1) Functional Requirements:

User Management:

The system should allow the creation and management of user accounts for students, faculty, and staff.

Users should be able to authenticate and log in to the system with their credentials.

The system should enforce different roles and permissions for different user types.

Course Management:

The system should support the creation, modification, and deletion of courses offered by the university.

Courses should have information such as title, description, credits, prerequisites, and instructor.

Students should be able to enroll in courses, drop courses, and view their course schedule.

Grade Management:

The system should allow instructors to enter and update grades for students in their courses.

Students should be able to view their grades for the courses they are enrolled in.

The system should calculate and maintain a student's GPA based on their grades.

Academic Records:

The system should store and manage academic records, including student information, transcripts, and course history.

Faculty and staff should be able to access and update student records as necessary.

The system should support the generation of official documents, such as transcripts and degree certificates.

Timetable Management:

The system should create and manage the university's class timetable.

It should ensure that there are no scheduling conflicts between courses and instructors.

Students should be able to view the timetable to know the schedule of their courses.

2) Non-functional Requirements:

Performance:

The system should handle a large number of concurrent users efficiently.

Response times should be fast to ensure a smooth user experience.

Security:

User data, such as passwords and personal information, should be securely stored and protected.

Access to sensitive data should be restricted based on user roles and permissions.

The system should protect against common security vulnerabilities, such as cross-site scripting (XSS) and SQL injection.

Scalability:

The system should be scalable to accommodate future growth in the number of users and courses.

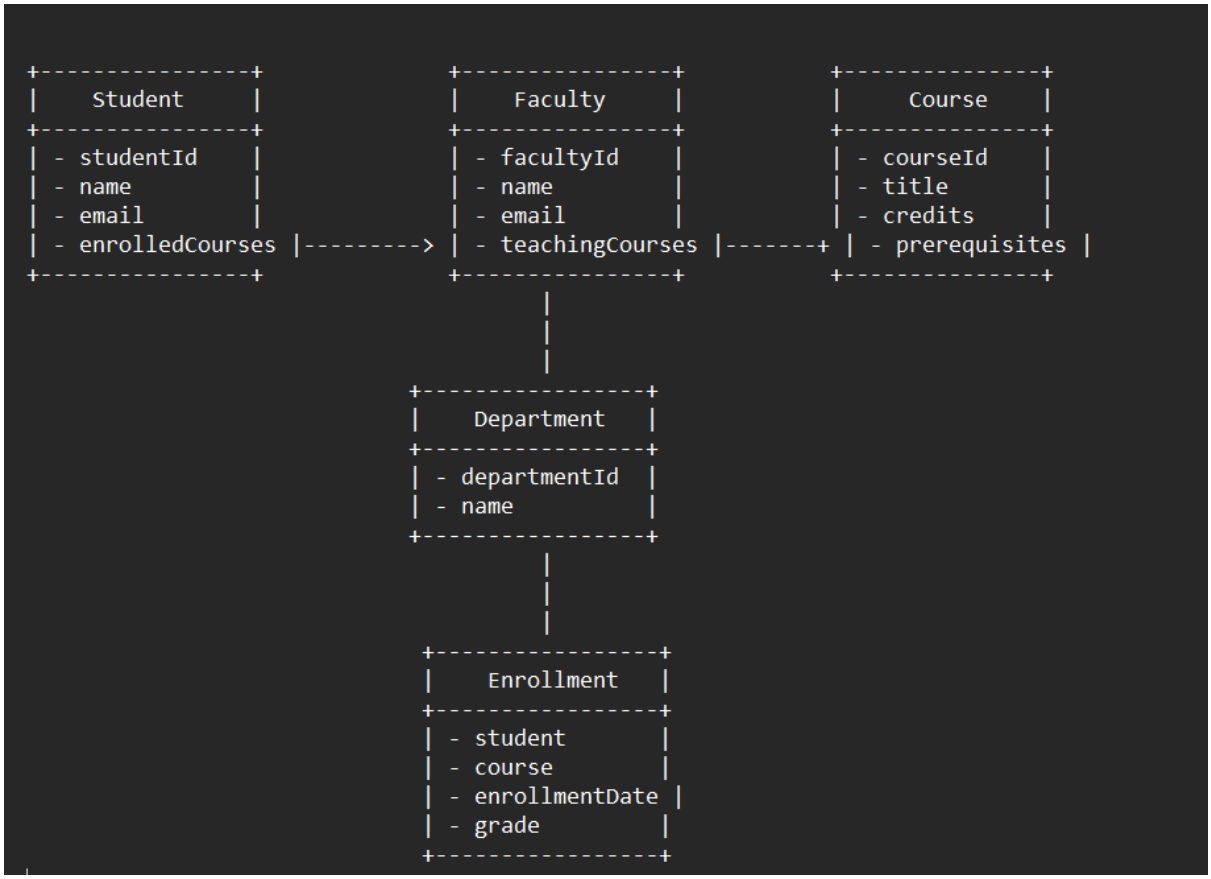
Reliability:

The system should be highly available and reliable, minimizing downtime and data loss.

Regular backups should be performed to prevent data loss in case of system failures.

Please note that designing the complete use cases, CRC cards, class diagrams, and Java code for the university system would require more extensive time and effort. You can use the provided requirements as a starting point to design the remaining components of the system.

3) Class Diagram :-



In this sample class diagram:

The Student class represents a student in the university system and has attributes such as studentId, name, email, and enrolledCourses. The enrolledCourses attribute indicates the courses in which the student is currently enrolled.

The Faculty class represents a faculty member and has attributes such as facultyId, name, email, and teachingCourses. The teachingCourses attribute represents the courses taught by the faculty member.

The Course class represents a course offered by the university and has attributes such as courseId, title, credits, and prerequisites. The prerequisites attribute indicates the courses that need to be completed before taking a particular course.

The Department class represents a department within the university and has attributes such as departmentId and name.

The Enrollment class represents the enrollment of a student in a course and has attributes such as student, course, enrollmentDate, and grade.

4) Use cases :

Enroll in a Course:

Description: This use case allows students to enroll in a course for the upcoming semester.

Actors: Student

Main Flow:

Student selects the desired course from the available course catalog.

System verifies if the student meets the prerequisites for the selected course.

System adds the student to the enrolled students list for the course.

View Course Schedule:

Description: This use case allows students to view their course schedule for the current semester.

Actors: Student

Main Flow:

Student requests to view their course schedule.

System retrieves and displays the schedule of courses the student is enrolled in.

Assign Grades:

Description: This use case allows faculty members to assign grades to students for completed coursework.

Actors: Faculty

Main Flow:

Faculty selects the course for which grades need to be assigned.

System displays the list of enrolled students in the selected course.

Faculty enters the grades for each student.

System saves the grades and updates the student's academic record.

Generate Transcript:

Description: This use case allows students to generate an official transcript of their academic records.

Actors: Student

Main Flow:

Student requests to generate their transcript.

System retrieves the student's academic records, including completed courses and grades.

System generates a transcript document and presents it to the student for download or printing.

Add Course:

Description: This use case allows administrators to add a new course to the university's course catalog.

Actors: Administrator

Main Flow:

Administrator provides the necessary details of the new course, such as title, description, credits, and prerequisites.

System validates the information provided.

System adds the new course to the course catalog.

Generate Faculty Schedule:

Description: This use case allows administrators to generate the teaching schedule for faculty members.

Actors: Administrator

Main Flow:

Administrator selects the semester and faculty members for schedule generation.

System generates a schedule that assigns faculty members to their respective courses and classes.

System presents the generated schedule to the administrator for review and distribution.

5) Demo Classes :-

```
// Student class
public class Student {
    private int studentId;
    private String name;
    private String email;
    private List<Course> enrolledCourses;

    // Constructor, getters, and setters
}

// Faculty class
public class Faculty {
    private int facultyId;
    private String name;
    private String email;
    private List<Course> teachingCourses;

    // Constructor, getters, and setters
}

// Course class
public class Course {
    private int courseId;
    private String title;
```

```
    private int credits;
    private List<Course> prerequisites;
    private Faculty instructor;
    private List<Student> enrolledStudents;

    // Constructor, getters, and setters
}

// Department class
public class Department {
    private int departmentId;
    private String name;

    // Constructor, getters, and setters
}

// Enrollment class
public class Enrollment {
    private Student student;
    private Course course;
    private Date enrollmentDate;
    private String grade;

    // Constructor, getters, and setters
}
```