University of the Punjab

Gujranwala Campus

Second Deliverable

Project ID

TABLE OF CONTENTS

1 Introduction

1.1 Usecase Description

Use Case: UC_Student_Registeration
Use Case: UC_Student_Login
Use Case: UC_Enroll_in_Course
Use Case: UC_View_Course_Content
Use Case: UC_Submit_Assignment
Use Case: UC_Take_Quiz
Use Case: UC_Instructor_Login
Use Case: UC_Create_Course
Use Case: UC_Add_Assignment/Quiz
Use Case: UC_Grade_Submission
Use Case: UC_View_Student_Performance
Use Case: UC_Post_Announcement
Use Case: UC_Join_Discussion
Use Case: UC_Logout
Use Case: UC_Admin_Login
Use Case: UC_Manage_Users
Use Case: UC_Manage_Courses
Use Case: UC_View_System_Reports
Use Case: UC_Moderate_Discussions
Use Case: UC_Logout

1.2 Usecase Diagram (refined and updated)

1.3 Domain Model

The domain model for the Learning Management System (LMS) captures the key conceptual entities and their relationships that form the foundation of the LMS environment. It is built from the system's requirements and real-world elements as outlined in the previous deliverable. This model focuses on the *core business objects*, their *attributes*, and how they interact to support core use cases such as user registration, course management, assessments, progress tracking, and reporting.

1.3.1 Scope of the Domain

The LMS domain covers educational interactions between **students**, **instructors**, and **administrators** with a digital system that provides services such as course management, assessments, discussions, analytics, and reporting. Key components of this domain include user management, course creation, enrollment, assignments, quizzes, and performance tracking.

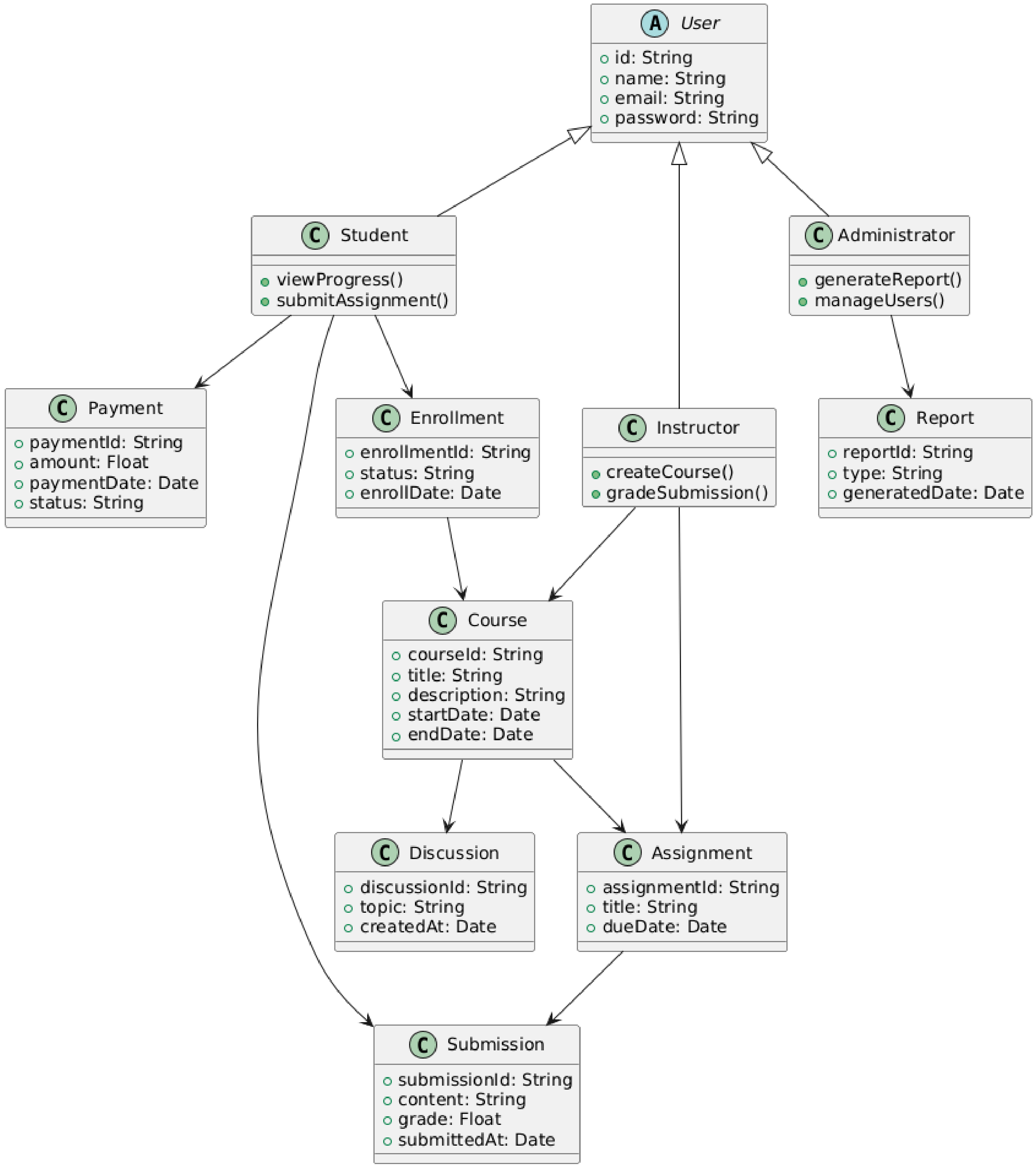
1.3.2 Identified Domain Classes (Conceptual Classes)

Class Name	Description
User	Abstract parent class representing all users of the system.
Student	Inherits from User, can enroll in courses, take quizzes, and view progress.
Instructor	Inherits from User, can create courses, manage content, and grade students.
Administrator	Inherits from User, manages users, reports, and system configurations.
Course	Represents a course in LMS, created by an instructor.
Enrollment	Represents a student's enrollment in a course.
Assignment	Task or quiz created within a course.
Submission	Work submitted by a student for an assignment.
Discussion	Forum threads related to courses.
Report	Analytics and performance insights generated for admin/instructors.
Payment	Tracks course payments made by students (if applicable).

1.3.3 Key Associations

- A Student *enrolls in* zero or more Courses through Enrollment.
- A Course *has many* Assignments.
- A Student *submits* one or more Submissions for Assignments.
- A Course *contains* multiple Discussion threads.
- An Instructor *creates* Courses and *evaluates* Submissions.
- An Administrator *generates* and *views* Reports.
- A Student *makes* Payments for paid Courses.

EduMinds: Domain Model



1.4 Sequence Diagram Overview

In the context of a **Learning Management System (LMS)**, a sequence diagram can be used to depict the communication between various actors (e.g., Student, Instructor, Admin) and the system's objects (e.g., Course, Content, Assignment).

1.4.1 Defining the Sequence Diagram

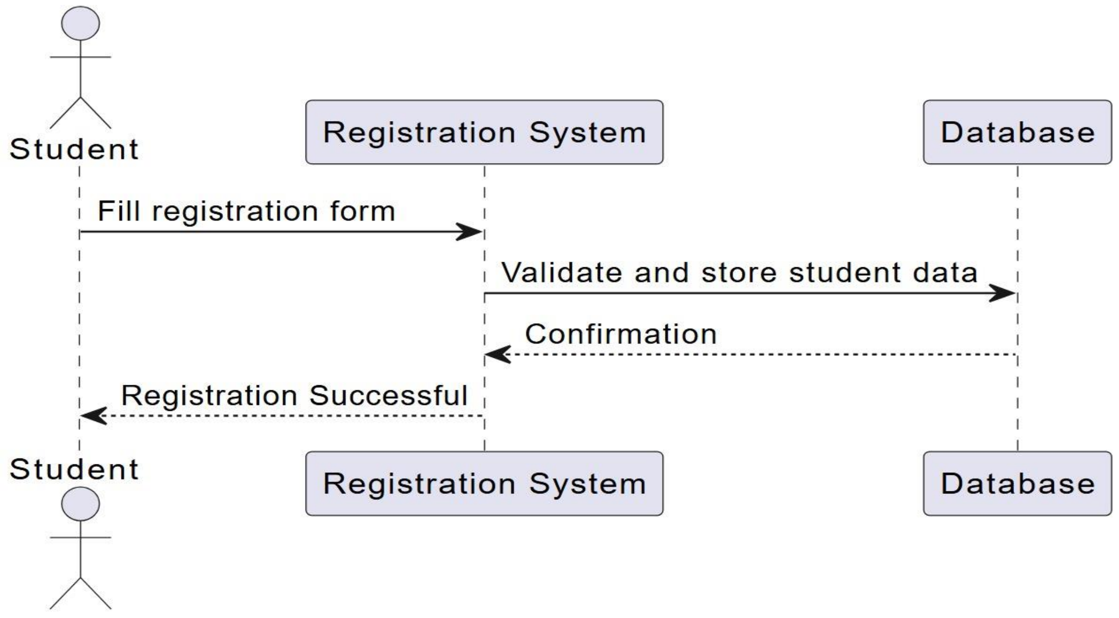
The sequence diagram for the LMS will represent various interactions such as a **Student** logging in, enrolling in a course, submitting assignments, and receiving feedback from the instructor.

1.4.2 Basic Sequence Diagram Symbols and Notations

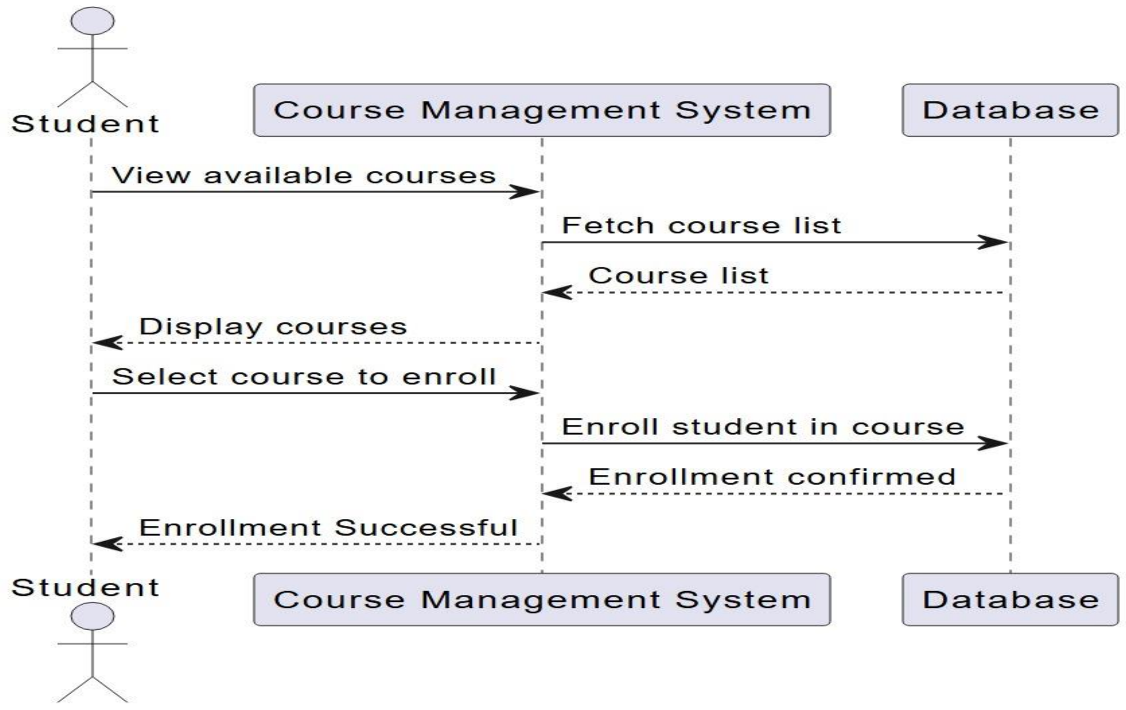
- **Objects**: Represented by rectangles with the underlined class name. For example, `Student : User`, `Course : LMS`
- **Lifelines**: Represented by dashed vertical lines indicating the existence of an object over time.
- **Activation Boxes**: Horizontal rectangles indicating the period an object is active.
- **Messages**: Arrows depicting communication between objects, can be synchronous (solid arrow) or asynchronous (half arrow).
- **Loops**: Represent repetitive actions or conditions. E.g., the process of taking multiple quizzes in a course.

---

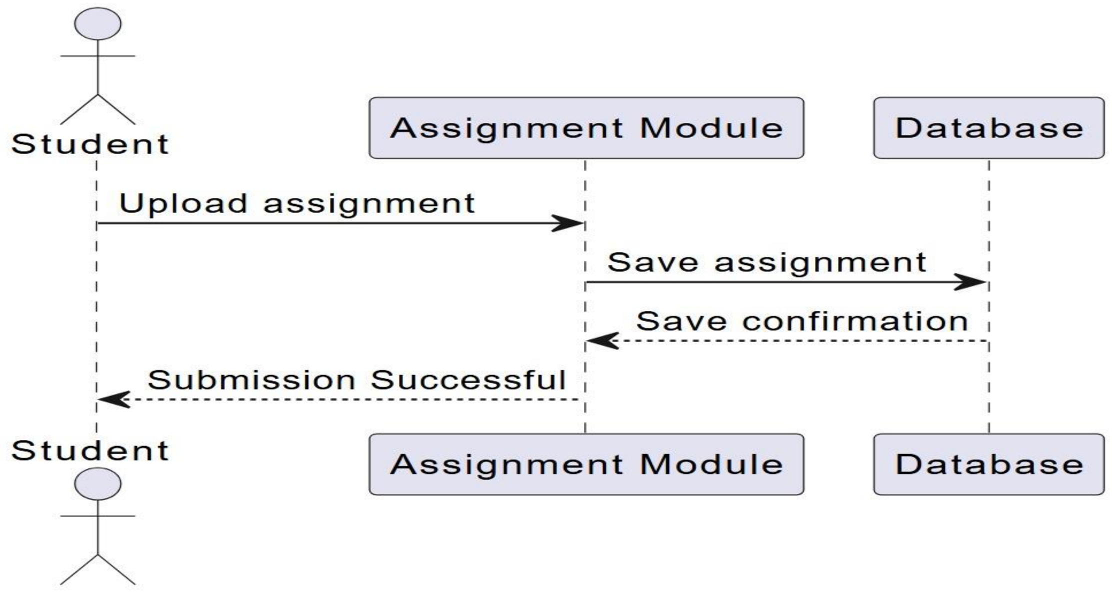
1. Sequence Diagram: Student Registration



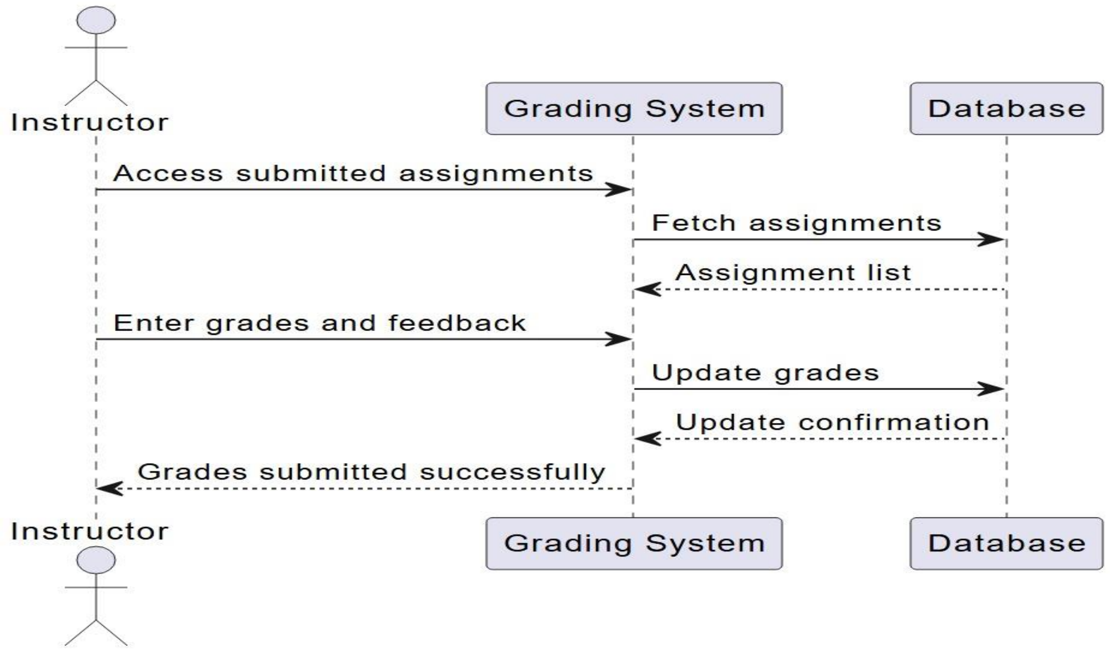
2. Sequence Diagram: Course Enrollment



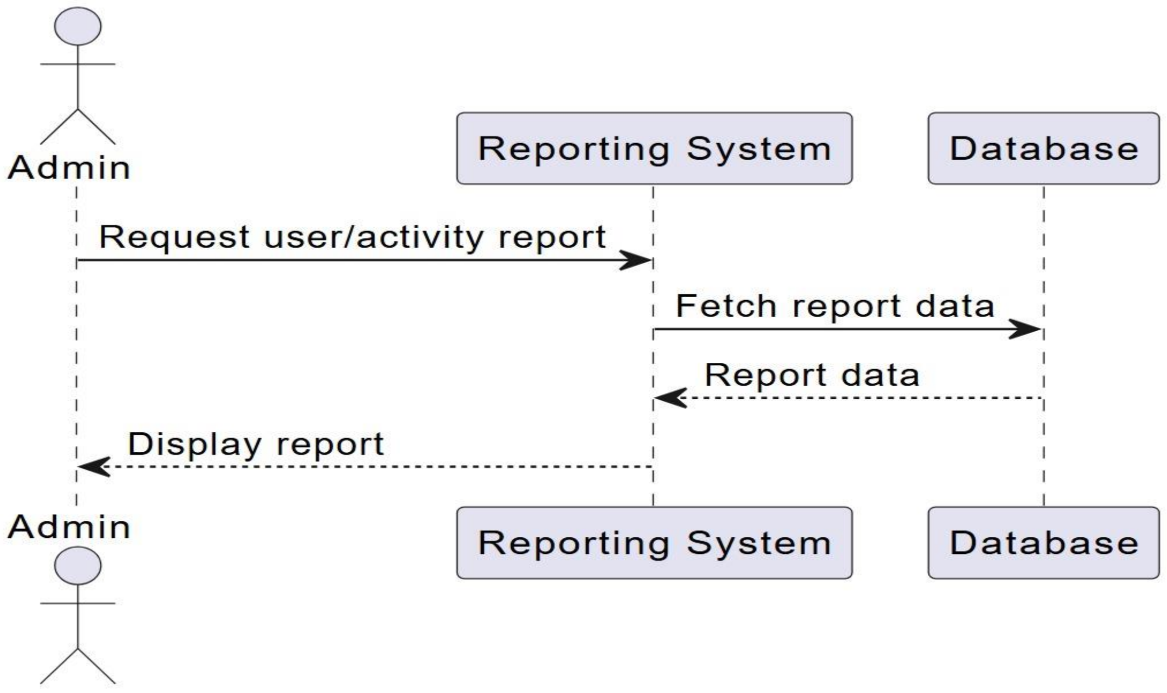
3. Sequence Diagram: Assignment Submission



4. Sequence Diagram: Instructor Grading an Assignment



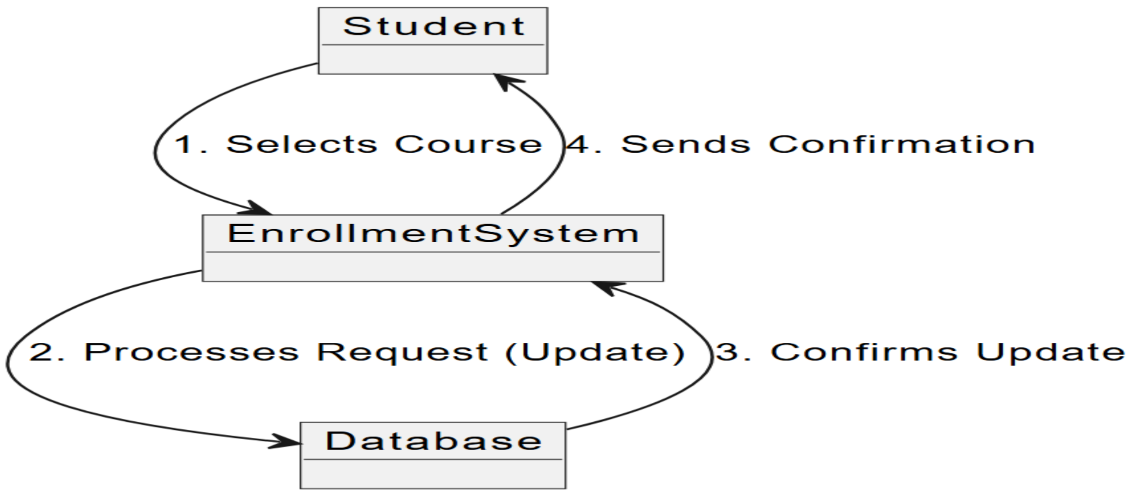
5. Sequence Diagram: Admin Viewing Reports



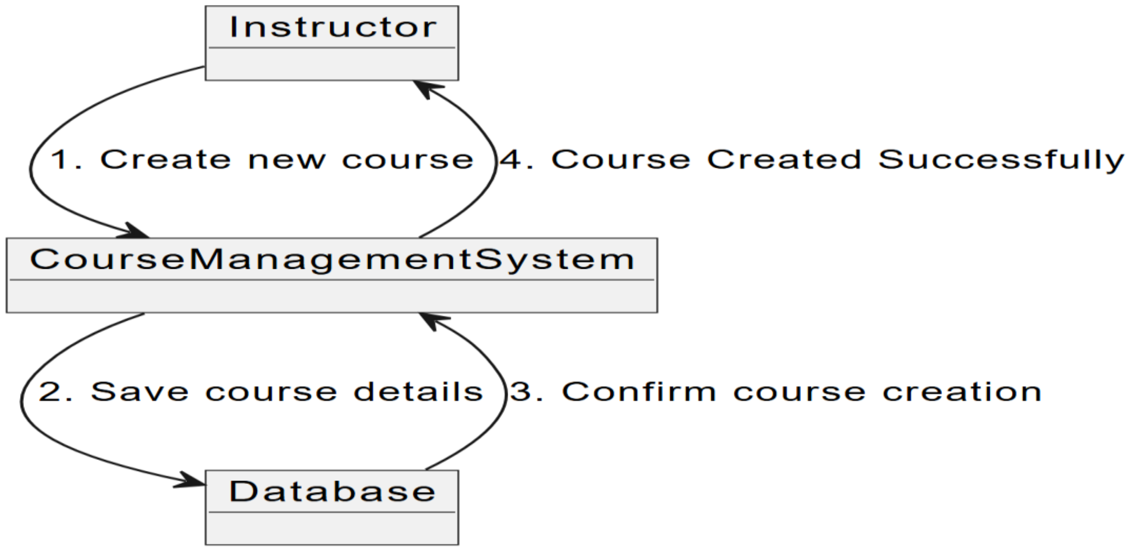
1.5 Collaboration Diagram

A **collaboration diagram** describes how various objects within the LMS system interact to complete the *Enroll in Course* use case. This use case involves multiple participants such as the student, course catalog, enrollment handler, payment gateway (for paid courses), and the course database.

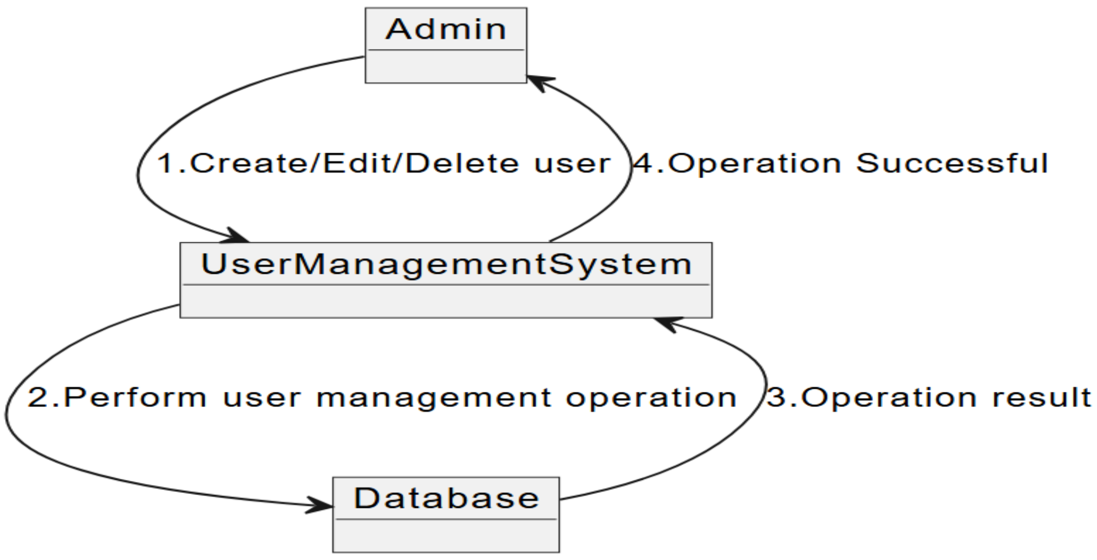
1.5.1 Collaboration Use Case: Enroll in Course



1.5.2 Collaboration Use Case: Instructor Creates a Course



1.5.3 Collaboration Use Case: Admin Manages a User



1.6 Operation Contracts

Operation Contract 1: Enroll in Cours

Responsibilities:
To register a student into a selected course (free or paid), update the course enrollment records, and associate the student with the course.

Cross References:

- Use Case:
- System Function: Course Enrollment
- Domain Model Classes: Student, Course, Enrollment, Payment

Exceptions:

- Course is full (max enrollment limit reached).
- Payment failed (for paid course).
- Invalid course ID or student not registered.

Preconditions:

- A `Student` object with a valid `studentId` exists in the system.
- A `Course` object with the provided `courseId` exists.
- Student is authenticated and authorized to enroll.
- If the course is paid, a valid payment method must be available.

Postconditions:

- A new `Enrollment` instance was created.
- The student was associated with the selected course.
- Course's enrollment count was incremented.
- Payment record was created and associated with the student (if course was paid).
- The enrollment record was stored in the database.
- The student gained access to course materials.

Operation Contract 2: Submit Assignment

Name:
submitAssignment(assignmentId, studentId, submissionData)

Responsibilities:
Allow students to submit assignment content for an enrolled course.

Cross References:

- Use Case: UC_Submit_Assignments
- System Function: Assignment Submission
- Domain Classes: Student, Assignment, Submission0

Exceptions:

- Submission deadline has passed.
- Student not enrolled in the course.
- Invalid assignment ID.

Preconditions:

- Student is enrolled in a course that includes the assignment.
- The assignment exists and is available for submission.
- Submission deadline has not passed.

Postconditions:

- A Submission instance was created and linked to the student and assignment.

- Submission content was stored.
- Timestamp of submission was recorded.
- Submission record was saved in the database.
- Instructor was notified about the new submission.

1.7 Design Class Diagram

Designing a Learning Management System (LMS) involves several steps in class diagram creation to model the system's behavior and structure effectively. Here's a high-level breakdown of how the process could look like, based on your detailed explanation, along with code for diagram construction:

1. Initial Design Classes

- **Domain Model**: We start by identifying key domain classes for the LMS. These could include classes like Course, Student, Instructor, Assignment, Quiz, Grade, and Lecture. Each of these classes represents a core unit of functionality in the system.
- **Trace Dependencies**: Map out dependencies between the domain classes. For example:
  - Course depends on Instructor and Student.
  - Assignment depends on Course and Student.

2. Boundary Classes

Boundary classes are used for interaction with the user interface. For an LMS, boundary classes could represent different sections of the system:

- CoursePage
- AssignmentPage
- StudentDashboard
- InstructorDashboard

These boundary classes will serve as the entry points for the system's operations, triggering behavior through control and entity classes.

3. Entity Classes

Entity classes represent objects that carry data and persist across sessions. For the LMS, entity classes would include:

- Course
- Student
- Assignment
- Quiz
- Grade

These entities encapsulate the main data and business logic for the LMS.

4. Control Classes

Control classes manage the flow of use cases. They encapsulate the logic required for orchestrating actions that aren't related to user interface (boundary) or data persistence (entity). Example control classes could be:

- CourseController
- GradeController
- AssignmentController

5. Persistent Classes

Classes that need to be persistent (store data in a database) are key in the LMS design. These would include:

- Student (stores user data)
- Course (stores course information)
- Assignment (stores details about assignments)
- Grade (stores grades)

6. Define Operations

Operations are the key actions that each class will perform. For example:

- Student:
    - enrollInCourse()
    - submitAssignment()
- Instructor:
    - createCourse()
    - gradeAssignment()

7.Class Visibility

Class visibility defines who can access or modify a class's operations and attributes:

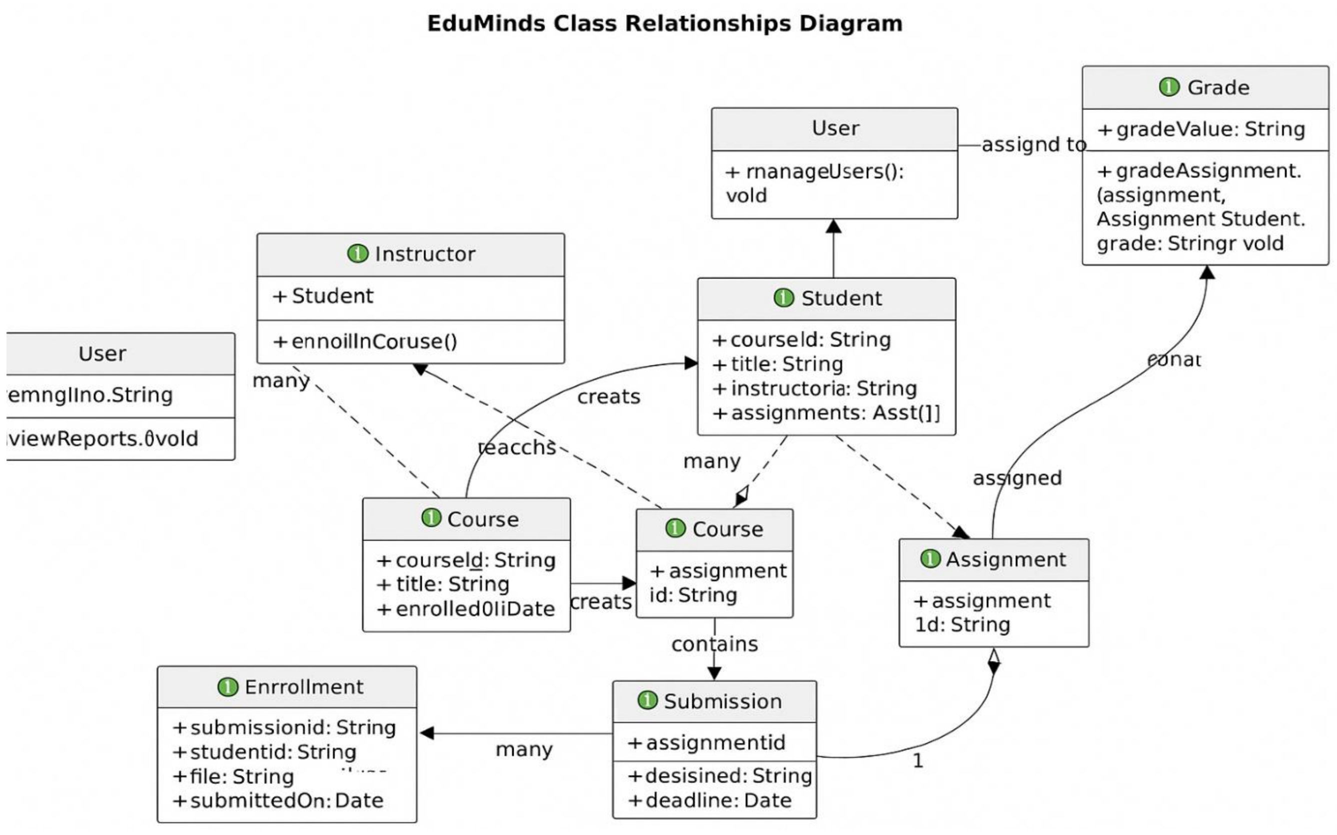
- **Public**: Accessible outside the class, like getStudentDetails()
- **Private**: Accessible only within the class itself
- **Protected**: Accessible by the class and its subclasses

8. Design Class Relationships

- **Composition**: For example, a Course contains multiple Assignments, and each Assignment cannot exist without a Course.
- **Inheritance**: An Instructor class might inherit from a User class, which could also be extended by a Student class.

- **Aggregation**: Course has many Students but a Student can exist without a Course.
- **Association**: Student and Course can be associated without having a whole/part relationship.

Class Diagram



1.8 Data Model

The LMS data model is a logical relational schema that describes how persistent data (like students, instructors, courses, enrollments, assignments, and submissions) are structured and interrelated. It is derived from the domain model and supports the backend database implementation.

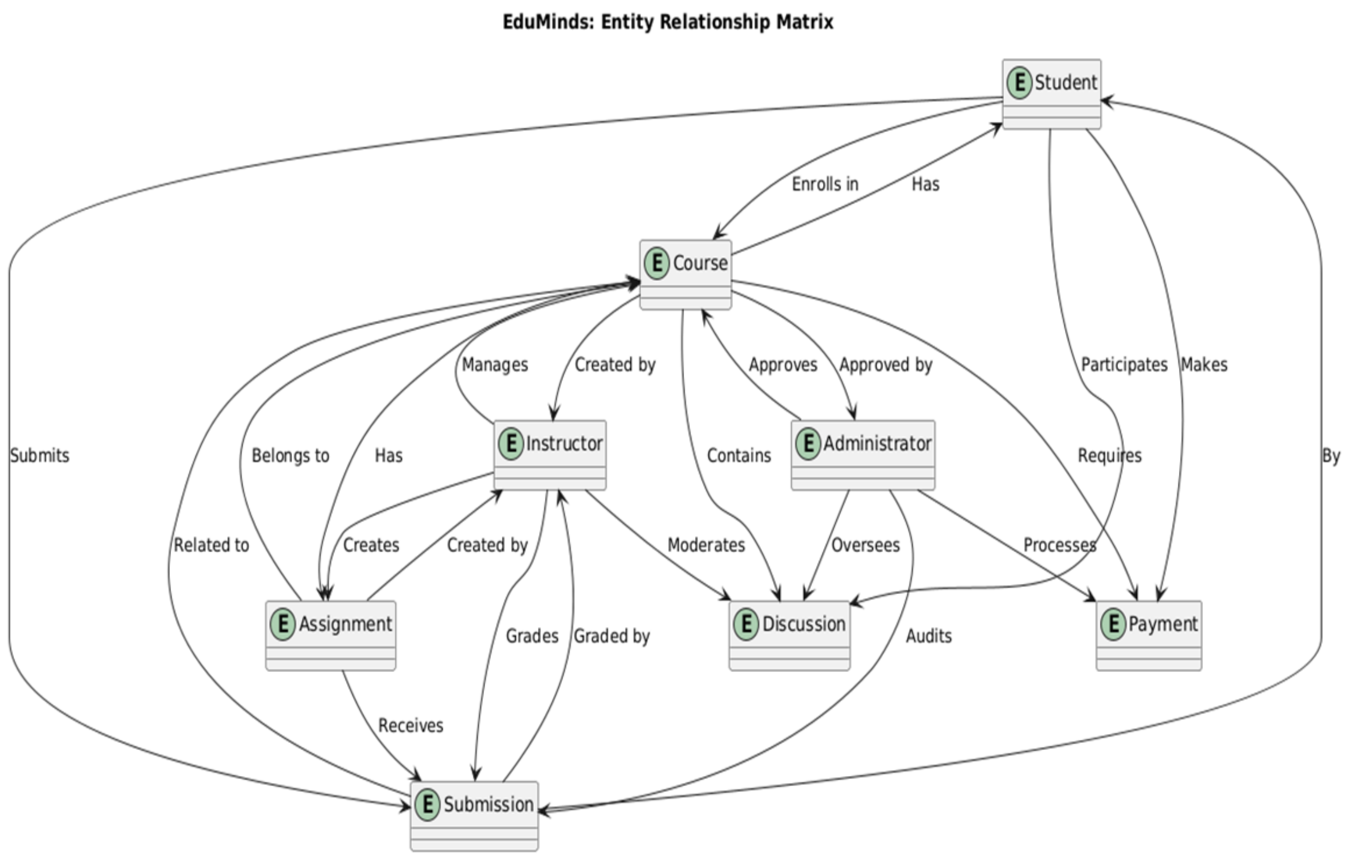
This model uses **primary keys**, **foreign keys**, **entity relationships**, and **cardinality** to represent the system's data behavior in a relational database system such as MongoDB (in practice with schema modeling) or SQL-based systems for this representation.

Identify Entities

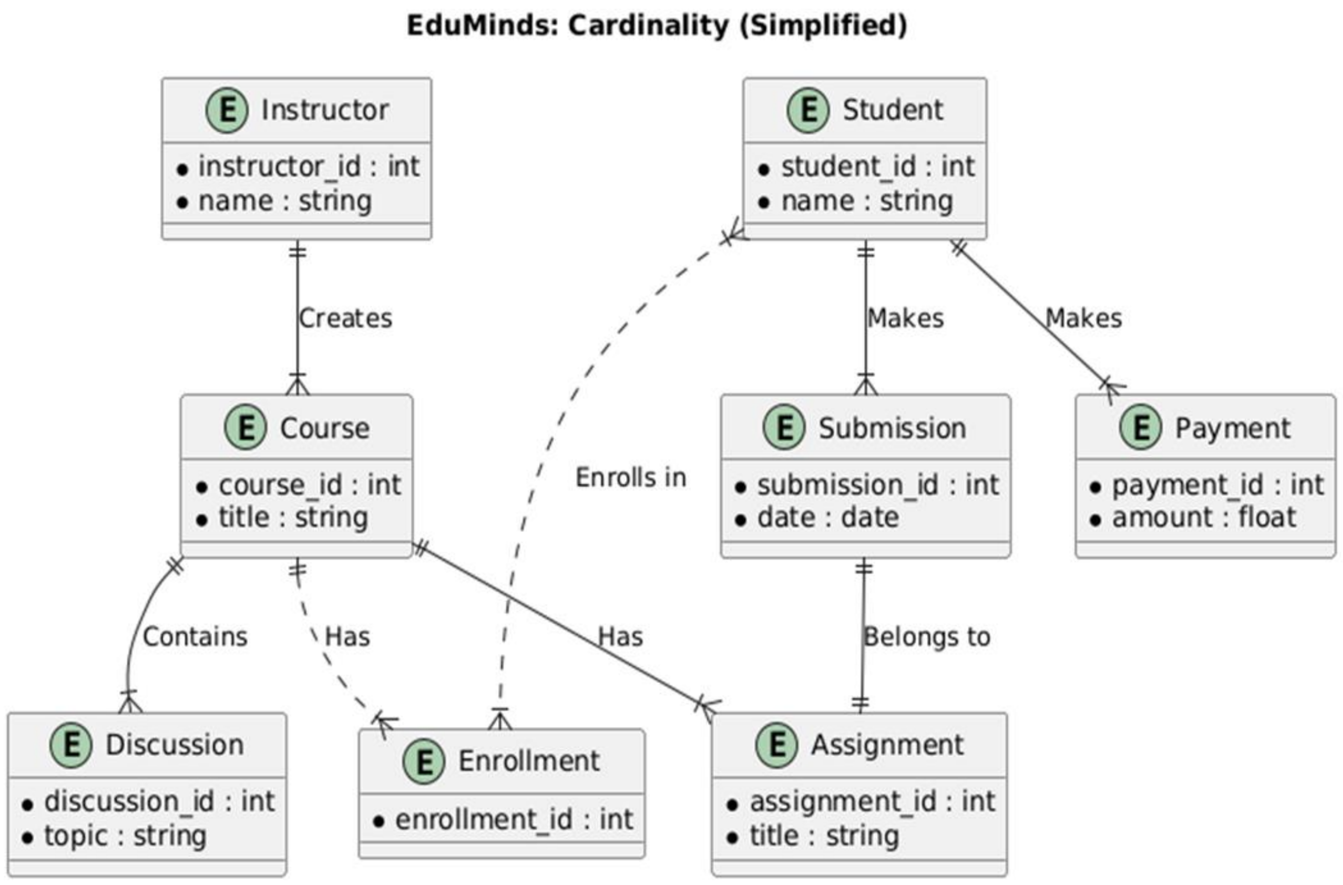
Entities for LMS include:

- Student
- Instructor
- Administrator
- Course
- Enrollment
- Assignment
- Submission
- Discussion
- Payment

Entity Relationship Matrix

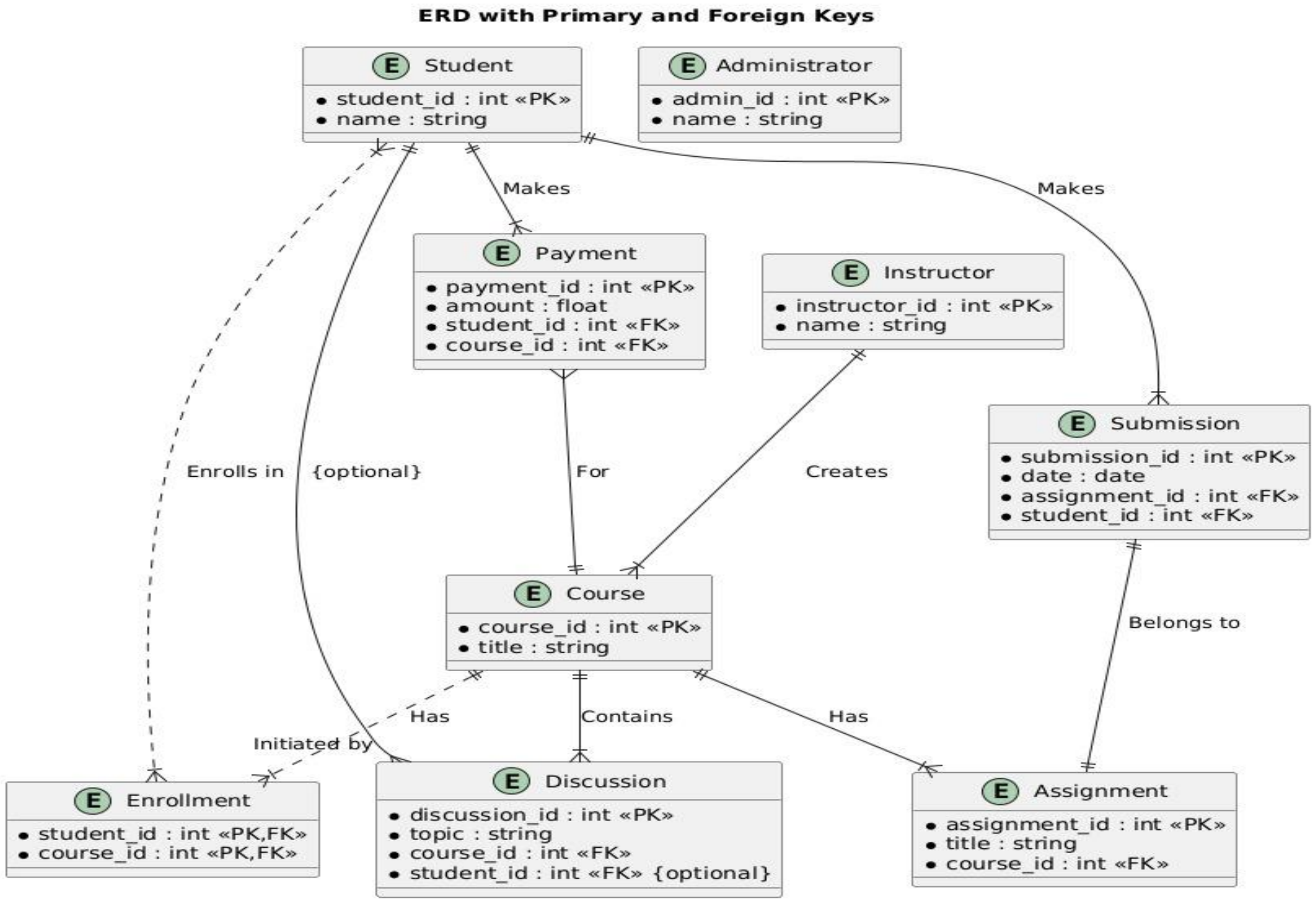


Cardinality (Simplified)



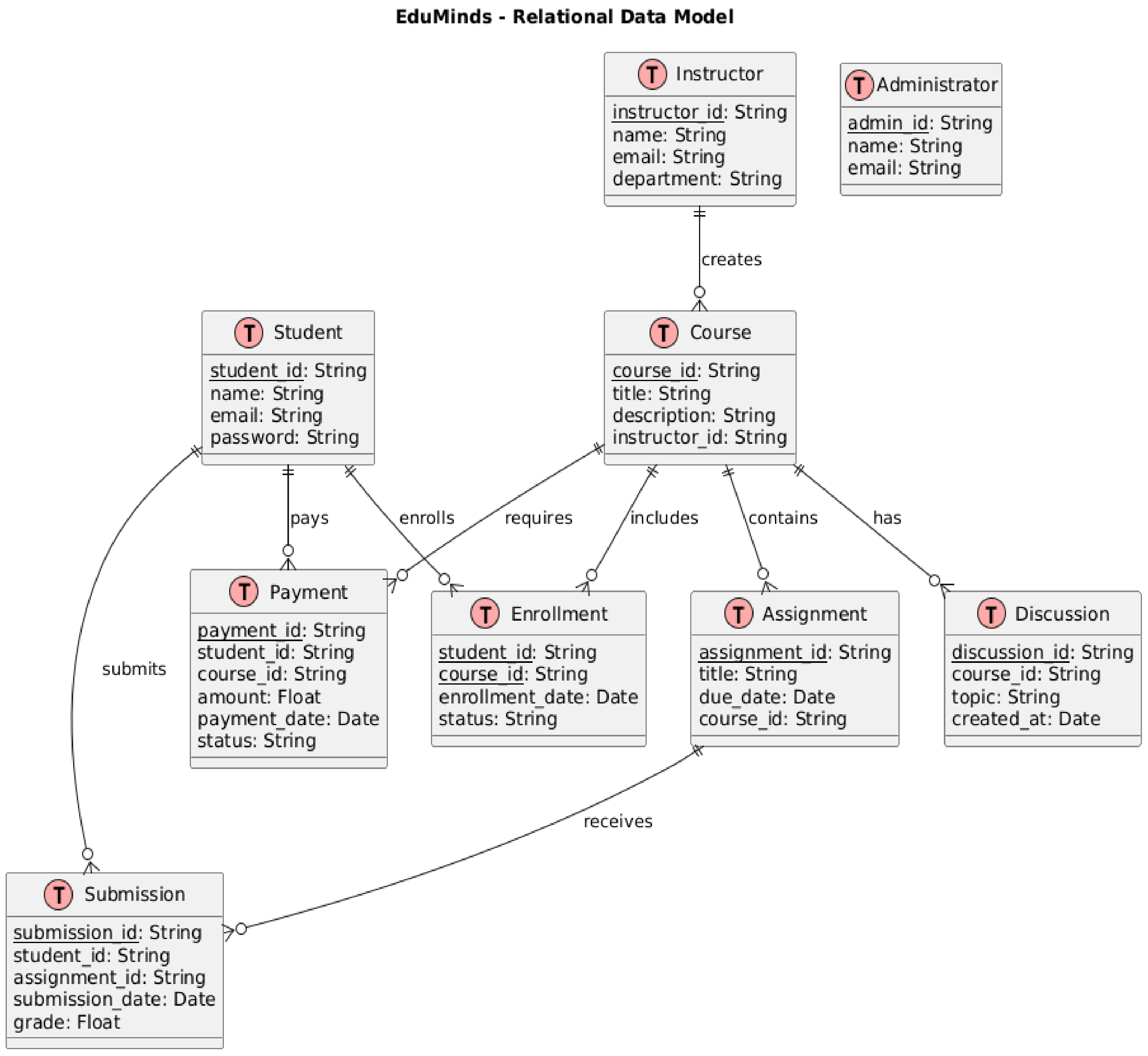
- One **Student** can enroll in many **Courses** (M:N with associative entity `Enrollment`)
- One **Instructor** can create many **Courses** (1:M)
- One **Course** has many **Assignments** (1:M)
- One **Student** can make many **Submissions** (1:M)
- One **Submission** belongs to one **Assignment**
- One **Course** can have multiple **Discussions**
- One **Student** can make multiple **Payments**

Define Primary Keys And foreign Keys:



Entity	**Primary Key**	**Foreign key**
Student	student_id	
Instructor	instructor_id	
Administrator	admin_id	
Course	course_id	
Enrollment	student_id,course_id (composite)	(student_id, course_id)
Assignment	assignment_id	(course_id)
Submission	submission_id	(assignment_id, student_id)
Discussion	discussion_id	(course_id, student_id)
Payment	payment_id	(student_id, course_id)

Fully Attributed ERD



EduMinds - Relational Data Model
Instructor
instructor_id: String
name: String
email: String
department: String
Administrator
admin_id: String
name: String
email: String
creates
Student
student_id: String
name: String
email: String
password: String
Course
course_id: String
title: String
description: String
instructor_id: String
pays
enrolls
requires
includes
contains
has
submits
Payment
payment_id: String
student_id: String
course_id: String
amount: Float
payment_date: Date
status: String
Enrollment
student_id: String
course_id: String
enrollment_date: Date
status: String
Assignment
assignment_id: String
title: String
due_date: Date
course_id: String
Discussion
discussion_id: String
course_id: String
topic: String
created_at: Date
receives
Submission
submission_id: String
student_id: String
assignment_id: String
submission_date: Date
grade: Float