SKILL_DEVELOPMENT      // Skill-based learning
```

}

Status.java

```java
package com.erp.academic.enums;

public enum Status {
    ACTIVE,
    INACTIVE,
    CLOSED
}
```

DegreeType.java

```java
package com.erp.academic.enums;

public enum DegreeType {
    BTECH,
    BCA,
    MCA
}
```

SubjectType.java

```java
package com.erp.academic.enums;

public enum SubjectType {
    THEORY,
    PRACTICAL,
    LAB
}
```

<dependency>

  <groupId>org.springframework.boot</groupId>

  <artifactId>spring-boot-starter-validation</artifactId>

</dependency>

🔹 **STEP 0: AcademicYear Entity**

```
package com.erp.academic.entity;

import com.erp.academic.enums.Status;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

import java.time.LocalDate;

/**
 * AcademicYear represents a single academic session (batch),
 * such as 2024-2025. It is used to separate year-wise data
 * across classes, semesters, students, attendance, and exams.
 */
@Entity
```

```
@Table(
  name = "academic_year",
  indexes = {
    @Index(name = "idx_academic_year_name", columnList = "academicYearName")
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class AcademicYear {

  /** Primary key for Academic Year */
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long academicYearId;

  /** Display name of academic year (e.g., 2024-2025) */
  @NotBlank(message = "Academic year name is required")
  @Column(nullable = false, unique = true, length = 20)
  private String academicYearName;

  /** Academic session start date */
  @NotNull(message = "Start date is required")
  private LocalDate startDate;

  /** Academic session end date */
  @NotNull(message = "End date is required")
  private LocalDate endDate;

  /** Current status of academic year */
```

```
    @NotNull(message = "Status is required")
    @Enumerated(EnumType.STRING)
    private Status status;
}
```

🔹 **STEP 1: Department Entity (MASTER)**

```
package com.erp.academic.entity;

import com.erp.academic.enums.Status;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

/**
 * Department represents the top-level academic unit
 * such as CSE, ECE, MECH.
 */
@Entity
@Table(
    name = "department",
    uniqueConstraints = {
        @UniqueConstraint(name = "uk_department_code", columnNames =
"departmentCode")
    },
    indexes = {
        @Index(name = "idx_department_name", columnList = "departmentName")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
```

```
@Builder
public class Department {

    /** Unique identifier for department */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long departmentId;

    /** Full name of the department */
    @NotBlank(message = "Department name is required")
    @Size(min = 3, max = 100)
    @Column(nullable = false)
    private String departmentName;

    /** Short department code (CSE, ECE, etc.) */
    @NotBlank(message = "Department code is required")
    @Size(min = 2, max = 10)
    @Column(nullable = false)
    private String departmentCode;

    /** Optional description of department */
    @Size(max = 255)
    private String description;

    /** Active/Inactive status */
    @NotNull(message = "Status is required")
    @Enumerated(EnumType.STRING)
    private Status status;
}
```

🔹 **STEP 2: Course Entity (Depends on Department)**

```
package com.erp.academic.entity;
```

```
import com.erp.academic.enums.*;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

/**
 * Course represents a degree program offered by a department,
 * such as B.Tech CSE, BCA, MCA.
 */
@Entity
@Table(
  name = "course",
  uniqueConstraints = {
    @UniqueConstraint(name = "uk_course_code", columnNames = "courseCode")
  },
  indexes = {
    @Index(name = "idx_course_name", columnList = "courseName")
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Course {

  /** Primary key for course */
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long courseId;
```

```
/** Course name */
@NotBlank(message = "Course name is required")
private String courseName;

/** Unique course code */
@NotBlank(message = "Course code is required")
private String courseCode;

/** Degree type (BTECH, BCA, MCA) */
@NotNull(message = "Degree type is required")
@Enumerated(EnumType.STRING)
private DegreeType degreeType;

/** Total semesters in the course */
@NotNull(message = "Duration is required")
@Min(1)
@Max(12)
private Integer durationSemesters;

/** Department to which this course belongs */
@NotNull(message = "Department is mandatory")
@ManyToOne
@JoinColumn(name = "department_id", nullable = false)
private Department department;

/** Optional description */
private String description;

/** Active/Inactive status */
@NotNull(message = "Status is required")
@Enumerated(EnumType.STRING)
```

```
    private Status status;
}
```

🔹 **STEP 3: Class Entity (Depends on Course + AcademicYear)**

```
package com.erp.academic.entity;

import com.erp.academic.enums.Status;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

/**
 * Class represents a course batch for a specific academic year,
 * e.g., CSE 1st Year (2024-2025).
 */
@Entity
@Table(
    name = "class",
    indexes = {
        @Index(name = "idx_class_course_year", columnList = "course_id, academic_year_id")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ClassEntity {

    /** Unique class identifier */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long classId;

    /** Display name of class */
    @NotBlank(message = "Class name is required")
    private String className;

    /** Course under which this class exists */
    @NotNull(message = "Course is required")
    @ManyToOne
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;

    /** Academic year (batch) of the class */
    @NotNull(message = "Academic year is required")
    @ManyToOne
    @JoinColumn(name = "academic_year_id", nullable = false)
    private AcademicYear academicYear;

// 🔥 REGULATION FK
    @ManyToOne
    @JoinColumn(name = "regulation_id", nullable = false)
    private Regulation regulation;

    /** Active/Inactive status */
    @NotNull(message = "Status is required")
    @Enumerated(EnumType.STRING)
    private Status status;
}
```

🔹 **STEP 4: Section Entity (Depends on Class)**

```
package com.erp.academic.entity;
```

```
import com.erp.academic.enums.Status;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

/**
 * Section represents a subdivision of a class (A, B, C).
 * Used for attendance, timetable, and exam allocation.
 */
@Entity
@Table(
  name = "section",
  uniqueConstraints = {
    @UniqueConstraint(
      name = "uk_section_class",
      columnNames = {"sectionName", "class_id"}
    )
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Section {

  /** Unique identifier for section */
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long sectionId;
```

```
    /** Section name (A, B, C) */
    @NotBlank(message = "Section name is required")
    @Pattern(regexp = "[A-Z]", message = "Section must be a single capital letter")
    private String sectionName;

    /** Parent class */
    @NotNull(message = "Class reference is required")
    @ManyToOne
    @JoinColumn(name = "class_id", nullable = false)
    private ClassEntity classEntity;

    /** Active/Inactive status */
    @NotNull(message = "Status is required")
    @Enumerated(EnumType.STRING)
    private Status status;
}
```

🔹 **STEP 5: Semester Entity (Depends on Course + AcademicYear)**

```
package com.erp.academic.entity;

import com.erp.academic.enums.Status;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

/**
 * Semester represents an academic term (Sem 1, Sem 2).
 * It is course-based and linked to an academic year.
 */
@Entity
@Table(
```

```
    name = "semester",
    uniqueConstraints = {
        @UniqueConstraint(
            name = "uk_course_semester_year",
            columnNames = {"semesterNumber", "course_id", "academic_year_id"}
        )
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Semester {

    /** Primary key */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long semesterId;

    /** Semester number (1–8) */
    @NotNull(message = "Semester number is required")
    @Min(1)
    @Max(12)
    private Integer semesterNumber;

    /** Course to which semester belongs */
    @NotNull(message = "Course is required")
    @ManyToOne
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;
```

```
    /** Academic year of semester */
    @NotNull(message = "Academic year is required")
    @ManyToOne
    @JoinColumn(name = "academic_year_id", nullable = false)
    private AcademicYear academicYear;

    // 🔥 REGULATION FK
    @ManyToOne
    @JoinColumn(name = "regulation_id", nullable = false)
    private Regulation regulation;

    /** Active/Inactive status */
    @NotNull(message = "Status is required")
    @Enumerated(EnumType.STRING)
    private Status status;
}
```

🔹 **STEP 6: Subject Entity (Depends on Semester + Course + Department)**

```
package com.erp.academic.entity;

import com.erp.academic.enums.*;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

/**
 * Subject represents an academic subject taught in a semester.
 * Used in attendance, exams, results, and faculty allocation.
 */
@Entity
```

```
@Table(
  name = "subject",
  uniqueConstraints = {
    @UniqueConstraint(name = "uk_subject_code", columnNames = "subjectCode")
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Subject {

  /** Primary key */
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long subjectId;

  /** Subject name */
  @NotBlank(message = "Subject name is required")
  private String subjectName;

  /** Unique subject code */
  @NotBlank(message = "Subject code is required")
  private String subjectCode;

  /** Subject type (Theory/Lab/Practical) */
  @NotNull(message = "Subject type is required")
  @Enumerated(EnumType.STRING)
  private SubjectType subjectType;

  /** Credit value of subject */
```

```
@NotNull(message = "Credits are required")
@Min(1)
@Max(10)
private Integer credits;

/** Grade points for evaluation */
@NotNull(message = "Grade points are required")
@Min(1)
@Max(10)
private Integer gradePoints;

/** Semester in which subject is taught */
@NotNull(message = "Semester is required")
@ManyToOne
@JoinColumn(name = "semester_id", nullable = false)
private Semester semester;

/** Course mapping */
@NotNull(message = "Course is required")
@ManyToOne
@JoinColumn(name = "course_id", nullable = false)
private Course course;

/** Department mapping */
@NotNull(message = "Department is required")
@ManyToOne
@JoinColumn(name = "department_id", nullable = false)
private Department department;

// 🔥 REGULATION FK
@ManyToOne
```

@JoinColumn(name = "regulation_id", nullable = false)

private Regulation regulation;

/ Active/Inactive status */**

@NotNull(message = "Status is required")

@Enumerated(EnumType.STRING)

private Status status;

}

Part 2

----------.

📘 PERSON MANAGEMENT – CORE MASTER DESIGN

🏛️ COLLEGE ERP SYSTEM
(Person → Student / Staff / Parent / User)

🔹 STEP 1: PERSON (CORE MASTER ENTITY)

The **Person** entity is the **central identity master** of the ERP.

It represents **any human associated with the institution**, regardless of role.

👥 Applicable For

- Student
- Teaching Staff
- Non-Teaching Staff
- Admin / Office Staff
- Parent / Guardian

⚠️ IMPORTANT DESIGN RULE

The Person table stores **ONLY biographical identity data**.

❌ It does NOT store:

- Login credentials
- Roles / permissions

- Academic data
- Employment data

This ensures **pure normalization and reuse**.

🔁 HIGH-LEVEL FLOW

🔁 COLLEGE ERP – MODEL FLOW (PERSON-CENTRIC) PERSON-CENTRIC MASTER DESIGN

The system is designed around one core identity and layered responsibilities.

🧱 SYSTEM LAYERS (HIGH LEVEL)

1. Identity Layer

2. Role Layer

3. Relationship Layer

4. Security Layer

1️⃣ IDENTITY LAYER (BASE OF EVERYTHING)

🧩 Core Entity: Person

Person

(person_id)

Purpose

- **Represents a human being**
- **Created once**
- **Reused across all modules**

Stores

- **Name**
- **DOB**
- **Gender**
- **Photo**
- **Status**

Does NOT Store

❌ **Login**
❌ **Role**

❌ Academic data
❌ Employment data

🔗 Dependent Identity Entities

Person

├— Contact (0..N)

├— Address (0..N)

└— PersonDocument (0..N)

- Cannot exist without Person
- Fully normalized
- Optional & repeatable

2️⃣ ROLE LAYER (WHAT THE PERSON DOES)

🎓 Student Role

Person → Student

- Academic data only
- One Person → One Student (max)
- Identity reused

👩‍🏫 Staff Role

Person → Staff

- Employment / HR data only
- One Person → One Staff (max)

🔑 Golden Rule

Person (1) → (0 or 1) Student

Person (1) → (0 or 1) Staff

🚫 Never mix role data into Person

3️⃣ RELATIONSHIP LAYER (PERSON ↔ PERSON)

👨‍👩‍👧 Parent / Guardian Mapping

Parents are also Persons.

Student

└── ParentGuardian

├── student_id

└── parent_person_id → Person

Why this design?

- **One parent → many students**
- **One student → multiple guardians**
- **No duplicate parent records**

🔐 SECURITY LAYER (SYSTEM ACCESS)

🔐 UserAccount (OPTIONAL)

Person → UserAccount → Role

Purpose

- **Login & authentication only**
- **Role-based access control**

Login Eligibility

Person Type	Login
Student	Yes
Staff	Yes
Parent	Usually No
Admin	Yes

🔁 COMPLETE MODEL FLOW (ONE VIEW)

Person

├── Contact

├── Address

├── PersonDocument

├── UserAccount → Role

├── Student → ParentGuardian → Person (Parent)

└── Staff

🔍 REAL-TIME SCENARIOS

👴 New Admission

1. Create Person
2. Add Contact, Address
3. Upload Documents
4. Create Student
5. Map Parent

👨‍💼 New Staff Joining

1. Create Person
2. Add Contact, Address
3. Upload Documents
4. Create Staff
5. Create UserAccount
6. Assign Roles

🔄 Student Becomes Staff

- Same Person
- Student record kept (history)
- New Staff record created

🚦 DATA OWNERSHIP RULE (VERY IMPORTANT)

Data	Owner Model
Name, DOB	Person
Phone, Email	Contact
Address	Address
Aadhaar / Docs	PersonDocument
Roll No	Student

Data	Owner Model
Salary	Staff
Login	UserAccount
Permissions	Role

🧠 FINAL THUMB RULES

- ✅ **Person = MASTER**
- ✅ **Student / Staff = ROLE EXTENSIONS**
- ✅ **UserAccount = SECURITY ONLY**
- ✅ **Parent = Just another Person**
- ✅ **NO DUPLICATION ANYWHERE**

🔁 MASTER ENTITY FLOW (BIG PICTURE)

Person (WHO)

```
|
├── Contact        (HOW TO REACH)
├── Address        (WHERE THEY LIVE / WORK)
├── PersonDocument   (PROOFS & VERIFICATION)
|
├── UserAccount     (LOGIN – OPTIONAL)
|   └── UserRole → Role
|
├── Student        (ACADEMIC ROLE)
|   └── ParentGuardian → Person (Parent)
|
└── Staff          (EMPLOYMENT ROLE)
    ├── Teaching
    └── Non-Teaching
```

🧩 ENTITY CATEGORIES

Category	Tables
Identity Master	Person
Communication	Contact, Address
Compliance	PersonDocument
Security	UserAccount, Role, UserRole
Academic Role	Student
Employment Role	Staff
Relationship Mapping	ParentGuardian

🔹 STEP 1: PERSON (CORE MASTER)

PURPOSE

Stores **biographical identity only**
Used by **EVERY MODULE**

❌ Never store here

- Login
- Salary
- Marks
- Department

🧩 Table: person

Column	Type	Size	Purpose
person_id (PK)	BIGINT	—	Internal identity
first_name	VARCHAR	50	Given name
middle_name	VARCHAR	50	Optional
last_name	VARCHAR	50	Surname
gender	ENUM	—	Govt forms
date_of_birth	DATE	—	Age / eligibility
photo_url	VARCHAR	255	Profile photo

Column	Type	Size	Purpose
status	ENUM	—	Soft delete

🔁 **Relation**

Person → Contact (1:N)

Person → Address (1:N)

Person → PersonDocument (1:N)

Person → UserAccount (1:1)

Person → Student (1:1)

Person → Staff (1:1)

🔹 **STEP 2: CONTACT (COMMUNICATION)**

WHY SEPARATE?

- Multiple phones
- Multiple emails
- Verification
- Future WhatsApp / SMS

🧩 **Table: contact**

Column	Type	Size	Purpose
contact_id (PK)	BIGINT	—	Record ID
person_id (FK)	BIGINT	—	Owner
phone_no1	VARCHAR	15	Primary
phone_alt	VARCHAR	15	Alternate
email_no1	VARCHAR	100	Primary
email_alt	VARCHAR	100	Alternate
is_primary	BOOLEAN	—	Default contact
verified	BOOLEAN	—	OTP/Admin

🔹 **STEP 3: ADDRESS (MULTI-ADDRESS)**

WHY?

- Govt requires permanent
- Communication requires current
- Staff needs office

🧩 Table: address

Column	Type	Size	Purpose
address_id (PK)	BIGINT	—	Address record
person_id (FK)	BIGINT	—	Owner
address_type	ENUM	—	PERM / CUR / OFF
address_line1	VARCHAR	255	Full address
city	VARCHAR	50	City
state	VARCHAR	50	State
country	VARCHAR	50	Country
pincode	VARCHAR	10	Postal

🔹 STEP 4: PERSON DOCUMENT

USED FOR

- Admission verification
- HR onboarding
- Audits

🧩 Table: person_document

Column	Type	Size	Purpose
document_id (PK)	BIGINT	—	Document
person_id (FK)	BIGINT	—	Owner
document_type	ENUM	—	Aadhaar etc
document_path	VARCHAR	255	File

Column	Type	Size	Purpose
verified	BOOLEAN	—	Admin verified

🔹 **STEP 5: USER & SECURITY**

RULE

Person ≠ User

Role	Login
Student	Yes
Staff	Yes
Parent	Usually No

🧩 **Table: user_account**

Column	Type	Size	Purpose
user_id (PK)	BIGINT	—	Login
person_id (FK)	BIGINT	—	Identity
username	VARCHAR	50	Login ID
password_hash	VARCHAR	255	Encrypted
account_status	ENUM	—	Security

🧩 **Role Tables**

role

role_id	role_name
1	ADMIN
2	FACULTY
3	STUDENT

user_role (M:N)

| user_id | role_id |

🔹 **STEP 6: STUDENT (ACADEMIC ROLE)**

PURPOSE

Only **academic information**

🧩 **Table: student**

Column	Type	Purpose
student_id (PK)	BIGINT	Student
person_id (FK)	BIGINT	Identity
class_id	BIGINT	Academic
section_id	BIGINT	Class group
admission_year	INT	Reporting
roll_number	VARCHAR(20)	Unique
status	ENUM	Active/Passed

🔹 **STEP 7: STAFF (EMPLOYMENT ROLE)**

PURPOSE

Only **HR / Job data**

🧩 **Table: staff**

Column	Type	Purpose
staff_id (PK)	BIGINT	Staff
person_id (FK)	BIGINT	Identity
department_id	BIGINT	Org
designation	VARCHAR(50)	Job title
staff_type	ENUM	Teaching
experience_years	INT	HR

Column	Type	Purpose
date_of_joining	DATE	Service
employment_mode	ENUM	Permanent
salary	DECIMAL	Payroll
status	ENUM	Active

🔹 **STEP 8: PARENT / GUARDIAN**

❓ **WHY parent_person_id?**

Because:
- Parent is ALSO a human
- One parent → many students
- Parent data reused
- No duplicate parent table

🧩 **Table: parent_guardian**

Column	Type	Purpose
parent_guardian_id (PK)	BIGINT	Mapping
student_id (FK)	BIGINT	Child
parent_person_id (FK)	BIGINT	Parent
relation_type	ENUM	Father
is_primary	BOOLEAN	Contact
occupation	VARCHAR(100)	Info
annual_income	DECIMAL	Fees
status	ENUM	Active

🔍 **RELATION UNDERSTANDING TABLE (VERY IMPORTANT)**

From	To	Relation	Why
Person	Contact	1:N	Multiple phones

From	To	Relation	Why
Person	Address	1:N	Multi address
Person	Document	1:N	Multiple proofs
Person	User	1:1	Optional login
Person	Student	1:1	Academic
Person	Staff	1:1	Employment
Student	Parent	M:N	Family
User	Role	M:N	RBAC

🔁 **REAL DATA FETCH FLOW (EXAMPLE)**

🎓 **Student Profile Page**

Student

→ Person

→ Contact

→ Address

→ PersonDocument

→ ParentGuardian

→ Parent(Person)

👨‍🏫 **Staff Profile Page**

Staff

→ Person

→ Contact

→ Address

→ Documents

→ UserAccount

→ Roles

✅ COMMON ENUM

- 🔹 StatusEnum.java
- 🔹 GenderEnum.java
- 🔹 AddressTypeEnum.java
- 🔹 DocumentTypeEnum.java
- 🔹 AccountStatusEnum.java
- 🔹 StaffTypeEnum.java
- 🔹 EmploymentModeEnum.java
- 🔹 RelationTypeEnum.java

- 🔹 **StatusEnum.java**

```
package com.erp.common.enums;

/**
 * Common status enum for all entities
 */
public enum StatusEnum {
    ACTIVE,
    INACTIVE,
    LOCKED,
    DISABLED
}
```

- 🔹 **GenderEnum.java**

```
package com.erp.common.enums;
```

```
/**
 * Gender of person
 */
public enum GenderEnum {
    MALE,
    FEMALE,
    OTHER
}
```

- AddressTypeEnum.java

```
package com.erp.common.enums;

/**
 * Address types
 */
public enum AddressTypeEnum {
    PERMANENT,
    CURRENT,
    OFFICE
}
```

- **DocumentTypeEnum.java**

```
package com.erp.common.enums;

/**
 * Person document types
 */
public enum DocumentTypeEnum {
```

```
    AADHAAR,
    DEGREE,
    RESUME,
    ID_PROOF
}
```

- AccountStatusEnum.java

```
package com.erp.common.enums;

/**
 * User account status
 */
public enum AccountStatusEnum {
    ACTIVE,
    LOCKED,
    DISABLED
}
```

- StaffTypeEnum.java

```
package com.erp.common.enums;

/**
 * Staff category
 */
public enum StaffTypeEnum {
    TEACHING,
    NON_TEACHING
```

}

🔹 **EmploymentModeEnum.java**

```
package com.erp.common.enums;

/**
 * Staff employment mode
 */
public enum EmploymentModeEnum {
   PERMANENT,
   CONTRACT
}
```

🔹 **RelationTypeEnum.java**

```
package com.erp.common.enums;

/**
 * Parent / Guardian relation
 */
public enum RelationTypeEnum {
   FATHER,
   MOTHER,
   GUARDIAN
}
```

Person

Stores who the person is, not what they do.

Used by:
• Student
• Staff
• Parent
• Admin

```
package com.erp.person.entity;

import com.erp.common.audit.BaseAuditEntity;
import com.erp.common.enums.GenderEnum;
import com.erp.common.enums.StatusEnum;
import jakarta.persistence.*;
import jakarta.validation.constraints.*;
import lombok.*;

import java.time.LocalDate;

/**
 * Person
 * ------
 * Central identity master for all humans in ERP.
 * Created ONCE and reused across modules.
 */
@Entity
@Table(name = "person")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Person extends BaseAuditEntity {
```

```
/** Surrogate primary key for internal reference */
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long personId;

/**
 * First name of the person
 * Example: "Rohan"
 */
@NotBlank
@Column(length = 50, nullable = false)
private String firstName;

/**
 * Middle name (optional)
 * Example: "Kumar"
 */
@Column(length = 50)
private String middleName;

/**
 * Last name / surname
 * Example: "Sharma"
 */
@NotBlank
@Column(length = 50, nullable = false)
private String lastName;

/**
 * Biological gender
```

```
     * Used for reports, government forms
     */
    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(length = 10, nullable = false)
    private GenderEnum gender;

    /**
     * Date of birth
     * Used for age calculation, eligibility
     */
    private LocalDate dateOfBirth;

    /**
     * Stored file path of profile photo
     * Example: uploads/person/123.jpg
     */
    @Column(length = 255)
    private String photoUrl;

    /**
     * Logical status of person record
     * INACTIVE → soft delete
     */
    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(length = 10, nullable = false)
    private StatusEnum status;
}
```

🔹 **STEP 2: CONTACT (COMMUNICATION DETAILS)**

• Multiple phone numbers
• Multiple emails
• Verification tracking

```
/**
 * Contact
 * -------
 * Stores communication details of a person.
 */
@Entity
@Table(name = "contact")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Contact extends BaseAuditEntity {

  /** Primary key */
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long contactId;

  /**
   * Owner of this contact
   * Many contacts → One person
   */
  @NotNull
  @ManyToOne
  @JoinColumn(name = "person_id", nullable = false)
  private Person person;

  /**
```

```
     * Primary mobile number
     * Stored as VARCHAR to support country codes
     */
    @Column(length = 15)
    private String phoneNo1;

    /**
     * Alternate mobile number
     */
    @Column(length = 15)
    private String phoneAlt;

    /**
     * Primary email ID
     */
    @Column(length = 100)
    private String emailNo1;

    /**
     * Alternate email ID
     */
    @Column(length = 100)
    private String emailAlt;

    /**
     * Indicates default contact for communication
     */
    private Boolean isPrimary = false;

    /**
     * Whether contact is verified (OTP/Admin)
```

```
    */
    private Boolean verified = false;
}
```

🔹 STEP 3: ADDRESS (MULTIPLE ADDRESS TYPES)

• Permanent address (government)
• Current address (communication)
• Office address (staff)

```
/**
 * Address
 * -------
 * Stores multiple addresses for a person.
 */
@Entity
@Table(name = "address")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Address extends BaseAuditEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long addressId;

    /**
     * Owner of address
     */
```

```
@NotNull
@ManyToOne
@JoinColumn(name = "person_id", nullable = false)
private Person person;

/**
 * Type of address
 */
@NotNull
@Enumerated(EnumType.STRING)
@Column(length = 15)
private AddressTypeEnum addressType;

/**
 * Full address line
 */
@Column(length = 255)
private String addressLine1;

/** City name */
@Column(length = 50)
private String city;

/** State name */
@Column(length = 50)
private String state;

/** Country name */
@Column(length = 50)
private String country;
```

```
    /**
     * Postal / ZIP code
     */
    @Column(length = 10)
    private String pincode;
}
```

🔹 STEP 4: PERSON DOCUMENT

📌 USED FOR

- Admission verification
- HR onboarding
- Compliance audits

```
/**
 * PersonDocument
 * --------------
 * Stores uploaded documents of a person.
 */
@Entity
@Table(name = "person_document")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class PersonDocument extends BaseAuditEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long documentId;

    /**
```

```
    * Document owner
    */
    @NotNull
    @ManyToOne
    @JoinColumn(name = "person_id", nullable = false)
    private Person person;

    /**
    * Type of document
    */
    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private DocumentTypeEnum documentType;

    /**
    * File storage path
    */
    @NotBlank
    @Column(length = 255, nullable = false)
    private String documentPath;

    /**
    * Verified by admin or authority
    */
    private Boolean verified = false;
}
```

- **STEP 5: USER ACCOUNT (AUTH)**

• Parents may not login
• One identity → zero or one account

```
/**
 * UserAccount
 * -----------
 * Authentication entity.
 */
@Entity
@Table(name = "user_account")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UserAccount extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long userId;

  /**
   * Identity linked to this login
   */
  @NotNull
  @OneToOne
  @JoinColumn(name = "person_id", nullable = false, unique = true)
  private Person person;

  /**
   * Login username
   */
```

```
    @NotBlank
    @Column(length = 50, nullable = false, unique = true)
    private String username;

    /**
     * BCrypt encrypted password
     */
    @NotBlank
    @Column(length = 255, nullable = false)
    private String passwordHash;

    /**
     * Account status for security
     */
    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(length = 15)
    private AccountStatusEnum accountStatus;
}
```

- STEP 7: STAFF (EMPLOYMENT ROLE)

```
/**
 * Staff
 * -----
 * Employment data of a person.
 */
```

```
@Entity
@Table(name = "staff")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Staff extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long staffId;

  /**
   * Identity of staff member
   */
  @NotNull
  @OneToOne
  @JoinColumn(name = "person_id", nullable = false, unique = true)
  private Person person;

  /** Department reference */
  private Long departmentId;

  /**
   * Job title
   * Example: Professor, Clerk
   */
  @Column(length = 50)
  private String designation;

  /** Teaching / Non-teaching */
```

```
    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private StaffTypeEnum staffType;

    /** Total experience in years */
    private Integer experienceYears;

    /** Date of joining */
    private java.time.LocalDate dateOfJoining;

    /** Employment type */
    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private EmploymentModeEnum employmentMode;

    /** Monthly salary */
    private Double salary;

    /** Active or inactive employee */
    @Enumerated(EnumType.STRING)
    @Column(length = 15)
    private StatusEnum status;
}
```

🔹 STEP 8: PARENT / GUARDIAN MAPPING

WHY parent_person_id EXISTS
Because **parent is also a Person**, reused across students.

```
/**
```

```
 * ParentGuardian
 * ----------
 * Maps student with parent/guardian.
 */
@Entity
@Table(name = "parent_guardian")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ParentGuardian extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long parentGuardianId;

  /**
   * Student reference
   */
  @NotNull
  @ManyToOne
  @JoinColumn(name = "student_id", nullable = false)
  private Student student;

  /**
   * Parent identity (Person)
   */
  @NotNull
  @ManyToOne
  @JoinColumn(name = "parent_person_id", nullable = false)
  private Person parentPerson;
```

```
/**
 * Relation with student
 */
@Enumerated(EnumType.STRING)
@Column(length = 20)
private RelationTypeEnum relationType;

/**
 * Primary guardian for communication
 */
private Boolean isPrimary = false;

/**
 * Parent occupation
 */
@Column(length = 100)
private String occupation;

/**
 * Annual income
 * Used for scholarship/fees
 */
private Double annualIncome;

/**
 * Active mapping status
 */
@Enumerated(EnumType.STRING)
@Column(length = 15)
private StatusEnum status;
```

}

🧩 **Entity: Student**

Table: student

Field Name	Data Type	Description
student_id (PK)	BIGINT	Student record
person_id (FK)	BIGINT	Identity
class_id (FK)	BIGINT	Academic class
section_id (FK)	BIGINT	Section
admission_year	INT	Admission year
roll_number	VARCHAR(20)	Unique roll
status	ENUM	ACTIVE / PASSED

🔗 **Relationship**

- **Person 1 → 1 Student**

🔹 **STEP 6: STUDENT**

```
/**
 * Student
 * -------
 * Academic role of a person.
 */
@Entity
@Table(name = "student")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Student extends BaseAuditEntity {
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long studentId;

/**
 * Identity of student
 */
@NotNull
@OneToOne
@JoinColumn(name = "person_id", nullable = false, unique = true)
private Person person;

/** Class mapping */
private Long classId;

/** Section mapping */
private Long sectionId;

/**
 * Year of admission
 */
private Integer admissionYear;

/**
 * Unique roll number
 */
@Column(length = 20, unique = true)
private String rollNumber;

/**
 * Academic status
```

```
    */
    @Enumerated(EnumType.STRING)
    @Column(length = 15)
    private StatusEnum status;
}
```

Part 3.

✅ STEP : TeachingAssignment (FACULTY ↔ SUBJECT ↔ CLASS)

PURPOSE

TeachingAssignment defines:

Which staff member teaches which subject,
for which class & section,
in which semester & academic year,
under which regulation.

This entity is used by:

- Timetable
- Attendance
- Exams
- Faculty workload
- Reports

📌 WHY THIS ENTITY IS REQUIRED

Without this table:

- You **cannot** mark attendance correctly
- You **cannot** generate faculty timetable
- You **cannot** calculate teaching load
- You **cannot** restrict subject access

🔗 DEPENDENCIES

Staff

Subject

ClassEntity

Section (optional)

Semester

AcademicYear

Regulation

1. One subject + one class + one semester
   → **cannot be taught by two staff at the same time**
2. One staff can teach **multiple subjects**
3. Same subject can be taught in **multiple sections**
4. Regulation must match the class/semester regulation

🛢 DATABASE CONSTRAINT (UNIQUE)

(subject_id, class_id, section_id, semester_id)

Ensures **no duplicate teaching allocation**.

🧩 JPA ENTITY — TeachingAssignment

```
@Entity
@Table(
  name = "teaching_assignment",
  uniqueConstraints = {
    @UniqueConstraint(
      columnNames = {"subject_id", "class_id", "section_id", "semester_id"}
    )
  }
)
public class TeachingAssignment extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long teachingAssignmentId;
```

```
@ManyToOne
@JoinColumn(name = "staff_id", nullable = false)
private Staff staff;

@ManyToOne
@JoinColumn(name = "subject_id", nullable = false)
private Subject subject;

@ManyToOne
@JoinColumn(name = "class_id", nullable = false)
private ClassEntity classEntity;

@ManyToOne
@JoinColumn(name = "section_id")
private Section section;

@ManyToOne
@JoinColumn(name = "semester_id", nullable = false)
private Semester semester;

@ManyToOne
@JoinColumn(name = "academic_year_id", nullable = false)
private AcademicYear academicYear;

@ManyToOne
@JoinColumn(name = "regulation_id", nullable = false)
private Regulation regulation;

private Integer weeklyHours;
```

```
    @Enumerated(EnumType.STRING)
    @Column(length = 15, nullable = false)
    private Status status;
}
```

👩‍🏫 Faculty Assignment

1. Create Person → Staff
2. Assign Subject to Staff via **TeachingAssignment**
3. Timetable & attendance auto-use this mapping

🔁 DATA FETCH FLOW (IMPORTANT)

Faculty Timetable

Staff

→ TeachingAssignment

→ Subject

→ Class

→ Section

→ Semester

Student Attendance

Student

→ Class + Section

→ TeachingAssignment

→ Subject

→ Staff

🚦 FINAL THUMB RULE

Question	Answer
Who teaches?	Staff
What is taught?	Subject

Question	Answer
Where?	Class + Section
When?	Semester + AcademicYear
Under what rules?	Regulation

👉 **All answered by TeachingAssignment**

🔹 **STEP : StudentEnrollment**

(Student → Class → Section → Semester)

🧠 **PURPOSE (REAL-LIFE MEANING)**

StudentEnrollment defines:

Which student is studying
in which class & section,
for which semester & academic year,
under which regulation.

Without this table:

- ❌ Attendance impossible
- ❌ Exams impossible
- ❌ Promotion impossible
- ❌ Result mapping impossible

🔗 **MODEL FLOW (VERY SIMPLE)**

Person
└── Student
└── StudentEnrollment
├── Class
├── Section
├── Semester
├── AcademicYear
└── Regulation

1️⃣ **Student table**
2️⃣ **StudentEnrollment table (Student → Class → Section → Semester)**
sizes, comments, real-life meaning.

✅ 1️⃣ STUDENT ENTITY (FINAL)

Student = Academic role of a Person
No personal/contact/address data here ❌
Only admission & academic identity ✅

```
@Entity
@Table(
  name = "student",
  uniqueConstraints = {
    @UniqueConstraint(columnNames = "person_id"),
    @UniqueConstraint(columnNames = "registration_number")
  }
)
public class Student extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long studentId;

  @OneToOne
  @JoinColumn(name = "person_id", nullable = false, unique = true)
  private Person person;

  @Column(length = 30, nullable = false)
  private String registrationNumber;

  @Column(length = 30, nullable = false)
  private String admissionNumber;
```

```
    private LocalDate admissionDate;

    @ManyToOne
    @JoinColumn(name = "course_id", nullable = false)
    private Course course;

    @Enumerated(EnumType.STRING)
    @Column(length = 15, nullable = false)
    private StatusEnum status;
}
```

🔁 **WHY STUDENT ENROLLMENT IS NEEDED**

❌ **Don't store class/section/semester in Student**
Because student **changes every year/semester**

✓ History required
✓ Promotions required
✓ Backlog / repeat possible

So we use **StudentEnrollment**

✅ 2️⃣ **STUDENT ENROLLMENT ENTITY (VERY IMPORTANT)**

Tracks where the student is studying NOW (and in past)

StudentEnrollment

Student → Semester → Class → Section

```
@Entity
@Table(
    name = "student_enrollment",
    uniqueConstraints = {
        @UniqueConstraint(columnNames = {"student_id", "semester_id"})
    },
    indexes = {
        @Index(columnList = "student_id"),
```

```
        @Index(columnList = "class_id")
    }
)
public class StudentEnrollment extends BaseAuditEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long studentEnrollmentId;

    @ManyToOne
    @JoinColumn(name = "student_id", nullable = false)
    private Student student;

    @ManyToOne
    @JoinColumn(name = "class_id", nullable = false)
    private ClassEntity classEntity;

    @ManyToOne
    @JoinColumn(name = "section_id")
    private Section section;

    @ManyToOne
    @JoinColumn(name = "semester_id", nullable = false)
    private Semester semester;

    @ManyToOne
    @JoinColumn(name = "academic_year_id", nullable = false)
    private AcademicYear academicYear;

    @ManyToOne
    @JoinColumn(name = "regulation_id", nullable = false)
```

```
    private Regulation regulation;

    @Column(length = 20, nullable = false)
    private String rollNumber;

    @Enumerated(EnumType.STRING)
    @Column(length = 15, nullable = false)
    private Status status;
}
```

COMPLETE STUDENT FLOW (SHORT)

Person
└── Student
    └── StudentEnrollment
        ├── Semester
        ├── Class
        └── Section

📘 TIMETABLE MODULE

(ERP Documentation – Academic Operations)

1️⃣ OVERVIEW

The **Timetable module** defines:

- **When** a class is conducted
- **Which subject** is taught
- **Which faculty** teaches
- **Which class & section** attends

Timetable is a **runtime operational layer** built on top of **TeachingAssignment** and **StudentEnrollment**.

2️⃣ WHY TIMETABLE IS A SEPARATE ENTITY

❌ Timetable data must **NOT** be stored in:

- TeachingAssignment (that is static allocation)
- StudentEnrollment (that is student placement)

✓ Timetable changes:

- Daily / weekly
- Room changes
- Faculty substitution
- Extra / remedial classes

👉 Hence, **Timetable = independent operational entity**

DEPENDENCY FLOW

TeachingAssignment

├— Staff

├— Subject

├— Class

├— Section

└— Semester

Timetable

└— TeachingAssignment

ENTITY RELATIONSHIP VIEW

Student

└— StudentEnrollment

└── Class + Section

Staff

└── TeachingAssignment

└── Subject

Timetable

└── TeachingAssignment

└── Time + Day + Room

5️⃣ BUSINESS RULES (VERY IMPORTANT)

🔒 Golden Rules

1. One class + section **cannot have two lectures at same time**
2. One faculty **cannot teach two classes at same time**
3. Timetable must always reference **TeachingAssignment**
4. Attendance can be marked **only if timetable exists**

6️⃣ TIMETABLE ENTITY (FINAL)

🧾 JPA ENTITY — Timetable

```
package com.erp.academic.entity;


import com.erp.common.audit.BaseAuditEntity;

import com.erp.common.enums.StatusEnum;

import jakarta.persistence.*;

import jakarta.validation.constraints.*;

import lombok.*;


import java.time.DayOfWeek;

import java.time.LocalTime;
```

```
/**
 * Timetable
 * ---------
 * Defines scheduled lecture slots
 * for a class, subject, and faculty.
 *
 * Used by:
 * - Attendance
 * - Faculty timetable
 * - Student timetable
 */
@Entity
@Table(
  name = "timetable",
  uniqueConstraints = {
    @UniqueConstraint(
      name = "uk_class_time_slot",
      columnNames = {
        "class_id",
        "section_id",
        "day_of_week",
        "start_time"
      }
    ),
    @UniqueConstraint(
      name = "uk_staff_time_slot",
      columnNames = {
        "staff_id",
        "day_of_week",
        "start_time"
      }
```

```
    )
  },
  indexes = {
    @Index(name = "idx_tt_class", columnList = "class_id"),
    @Index(name = "idx_tt_staff", columnList = "staff_id"),
    @Index(name = "idx_tt_assignment", columnList = "teaching_assignment_id")
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Timetable extends BaseAuditEntity {

  /** Primary key */
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long timetableId;

  /**
   * Teaching assignment reference
   * (who teaches what to which class)
   */
  @NotNull
  @ManyToOne
  @JoinColumn(name = "teaching_assignment_id", nullable = false)
  private TeachingAssignment teachingAssignment;

  /**
   * Redundant but optimized access (derived from TeachingAssignment)
   */
```

```
@NotNull
@ManyToOne
@JoinColumn(name = "staff_id", nullable = false)
private Staff staff;

@NotNull
@ManyToOne
@JoinColumn(name = "class_id", nullable = false)
private ClassEntity classEntity;

@ManyToOne
@JoinColumn(name = "section_id")
private Section section;

/**
 * Day of week
 * MONDAY – SATURDAY
 */
@NotNull
@Enumerated(EnumType.STRING)
@Column(length = 10, nullable = false)
private DayOfWeek dayOfWeek;

/**
 * Lecture start time
 * Example: 09:00
 */
@NotNull
private LocalTime startTime;

/**
```

```
    * Lecture end time
    * Example: 09:50
    */
    @NotNull
    private LocalTime endTime;

    /**
    * Room / Lab number
    * Example: CSE-LAB-2, R-301
    */
    @Column(length = 30)
    private String roomNo;

    /**
    * Timetable status
    * ACTIVE / INACTIVE
    */
    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(length = 15, nullable = false)
    private StatusEnum status;
}
```

7️⃣ **REAL-TIME FLOW**

🗓️ **Timetable Creation**

1. Create **TeachingAssignment**
2. Create **Timetable slots**
3. Assign time + room
4. Publish timetable

👩‍🎓 **Student View**

Student

→ StudentEnrollment

→ Class + Section

→ Timetable

→ Subject + Faculty + Time

👩‍🏫 Faculty View

Staff

→ TeachingAssignment

→ Timetable

→ Class + Subject + Time

8️⃣ ATTENDANCE DEPENDENCY (CRITICAL)

Attendance

└── Timetable

  └── TeachingAssignment

    └── Subject + Staff

❌ No timetable → ❌ No attendance

9️⃣ FINAL THUMB RULE

Question	Answer
Who teaches?	TeachingAssignment
When is class?	Timetable
Where?	Room
Who attends?	StudentEnrollment
Can attendance start?	Only if timetable exists

🔜 NEXT MODULE (RECOMMENDED)

➡ Attendance
(because it directly depends on Timetable)

Just say:
Next: Attendance 👌

Then we move **STEP 2️⃣ : ATTENDANCE — short, clear, production-ready.**

📘 ATTENDANCE MODULE

(Depends on Timetable)

1️⃣ PURPOSE (WHY ATTENDANCE EXISTS)

Attendance records:

- **Which student**
- **Attended which lecture**
- **On which date**
- **For which timetable slot**

👉 Attendance is **transactional data** (daily).

2️⃣ HARD DEPENDENCY FLOW

Student
└── StudentEnrollment
    └── Class + Section

Timetable
└── TeachingAssignment
    └── Subject + Faculty

Attendance
├── Student
└── Timetable

❌ No Timetable → ❌ No Attendance
❌ Student not enrolled → ❌ No Attendance

3️⃣ BUSINESS RULES (VERY IMPORTANT)

1. Attendance **must reference Timetable**
2. One student → **one record per timetable per date**
3. Faculty can mark attendance **only for assigned timetable**
4. Attendance is **immutable after lock (optional rule)**

4️⃣ ATTENDANCE ENTITY (FINAL JPA)

```
package com.erp.academic.entity;


import com.erp.common.audit.BaseAuditEntity;
import com.erp.common.enums.AttendanceStatus;
import jakarta.persistence.*;
import jakarta.validation.constraints.NotNull;
import lombok.*;


import java.time.LocalDate;


/**
 * Attendance
 * ----------
 * Stores student presence for a specific
 * timetable lecture on a given date.
 */
@Entity
@Table(
   name = "attendance",
   uniqueConstraints = {
      @UniqueConstraint(
         name = "uk_attendance_student_slot",
         columnNames = {
```

```
            "student_id",
            "timetable_id",
            "attendance_date"
        }
    )
  },
  indexes = {
    @Index(name = "idx_att_student", columnList = "student_id"),
    @Index(name = "idx_att_timetable", columnList = "timetable_id")
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Attendance extends BaseAuditEntity {

  /** Primary key */
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long attendanceId;

  /**
   * Student whose attendance is recorded
   */
  @NotNull
  @ManyToOne
  @JoinColumn(name = "student_id", nullable = false)
  private Student student;

  /**
```

```
    * Timetable slot for which attendance is marked
    */
    @NotNull
    @ManyToOne
    @JoinColumn(name = "timetable_id", nullable = false)
    private Timetable timetable;

    /**
    * Date of lecture
    * Example: 2025-01-15
    */
    @NotNull
    @Column(name = "attendance_date", nullable = false)
    private LocalDate attendanceDate;

    /**
    * Attendance status
    * PRESENT / ABSENT / LEAVE
    */
    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(length = 10, nullable = false)
    private AttendanceStatus status;
}
```

5️⃣ ENUM — AttendanceStatus

```
package com.erp.common.enums;

/**
 * AttendanceStatus
 * ----------------
```

```
 * Represents student presence state.
 */
public enum AttendanceStatus {
    PRESENT,
    ABSENT,
    LEAVE
}
```

6️⃣ REAL-LIFE FLOW (STEP BY STEP)

👩‍🏫 Faculty Marks Attendance

Login

→ View My Timetable (Today)

→ Select Lecture

→ System loads students (via StudentEnrollment)

→ Mark attendance

→ Save

👨‍🎓 Student View

Login

→ My Attendance

→ Subject-wise %

→ Monthly report

7️⃣ WHY ATTENDANCE IS PERFECTLY NORMALIZED

Data	Stored Where
Student info	Student
Class/Section	StudentEnrollment
Subject	TeachingAssignment
Faculty	TeachingAssignment

Data	Stored Where
Time	Timetable
Presence	Attendance

✓ No duplication
✓ No update anomaly
✓ Scalable

TeachingAssignment defines responsibility

Timetable defines schedule

Attendance records execution

----------....

Part 4

➡️ **NEXT LOGICAL MODULE**

➞ Exam / Internal Assessment

4️⃣ **COMPLETE REAL-TIME FLOW (STEP BY STEP)**

👩‍🏫 **Faculty Side (Teacher)**

1. Faculty logs in
2. Selects Subject → Data Structures
3. Selects Assessment → Unit Test 1
4. System fetches students using:

   StudentEnrollment (Sem 3, Section A)
5. Faculty enters marks
6. Data saved in StudentAssessmentResult

🎓 **Student Side (Rahul)**

1. Rahul logs in

2. Clicks "My Results"

3. Selects Semester 3

4. System shows:

Data Structures

├── Unit Test 1 : 24 / 30

├── Mid Sem : 21 / 30

└── End Sem : 68 / 100

Student

→ StudentEnrollment

→ Semester

→ Assessment

→ StudentAssessmentResult

📘 EXAM / INTERNAL ASSESSMENT – MODEL FLOW

🔁 OVERALL FLOW (ONE LINE)

Student

└── StudentEnrollment

└── Semester

└── Assessment (Internal / External)

└── StudentAssessmentResult

```

---

## 1️⃣ AssessmentType (MASTER)

### 👉 WHY
```

Defines **what kind of exam it is**.

```java
public enum AssessmentType {
   INTERNAL,   // Unit Test, Mid Sem, Assignment
   EXTERNAL    // End Semester Exam
}
```

2️⃣ Assessment (EXAM DEFINITION)

> Defines **an exam for a subject in a semester**

```java
package com.erp.academic.entity;

import com.erp.common.audit.BaseAuditEntity;
import com.erp.common.enums.AssessmentType;
import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDate;

/**
 * Assessment
 * ----------
 * Represents an exam or internal assessment
 * conducted for a subject in a semester.
 */
```

```
@Entity
@Table(
  name = "assessment",
  uniqueConstraints = {
    @UniqueConstraint(
      name = "uk_assessment_subject_sem_type",
      columnNames = {
        "subject_id",
        "semester_id",
        "assessment_type",
        "assessment_name"
      }
    )
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Assessment extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long assessmentId;

  /** Internal / External */
  @Enumerated(EnumType.STRING)
  @Column(length = 20, nullable = false)
  private AssessmentType assessmentType;

  /**
```

```
     * Name of assessment
     * Examples:
     * "Unit Test 1"
     * "Mid Sem"
     * "End Sem"
     */
    @Column(length = 50, nullable = false)
    private String assessmentName;

    /** Subject for which exam is conducted */
    @ManyToOne
    @JoinColumn(name = "subject_id", nullable = false)
    private Subject subject;

    /** Semester */
    @ManyToOne
    @JoinColumn(name = "semester_id", nullable = false)
    private Semester semester;

    /** Maximum marks */
    @Column(nullable = false)
    private Integer maxMarks;

    /** Passing marks */
    @Column(nullable = false)
    private Integer passMarks;

    /** Exam date */
    private LocalDate examDate;
}
```

3️⃣ StudentAssessmentResult (TRANSACTION)

> Stores **marks of each student**

```java
package com.erp.academic.entity;

import com.erp.common.audit.BaseAuditEntity;
import jakarta.persistence.*;
import lombok.*;

/**
 * StudentAssessmentResult
 * -----------------------
 * Stores marks obtained by a student
 * in a particular assessment.
 */
@Entity
@Table(
    name = "student_assessment_result",
    uniqueConstraints = {
        @UniqueConstraint(
            name = "uk_student_assessment",
            columnNames = {"student_id", "assessment_id"}
        )
    }
)
@Data
```

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class StudentAssessmentResult extends BaseAuditEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long resultId;

    /** Student */
    @ManyToOne
    @JoinColumn(name = "student_id", nullable = false)
    private Student student;

    /** Assessment */
    @ManyToOne
    @JoinColumn(name = "assessment_id", nullable = false)
    private Assessment assessment;

    /** Marks obtained */
    @Column(nullable = false)
    private Integer marksObtained;

    /** Absent flag */
    @Column(nullable = false)
    private Boolean absent = false;
}
```

4️⃣ REAL-TIME SCENARIOS

📝 Internal Marks Entry

Faculty Login

→ Select Subject

→ Select Assessment (Unit Test 1)

→ System fetches students (Enrollment)

→ Enter marks

→ Save

🎓 Student View

Login

→ My Results

→ Subject-wise marks

→ Internal + External breakdown i want this again but with with real-life example

6️⃣ WHY THIS DESIGN IS CORRECT ✅

✓ One assessment → many students

✓ One student → many assessments

✓ Internals & externals handled cleanly

✓ No duplication

✓ Future-proof (CGPA, grading, moderation)

🎯 RESULT CALCULATION MODULE

(Internal + External → Final Grade)

🔁 REAL-LIFE FLOW (ONE LINE)

StudentAssessmentResult (Internal + External)

→ SubjectResult

→ SGPA

→ CGPA

1️⃣ SubjectResult (FINAL RESULT PER SUBJECT)

🧠 REAL-LIFE MEANING

Final marks & grade of a student for ONE subject in ONE semester

This is what appears on:

- **Marksheet**
- **Transcript**
- **Result portal**

📄 TABLE: subject_result

Column	Purpose
student_id	Whose result
subject_id	Which subject
semester_id	Which semester
internal_marks	Sum of internals
external_marks	End sem marks
total_marks	Internal + External
grade	A / B / C / F
grade_point	10 / 9 / 8 etc
status	PASS / FAIL

✅ JPA ENTITY — SubjectResult

```
package com.erp.result.entity;


import com.erp.common.audit.BaseAuditEntity;

import com.erp.common.enums.StatusEnum;

import com.erp.student.entity.Student;

import com.erp.academic.entity.Subject;

import com.erp.academic.entity.Semester;
```

```
import jakarta.persistence.*;
import lombok.*;

/**
 * SubjectResult
 * -------------
 * Final result of a student for one subject in a semester.
 */
@Entity
@Table(
  name = "subject_result",
  uniqueConstraints = {
    @UniqueConstraint(
      name = "uk_student_subject_sem",
      columnNames = {"student_id", "subject_id", "semester_id"}
    )
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class SubjectResult extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long subjectResultId;

  /** Student */
  @ManyToOne
  @JoinColumn(name = "student_id", nullable = false)
```

```
private Student student;

/** Subject */
@ManyToOne
@JoinColumn(name = "subject_id", nullable = false)
private Subject subject;

/** Semester */
@ManyToOne
@JoinColumn(name = "semester_id", nullable = false)
private Semester semester;

/** Total internal marks */
private Integer internalMarks;

/** External (end sem) marks */
private Integer externalMarks;

/** Internal + External */
private Integer totalMarks;

/** Grade (A, B, C, F) */
@Column(length = 5)
private String grade;

/** Grade point (10, 9, 8...) */
private Integer gradePoint;

/** PASS / FAIL */
@Enumerated(EnumType.STRING)
@Column(length = 15)
```

```
    private StatusEnum status;
}
```

🧠 **REAL-LIFE EXAMPLE (SUBJECT RESULT)**

Rahul Kumar – Data Structures – Sem 3

Component	Marks
Internal	25
External	68
Total	93
Grade	A
Grade Point	10
Status	PASS

👉 **Stored in SubjectResult**

2️⃣ **SemesterResult (SGPA)**

🧠 **REAL-LIFE MEANING**

Overall performance of a student in ONE semester

📄 **TABLE: semester_result**

Column	Meaning
student_id	Student
semester_id	Semester
total_credits	Credits attempted
earned_credits	Credits passed
sgpa	Semester GPA
status	PASS / FAIL

✅ **JPA ENTITY — SemesterResult**

```
package com.erp.result.entity;

import com.erp.common.audit.BaseAuditEntity;
import com.erp.common.enums.StatusEnum;
import com.erp.student.entity.Student;
import com.erp.academic.entity.Semester;
import jakarta.persistence.*;
import lombok.*;

/**
 * SemesterResult
 * --------------
 * Stores SGPA of a student for a semester.
 */
@Entity
@Table(
  name = "semester_result",
  uniqueConstraints = {
    @UniqueConstraint(
      name = "uk_student_semester",
      columnNames = {"student_id", "semester_id"}
    )
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class SemesterResult extends BaseAuditEntity {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long semesterResultId;

    @ManyToOne
    @JoinColumn(name = "student_id", nullable = false)
    private Student student;

    @ManyToOne
    @JoinColumn(name = "semester_id", nullable = false)
    private Semester semester;

    /** Total credits registered */
    private Integer totalCredits;

    /** Credits earned (passed subjects) */
    private Integer earnedCredits;

    /** Semester GPA */
    private Double sgpa;

    /** PASS / FAIL */
    @Enumerated(EnumType.STRING)
    @Column(length = 15)
    private StatusEnum status;
}
```

🧠 **REAL-LIFE SGPA CALCULATION**

SGPA =

Σ (Subject Credit × Grade Point)

Σ Subject Credits

Example:

(4×10 + 4×9 + 3×8) / (4+4+3) = 9.27

3️⃣ CGPA (OVERALL PERFORMANCE)

🧠 REAL-LIFE MEANING

Complete academic performance till date

📄 TABLE: cgpa_result

Column	Meaning
student_id	Student
total_credits	Credits till now
cgpa	Final CGPA
status	ACTIVE / COMPLETED

✅ JPA ENTITY — CgpaResult

```
package com.erp.result.entity;


import com.erp.common.audit.BaseAuditEntity;
import com.erp.common.enums.StatusEnum;
import com.erp.student.entity.Student;
import jakarta.persistence.*;
import lombok.*;


/**
 * CgpaResult
 * ----------
 * Stores cumulative GPA of a student.
 */
```

```
@Entity
@Table(
  name = "cgpa_result",
  uniqueConstraints = {
    @UniqueConstraint(
      name = "uk_cgpa_student",
      columnNames = {"student_id"}
    )
  }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CgpaResult extends BaseAuditEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long cgpaResultId;

  @OneToOne
  @JoinColumn(name = "student_id", nullable = false)
  private Student student;

  /** Total credits completed */
  private Integer totalCredits;

  /** Cumulative GPA */
  private Double cgpa;

  /** ACTIVE / COMPLETED */
```

```
    @Enumerated(EnumType.STRING)
    @Column(length = 15)
    private StatusEnum status;
}
```

4️⃣ REVALUATION / IMPROVEMENT EXAM (DESIGN READY)

🧠 WHY NEEDED

- Student unhappy with marks
- Backlog clearance
- Improvement attempt

📄 TABLE: revaluation_request

```
package com.erp.result.entity;

import com.erp.common.audit.BaseAuditEntity;
import com.erp.student.entity.Student;
import com.erp.academic.entity.Assessment;
import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "revaluation_request")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class RevaluationRequest extends BaseAuditEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long requestId;
```

```
    @ManyToOne
    private Student student;

    @ManyToOne
    private Assessment assessment;

    /** Reason by student */
    @Column(length = 255)
    private String reason;

    /** Approved / Rejected */
    @Column(length = 20)
    private String status;
}
```

🧠 **COMPLETE RESULT FLOW (FINAL)**

Assessment
→ StudentAssessmentResult
→ SubjectResult
→ SemesterResult (SGPA)
→ CgpaResult
