University of the Punjab
Gujranwala Campus

Second Deliverable

Project ID BIT-2123

© Department of Information Technology, University Of The Punjab, Gujranwala Campus.

TABLE OF CONTENTS

1 Introduction…….3

1.1 Usecase Description……..3

1.2 Usecase Diagram (refined and updated)……...17

1.3 Domain Model…….…….....20

1.4 Sequence Diagram Overview…….…….23

1.5 Collaboration Diagram……..…....33

1.6 Operation Contracts……...……..35

1.7 Design Class Diagram……...37

1.8 Data Model……...…….39

1 Introduction

The third deliverable focuses on **Use Case Modeling and Software Design** of the EduMids, following the comprehensive system analysis completed in the previous phase. With a clear understanding of the problem domain and user requirements, this deliverable transitions into designing a solution using **object-oriented techniques**. Our goal is to map real-world requirements into structured system components using **Unified Modeling Language (UML)** tools. These models not only support the development team in understanding the system behavior but also serve as a blueprint for implementation and communication among stakeholders.
This document includes detailed artifacts such as:

1. Use case description
2. Use case diagram refined
3. Domain Model
4. Sequence Diagram
5. Collaboration Diagram
6. Operation Contracts
7. Design Class Diagram
8. Data Model

Each artifact contributes to visualizing the system's structure, behavior, and data handling in a precise and standardized manner. These models aim to ensure the EduMids platform is scalable, modular, and aligned with modern software design principles

Now we discuss these artifacts one by one as follows:

1.1 Usecase Description

1.1.1 Use Case : UC_Student_Registeration

Brief Description:

This use case describes the process of a student registering into the EduMids The student provides required personal information and credentials to create an account. Upon submission, the system stores the data and grants access to the platform. This use case represents a successful registration process (happy path).

Preconditions:

- The student must have access to the EduMids registration page.
- The system is online and functioning.
- Required fields (e.g., name, email, password) are available and validated for input.

Basic Flow:

1. Student navigates to the EduMids registration page.
2. System displays the registration form.
3. Student fills out required information including name, email, and password.

4. Student submits the registration form.
5. System validates the input fields (format, duplicates, etc.).
6. System stores the student's data in the database.
7. System confirms successful registration and redirects to the login page.

Alternate Flows:

A1: Invalid Input

- 5a. System detects invalid email format or missing required fields.
- 5b. System displays an error message.
- 5c. Student corrects input and resubmits the form.

A2: Email Already Exists

- 5a. System detects that the email address is already registered.
- 5b. System notifies the student and prompts to use a different email or log in.

A3: Server Error

- 6a. System fails to save data due to server/database issue.
- 6b. System displays an error and requests the student to try again later.

Postconditions:

- Student's account is successfully created in the system.
- Student is able to log in and access the EduMids with the registered credentials.
- The system maintains integrity and stores only valid user data.

1.1.2 Use Case : UC_Student_Login

Brief Description:
This use case describes how a student logs into the EduMids using valid credentials. Upon successful authentication, the student is redirected to their dashboard to access learning resources and tools.

Preconditions:

- The student already has a registered account in the EduMids.
- The EduMids login page is accessible and the system is online.

Basic Flow :

1. Student navigates to the EduMids login page.
2. Student enters their registered email and password.

3. Student clicks the **Login** button.
4. System verifies the provided credentials:
5. Matches email and password against stored records.
6. Upon successful authentication, the system redirects the student to their dashboard.

Alternate Flows:

A1: Invalid Credentials

- 4a. System detects incorrect email or password.
- 4b. System displays an error message prompting re-entry.
- 4c. Student re-enters credentials or uses “Forgot Password” option.

A2: Account Locked or Inactive

- 4a. System detects the student’s account is locked or inactive.
- 4b. System informs the student and advises to contact support.

A3: Server Error

- 4a. System encounters a server/database issue during login attempt.
- 4b. System displays an error message and suggests trying again later.

Postconditions:

- The student is successfully logged into the EduMids.
- The student gains access to their personal dashboard and related functionalities.
- A secure session is established for the student.

1.1.3 Use Case : UC_Enroll_in_Course

Brief Description:
This use case describes the process of a student enrolling in a course on the EduMids The student selects a course, and if the course is paid, completes the payment process. The system then stores the enrollment and confirms the registration.

Preconditions:

- The student is logged into the EduMids.
- The course is available for enrollment.
- Payment gateway (for paid courses) is operational.

Basic Flow:

1. Student browses available courses and selects a course.

2. Student clicks the **Enroll** button.
3. If the course is paid:
4. System redirects the student to the payment page.
5. Student completes the payment using available options.
6. System confirms the enrollment.
7. System stores the enrollment record in the database.

Alternate Flows:

A1: Payment Failed (for Paid Courses)

- 3a. Payment is declined or interrupted.
- 3b. System notifies the student of the failure.
- 3c. Student can retry payment or cancel the process.

A2: Course Already Enrolled

- 2a. System detects that the student is already enrolled in the selected course.
- 2b. System notifies the student and prevents duplicate enrollment.

A3: Server Error

- 5a. System encounters an issue while saving enrollment data.
- 5b. System displays an error message and suggests retrying later.

Postconditions:

- An enrollment record is created and stored in the EduMids database.
- The student is granted access to the selected course content.
- Payment (if applicable) is recorded successfully.

1.1.4 Use Case : UC_View_Course_Content

Brief Description:
This use case describes how a student views course content such as videos, notes, and other learning materials after enrolling in a course on the EduMids

Preconditions:

- The student is logged into the EduMids.
- The student is enrolled in the selected course.
- The course content is published and available.

Basic Flow:

1. Student navigates to their dashboard and clicks on an enrolled course.
2. System displays the course content, organized by sections or modules.
3. Student views available materials such as videos, notes, quizzes, and downloadable resources.

Alternate Flows:

A1: Not Enrolled in Course

- 1a. Student clicks on a course they are not enrolled in.
- 1b. System restricts access and displays a message prompting enrollment.

A2: Content Not Available

- 2a. System detects that course materials have not been uploaded or published.
- 2b. System notifies the student that content is currently unavailable.

A3: Server or Media Error

- 3a. System encounters an error loading videos or documents.
- 3b. System displays an error message and suggests trying again later.

Postconditions:

- The student successfully accesses and views the course content.
- System may update activity logs or progress tracking as the content is viewed.

1.1.5 Use Case : UC_Submit_Assignment

Brief Description:
This use case describes how a student submits an assignment through the EduMids by uploading a file and confirming the submission.

Preconditions:

- The student is logged into the EduMids.
- The student is enrolled in the course.
- The assignment is available and submission is open.

Basic Flow:

1. Student navigates to the course and selects the available assignment.
2. System displays the assignment details and upload interface.
3. Student uploads the assignment file.

4. Student clicks the **Submit** button.
5. System stores the submission and confirms successful upload.

Alternate Flows:

A1: No File Selected

- 3a. Student clicks **Submit** without uploading a file.
- 3b. System displays an error prompting the student to upload a file.

A2: Invalid File Format

- 3a. Student uploads a file with an unsupported format.
- 3b. System displays an error message and requests a valid file type.

A3: Server/Storage Error

- 5a. System fails to store the uploaded file due to a backend issue.
- 5b. System displays an error and prompts the student to try again later.

Postconditions:

- The assignment submission is successfully stored in the EduMids.
- The submission is linked to the correct student and assignment.
- The system may notify the instructor of the new submission.

1.1.6 Use Case : UC_Take_Quiz

Brief Description:
This use case describes how a student takes a quiz within the EduMids. The student starts the quiz, answers the questions, submits it, and receives a score.

Preconditions:

- The student is logged into the EduMids.
- The student is enrolled in the course offering the quiz.
- The quiz is published and within its availability time window.

Basic Flow :

1. Student navigates to the course and opens the available quiz.
2. System displays quiz instructions and questions.
3. Student answers the questions within the given time.
4. Student submits the quiz.
5. System stores the quiz attempt and displays the score (if auto-graded).

Alternate Flows:

A1: Quiz Not Available

- 1a. Quiz is not live or has expired.
- 1b. System prevents access and notifies the student.

A2: Incomplete Submission

- 4a. Student attempts to submit before answering all required questions.
- 4b. System prompts the student to complete unanswered questions.

A3: Server Error During Submission

- 4a. System encounters an error during submission.
- 4b. System displays an error and may auto-save progress if supported.

Postconditions:

- The quiz attempt is recorded and linked to the student.
- The score is calculated and shown (if immediate feedback is enabled).
- The instructor can review the attempt and grade if manual evaluation is required.

1.1.7 Use Case : UC_Instructor_Login

Brief Description:
This use case describes how an instructor logs into the EduMids using verified credentials to access their teaching dashboard.

Preconditions:

- Instructor has a registered account.
- The EduMids is accessible and functioning.

Basic Flow:

1. Instructor navigates to the EduMids login page.
2. Enters email and password.
3. Clicks the **Login** button.
4. System validates credentials.
5. Instructor is redirected to the dashboard.

Postconditions:

- Instructor is logged in and can access course management features.

1.1.8 Use Case : UC_Create_Course

Brief Description:
This use case describes how an instructor creates a new course by entering course details and structure.

Preconditions:

- Instructor is authenticated and on the dashboard.

Basic Flow:

1. Instructor clicks **Create Course**.
2. Fills in course title, description, price, thumbnail, and structure (modules, lessons).
3. Clicks **Submit**.
4. System validates and stores the course.

Postconditions:

- A new course is created and available for students (if published).

1.1.9 Use Case : UC_Add_Assignment/Quiz

Brief Description:
This use case describes how an instructor adds assessments (assignments or quizzes) to a course.

Preconditions:

- Instructor is logged in.
- The course exists.

Basic Flow:

1. Instructor opens the course management page.
2. Selects the section to add an assignment or quiz.
3. Enters title, instructions, deadline, and uploads necessary materials.
4. Clicks **Save**.

Postconditions:

- Assessment is available to enrolled students.

1.1.10 Use Case : UC_Grade_Submission

Brief Description:
This use case describes how an instructor reviews submitted student work and provides grades and feedback.

Preconditions:

- Instructor is logged in.
- Submissions are available for grading.

Basic Flow:

1. Instructor opens a submission.
2. Reviews uploaded file or answers.
3. Enters marks and feedback.
4. Clicks **Submit Grade**.

Postconditions:

- Grade and feedback are stored and visible to the student.

1.1.11 Use Case : UC_View_Student_Performance

Brief Description:
This use case describes how an instructor views performance metrics of students enrolled in their courses.

Preconditions:

- Instructor is logged in and has active courses with enrolled students.

Basic Flow:

1. Instructor opens the performance tab of a course.
2. System displays reports such as quiz scores, assignment grades, and activity progress.

Postconditions:

- Instructor has access to insights for evaluating student performance.

1.1.12 Use Case : UC_Post_Announcement

Brief Description:
This use case describes how an instructor posts announcements related to deadlines, schedules, or updates.

Preconditions:

- Instructor is logged in and managing a course.

Basic Flow:

1. Instructor selects the course.
2. Clicks **Post Announcement**.
3. Enters title and message.
4. Clicks **Publish**.

Postconditions:

- Announcement is visible to all students enrolled in the course.

1.1.13 Use Case : UC_Join_Discussion

Brief Description:
This use case describes how an instructor participates in forum discussions or answers student queries.

Preconditions:

- Instructor is logged in.
- Discussion forums are enabled.

Basic Flow:

1. Instructor navigates to course discussion forum.
2. Views and replies to student questions or posts new discussions.

Postconditions:

- Instructor's participation is recorded and visible to students.

1.1.14 Use Case : UC_Logout

Brief Description:
This use case describes how the instructor logs out from the EduMids and ends the current session.

Preconditions:

- Instructor is logged in.

Basic Flow:

1. Instructor clicks the **Logout** button.
2. System ends session and redirects to login page.

Postconditions:

- Instructor session is terminated; system is secure.

1.1.15 Use Case : UC_Admin_Login

Brief Description:
This use case describes how an administrator logs into the EduMids using elevated credentials to access administrative functionalities.

Preconditions:

- Admin account exists with valid credentials.
- The system is accessible.

Basic Flow:

1. Admin navigates to the EduMids login page.
2. Enters admin credentials.
3. Clicks **Login**.
4. System verifies credentials and grants access to the admin dashboard.

Postconditions:

- Admin is logged in and able to manage users, courses, and the system.

1.1.16 Use Case : UC_Manage_Users

Brief Description:
This use case describes how the admin manages user accounts including adding, editing, deleting, blocking, or resetting passwords.

Preconditions:

- Admin is logged in.

Basic Flow:

1. Admin navigates to the user management section.
2. Views the list of users (students/instructors).
3. Performs actions such as add, update, delete, reset password, or block account.
4. System processes and reflects the changes.

Postconditions:

- User data is updated, removed, or modified as requested.

1.1.17 Use Case : UC_Manage_Courses

Brief Description:
This use case describes how the admin can access and manage all courses on the platform.

Preconditions:

- Admin is logged in.

Basic Flow:

1. Admin accesses the course management panel.
2. Views the list of all courses.
3. Selects a course to view, edit, or delete.
4. System performs the requested course management action.

Postconditions:

- The selected course is updated, removed, or remains unchanged based on the admin's action.

1.1.18 Use Case : UC_View_System_Reports

Brief Description:
This use case describes how the admin accesses platform-wide reports for monitoring and analysis.

Preconditions:

- Admin is logged in.

Basic Flow:

1. Admin navigates to the reports section.
2. Selects a report type (e.g., number of users, active courses, revenue, user engagement).
3. System generates and displays the selected report.

Postconditions:

- Admin views the requested data for analysis or decision-making.

1.1.19 Use Case : UC_Moderate_Discussions

Brief Description:
This use case describes how the admin moderates forum discussions to ensure proper conduct.

Preconditions:

- Admin is logged in.
- Discussions or comments exist on the platform.

Basic Flow:

1. Admin accesses the discussion/forum management area.
2. Reviews discussion threads or user comments.
3. Removes inappropriate content or bans users from participation.

Postconditions:

- Inappropriate content is removed and user restrictions are applied as necessary.

1.1.20 Use Case : UC_Logout

Brief Description:
This use case describes how the admin logs out of the system to end the current session.

Preconditions:

- Admin is logged in.

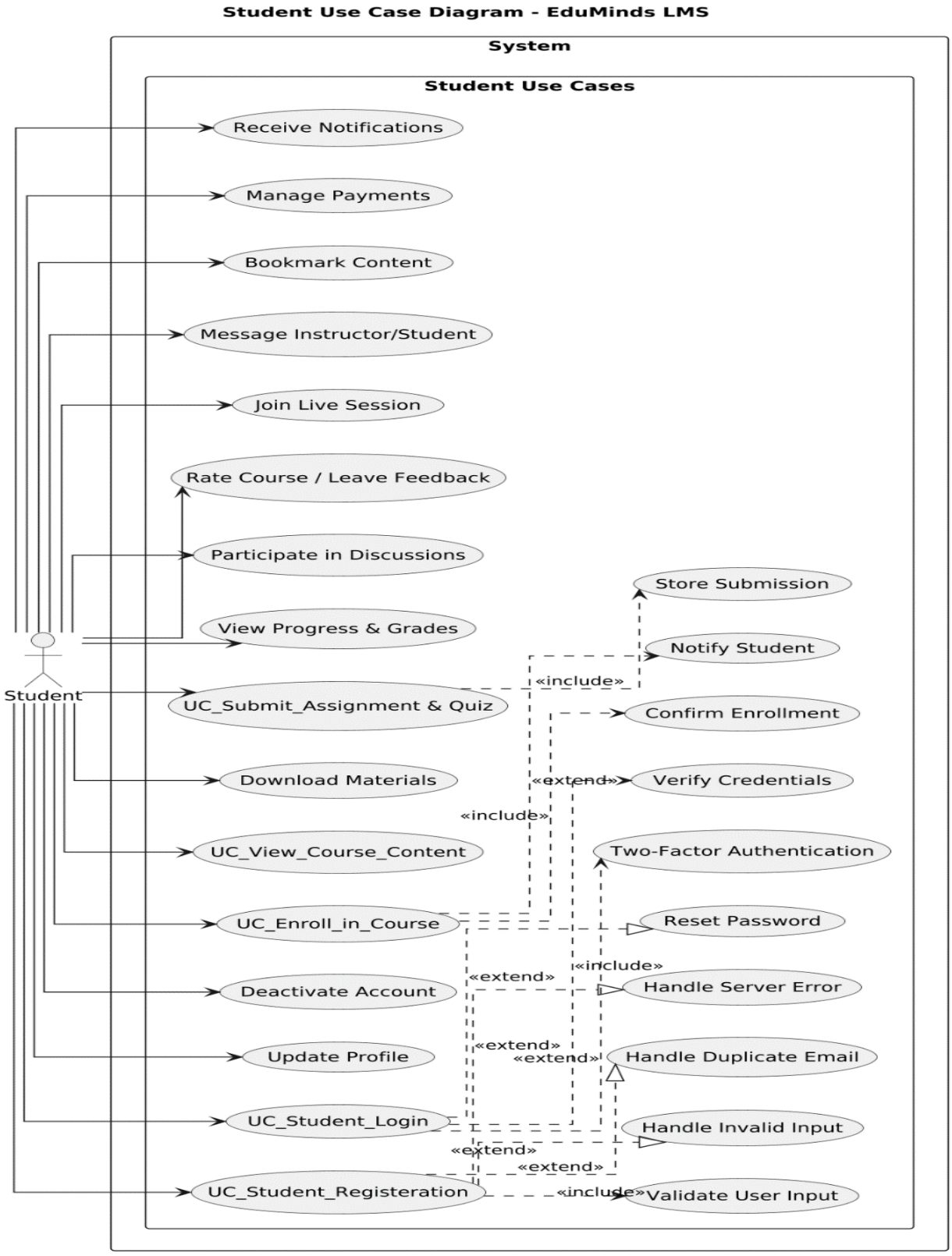
Basic Flow:

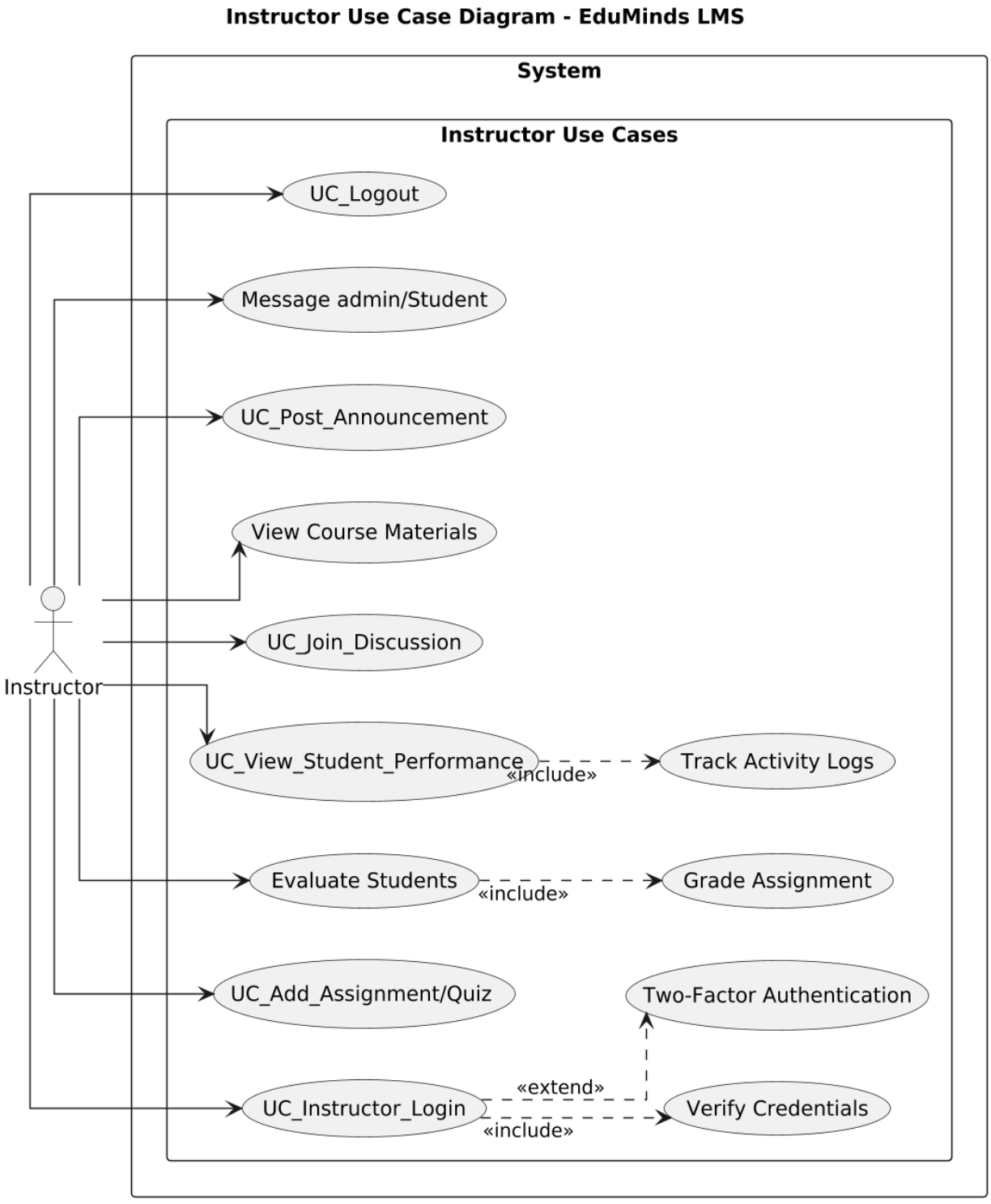
1. Admin clicks the **Logout** button.
2. System ends the session and returns to the login page.

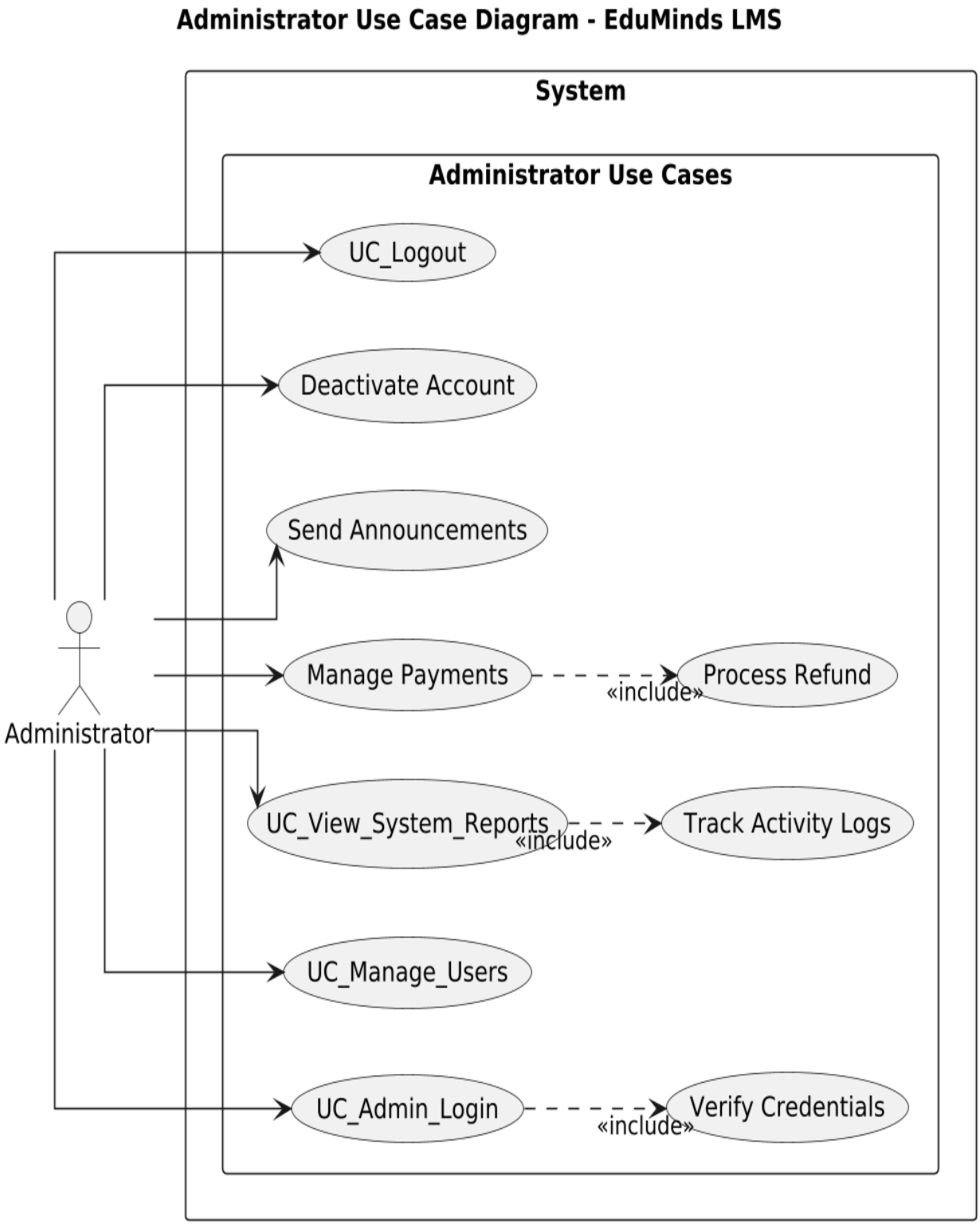
Postconditions:

- Admin session is securely terminated.

1.2 Usecase Diagram (refined and updated)




Instructor Use Case Diagram - EduMinds LMS
System
Instructor Use Cases
UC_Logout
Message admin/Student
UC_Post_Announcement
View Course Materials
UC_Join_Discussion
Instructor
UC_View_Student_Performance
«include»
Track Activity Logs
Evaluate Students
«include»
Grade Assignment
UC_Add_Assignment/Quiz
Two-Factor Authentication
«extend»
UC_Instructor_Login
Verify Credentials
«include»


Administrator Use Case Diagram - EduMinds LMS
System
Administrator Use Cases
UC_Logout
Deactivate Account
Send Announcements
Manage Payments
Process Refund
«include»
Administrator
UC_View_System_Reports
«include»
Track Activity Logs
UC_Manage_Users
UC_Admin_Login
Verify Credentials
«include»

1.3 Domain Model

The domain model for the EduMids-Your Pathway To khowledge captures the key conceptual entities and their relationships that form the foundation of the EduMids 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 EduMids 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.

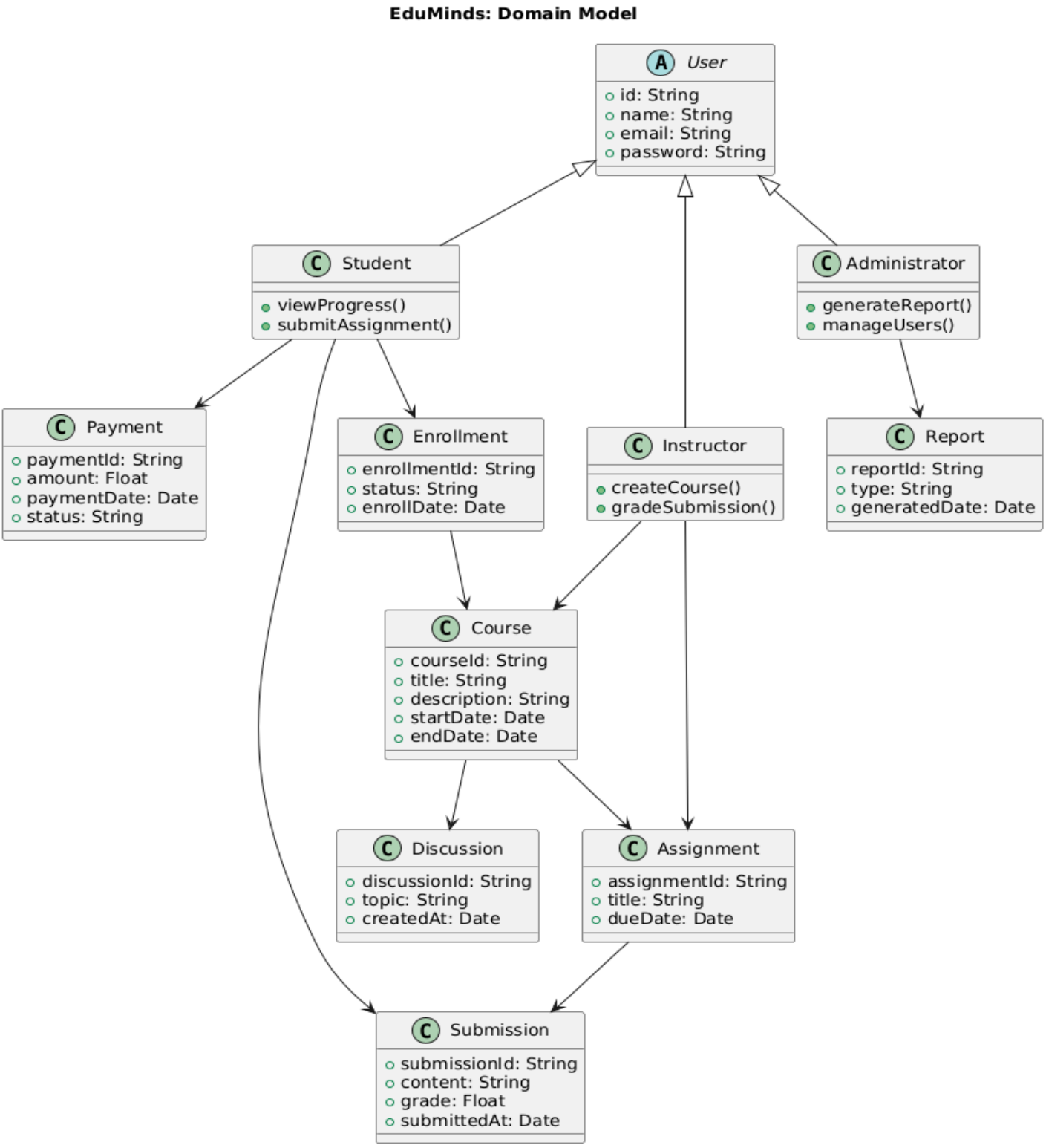
1.3.2 Identified Domain Classes (Conceptual Classes)

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 EduMids, 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.

Class Name	Description
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
User
id: String
name: String
email: String
password: String
Student
viewProgress()
submitAssignment()
Administrator
generateReport()
manageUsers()
Payment
paymentId: String
amount: Float
paymentDate: Date
status: String
Enrollment
enrollmentId: String
status: String
enrollDate: Date
Instructor
createCourse()
gradeSubmission()
Report
reportId: String
type: String
generatedDate: Date
Course
courseId: String
title: String
description: String
startDate: Date
endDate: Date
Discussion
discussionId: String
topic: String
createdAt: Date
Assignment
assignmentId: String
title: String
dueDate: Date
Submission
submissionId: String
content: String
grade: Float
submittedAt: Date

1.4 Sequence Diagram

In the context of a EduMids-Your Pathway To khowledge 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).

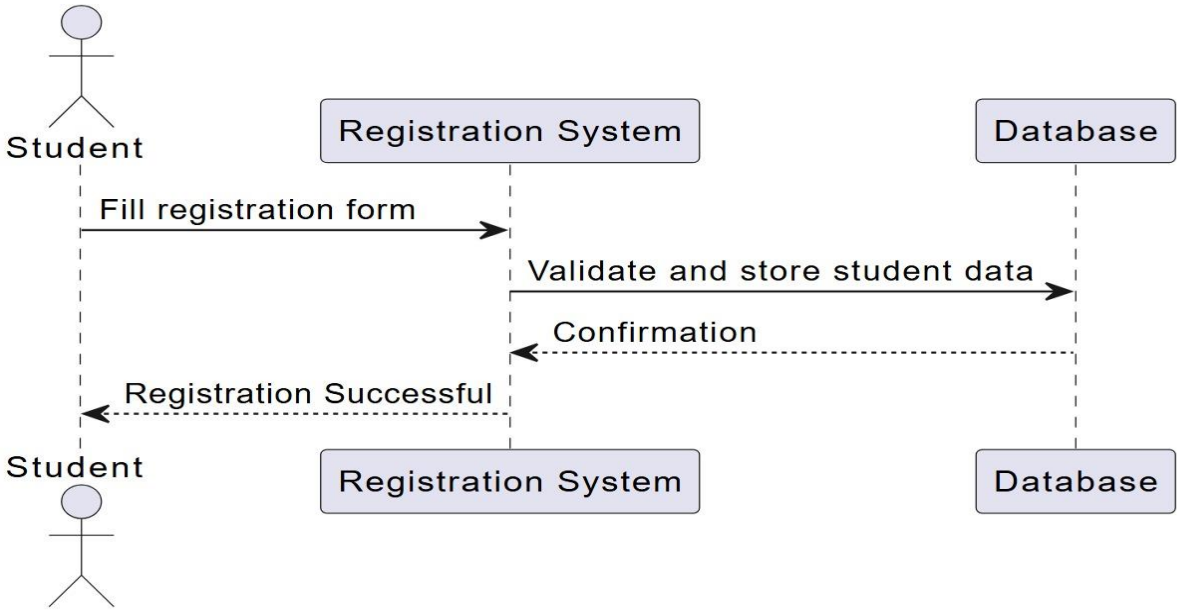
Defining the Sequence Diagram

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

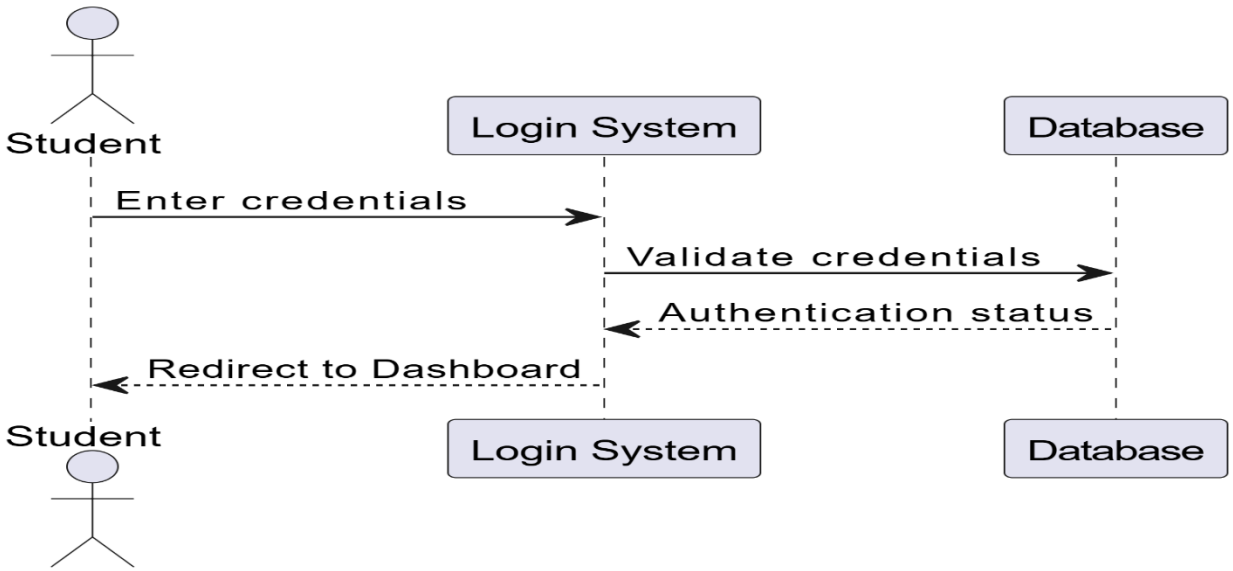
Basic Sequence Diagram Symbols and Notations

- **Objects**: Represented by rectangles with the underlined class name. For example, `Student : User`, `Course :` EduMids
- **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.

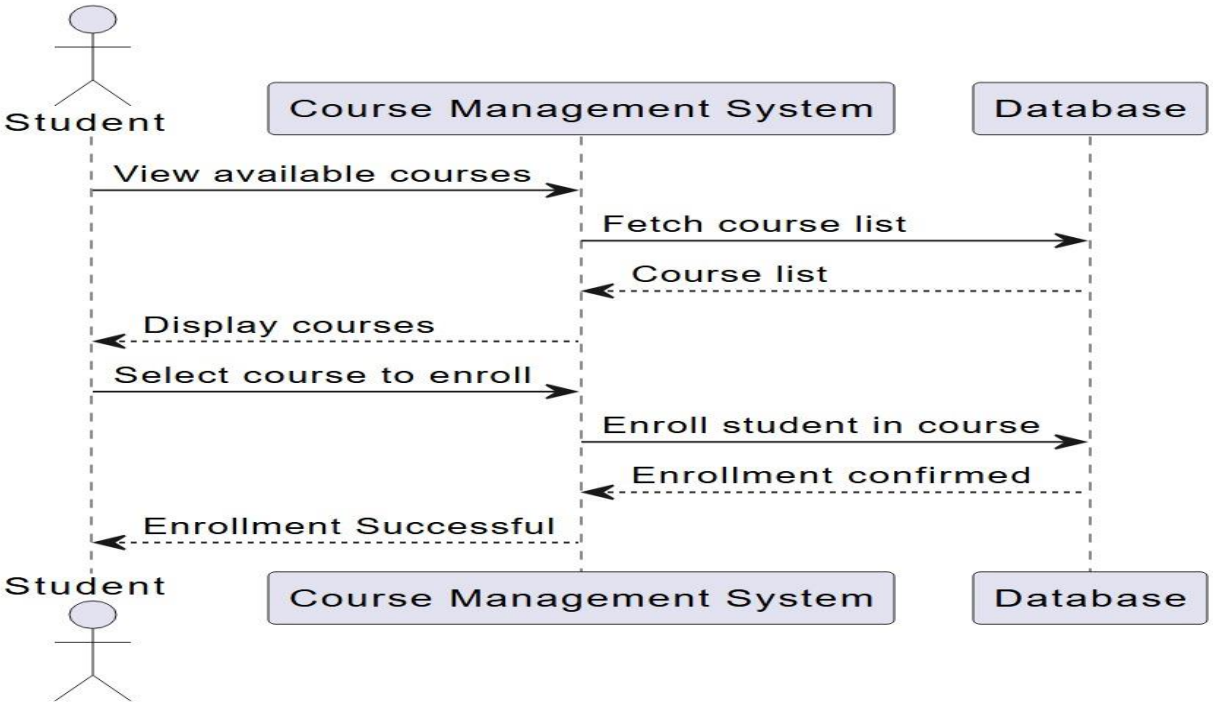
1.4.1 Sequence Diagram: Student_Registeration



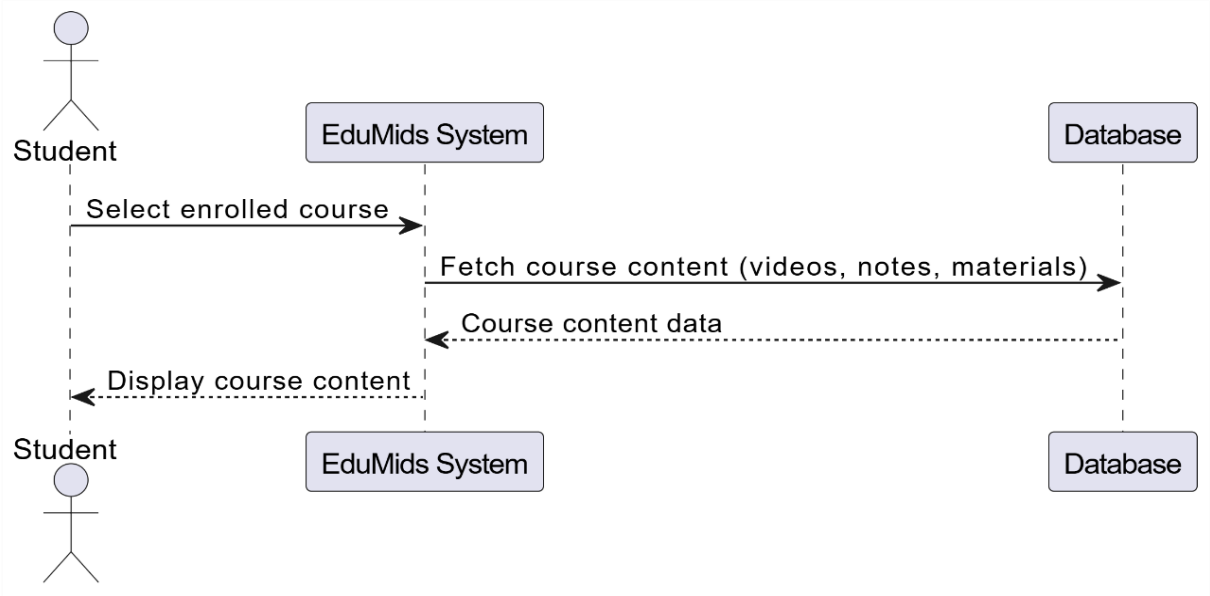
1.4.2 Sequence Diagram: Student_Login



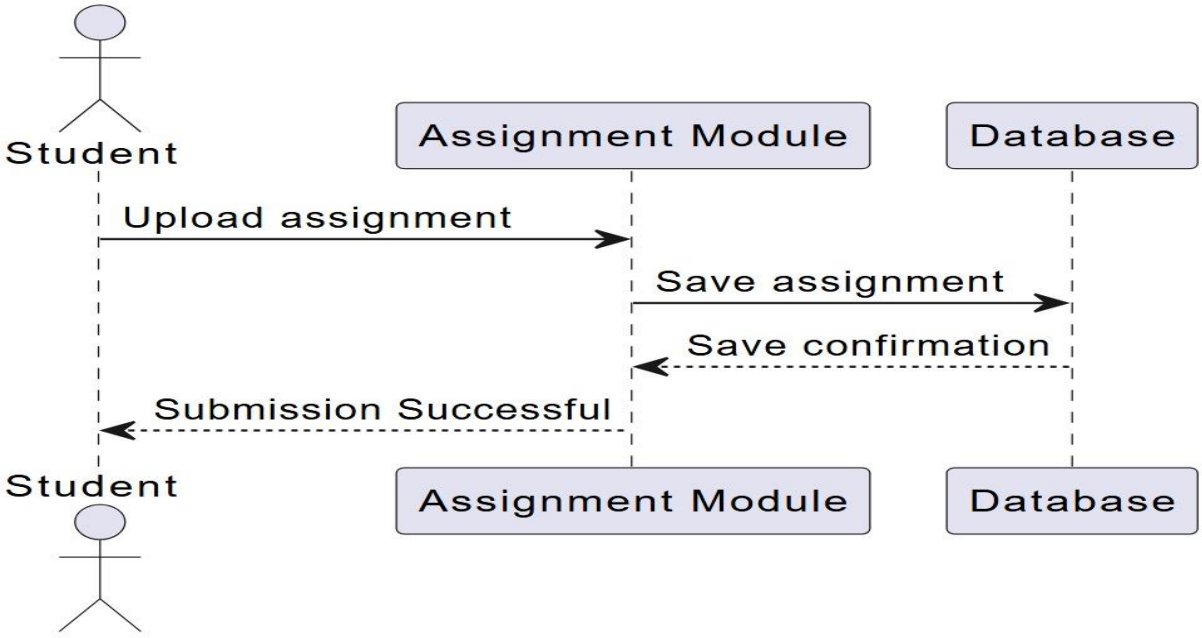
1.4.3 Sequence Diagram: Enroll_in_Course



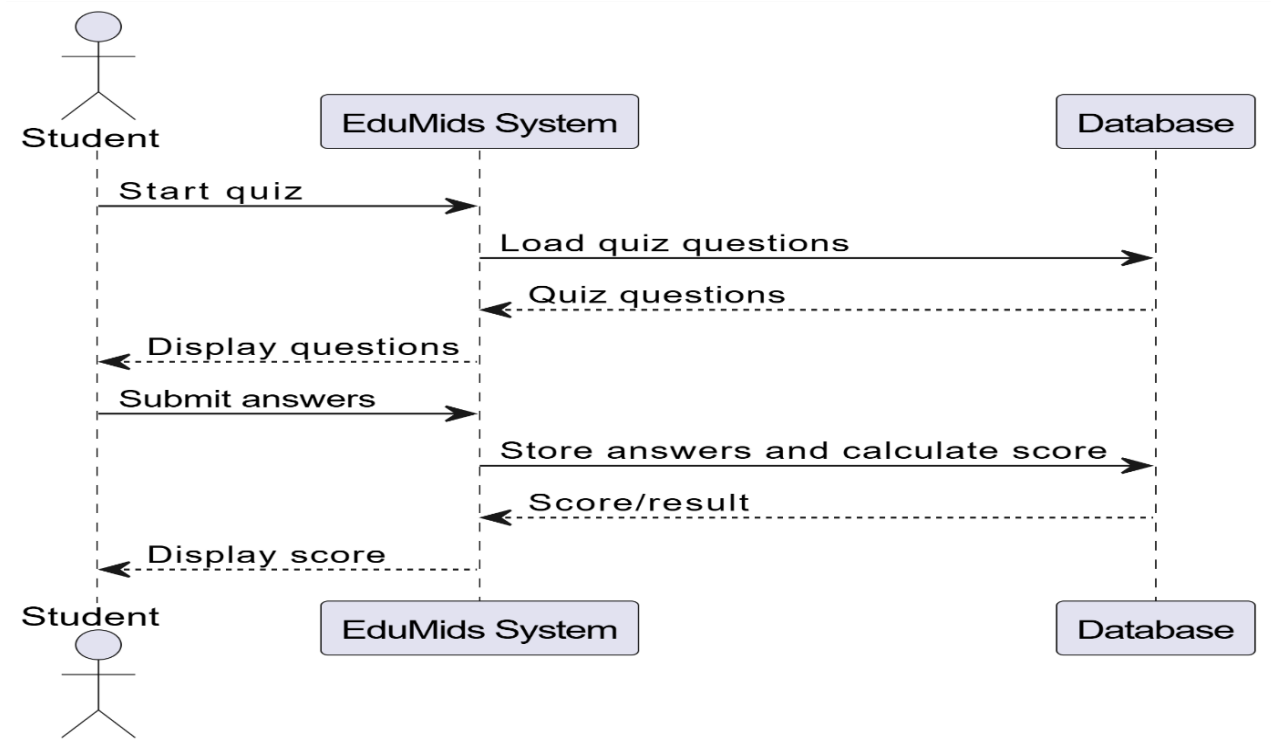
1.4.4 Sequence Diagram: View_Course_Content



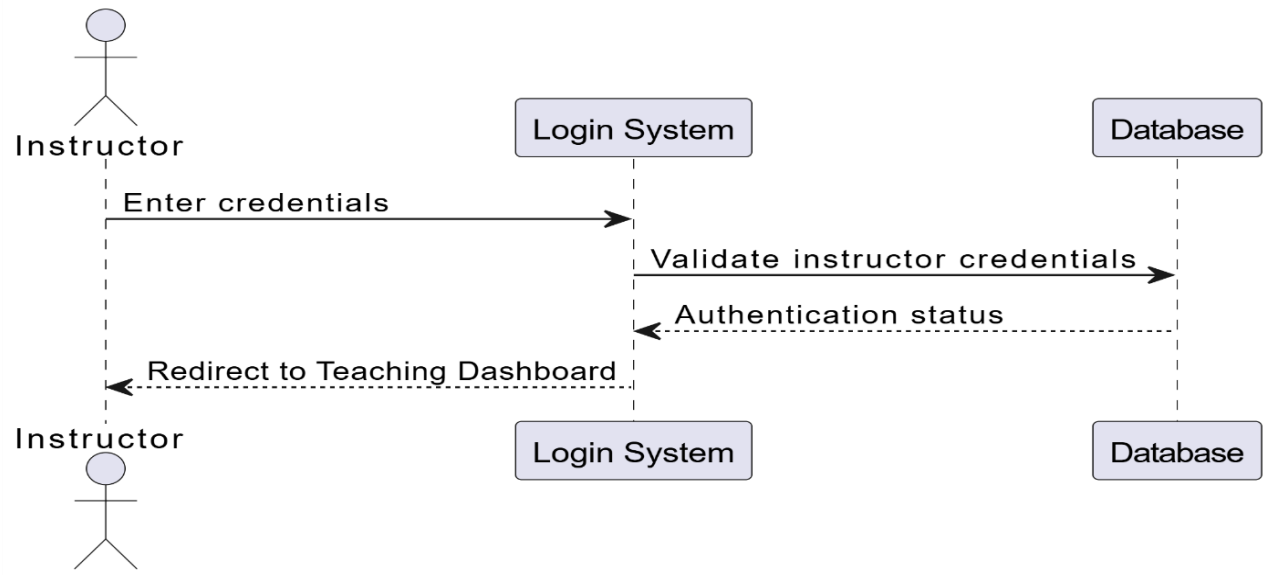
1.4.5 Sequence Diagram: Submit_Assignment



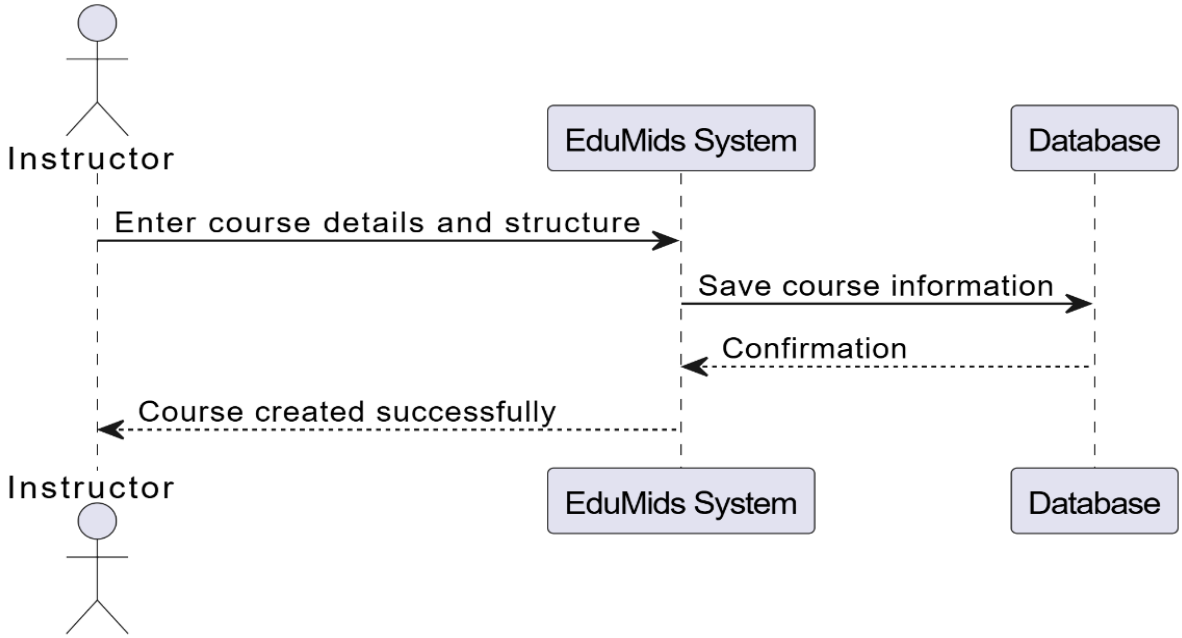
1.4.6 Sequence Diagram: Take_Quiz



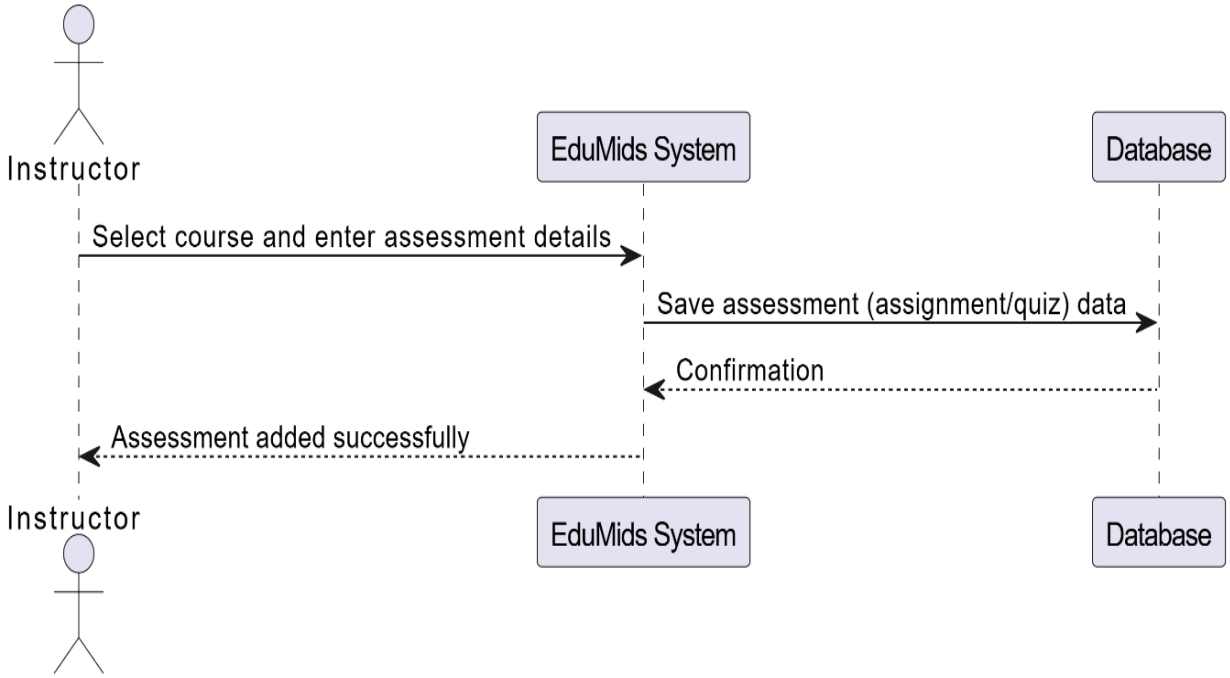
1.4.7 Sequence Diagram: Instructor_Login



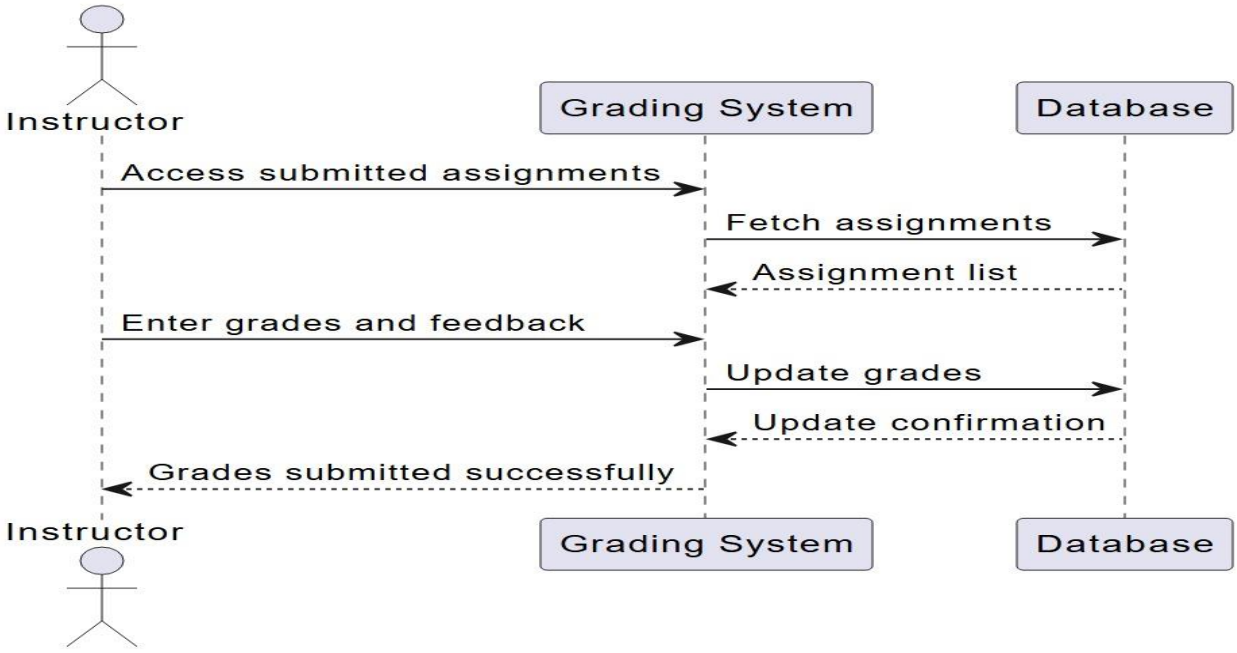
1.4.8 Sequence Diagram: Create_Course



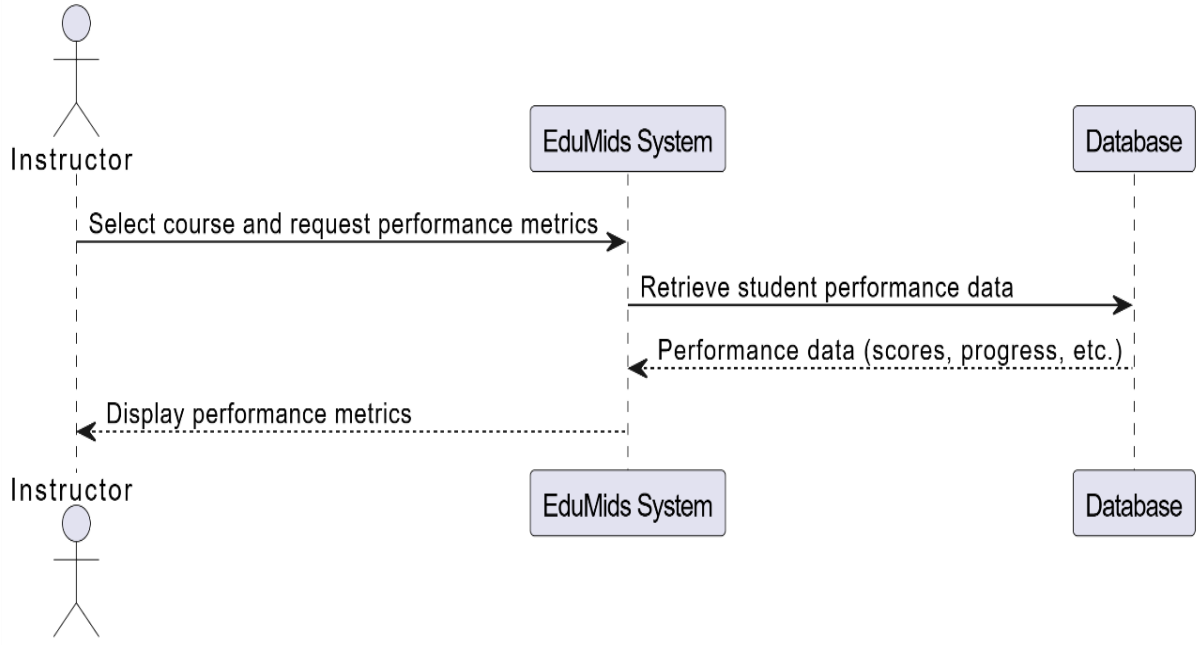
1.4.9 Sequence Diagram: Add_Assignment/Quiz



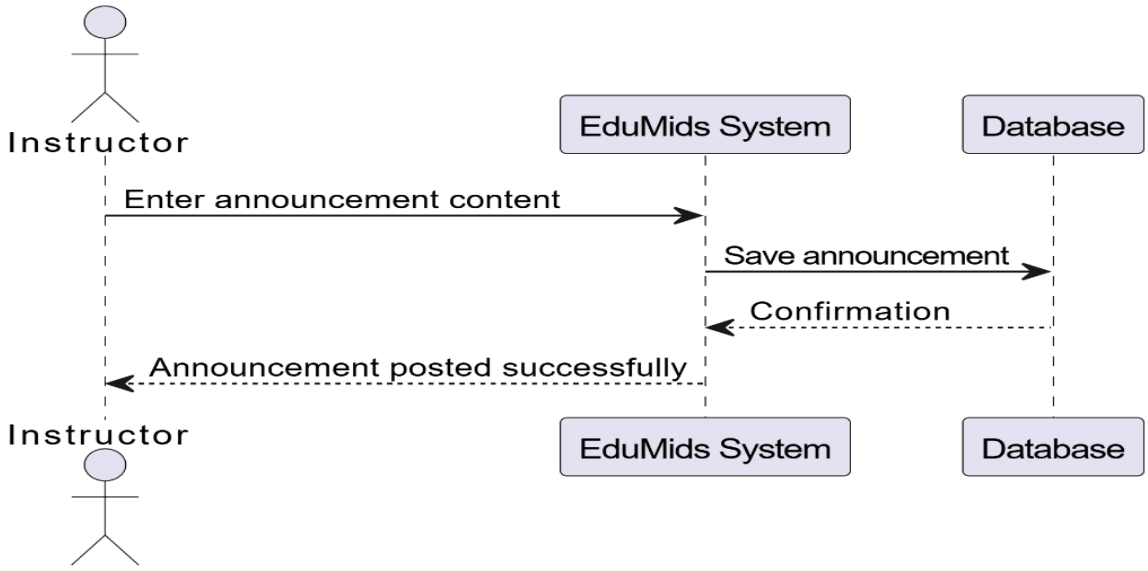
1.4.10 Sequence Diagram: Grade_Submission/Quiz



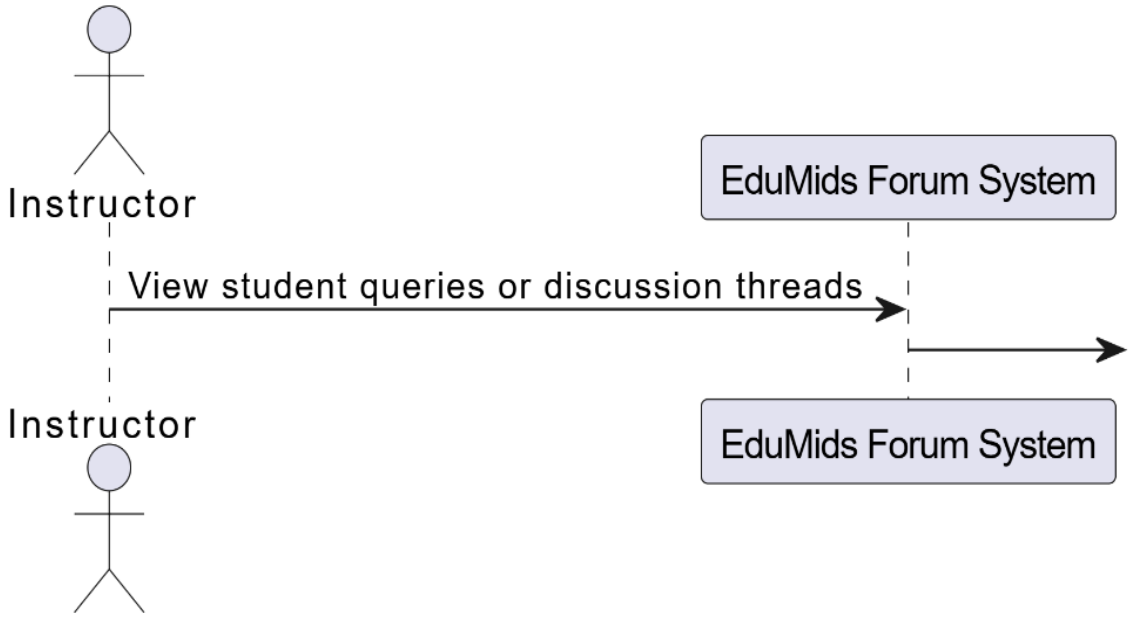
1.4.11 Sequence Diagram: View_Student_Performance



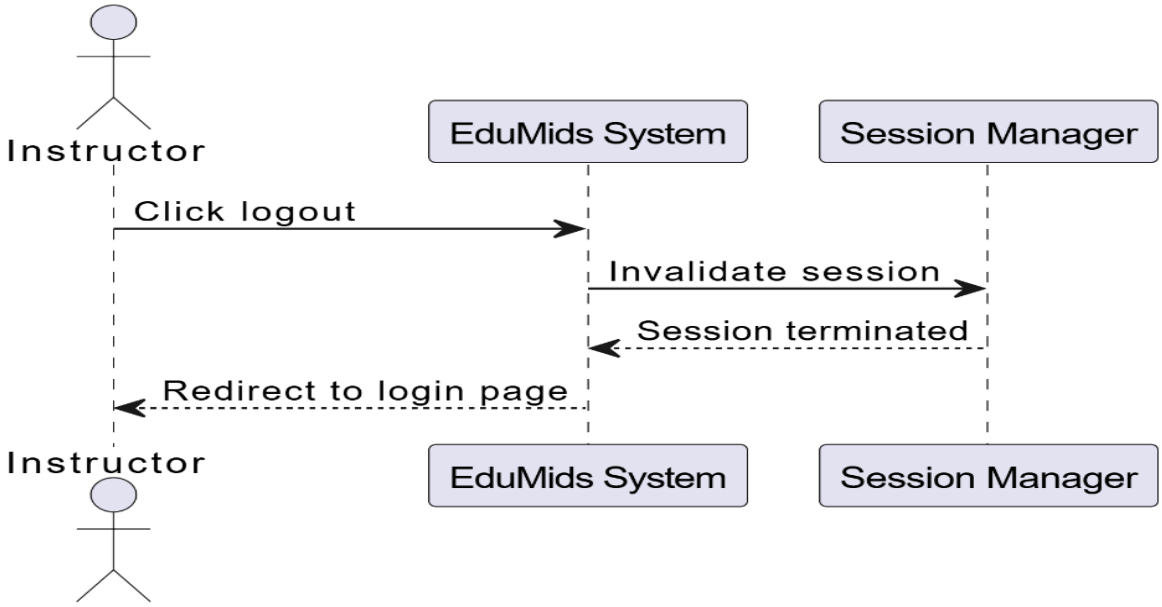
1.4.12 Sequence Diagram: Post_Announcement



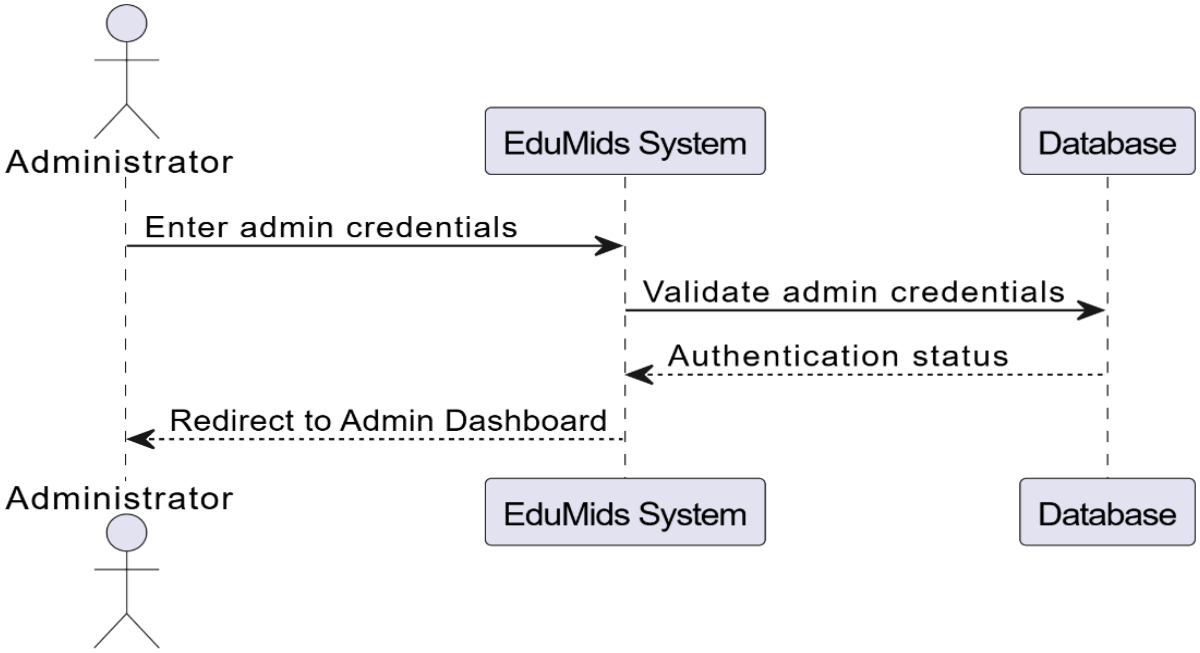
1.4.13 Sequence Diagram: Join_Discussion



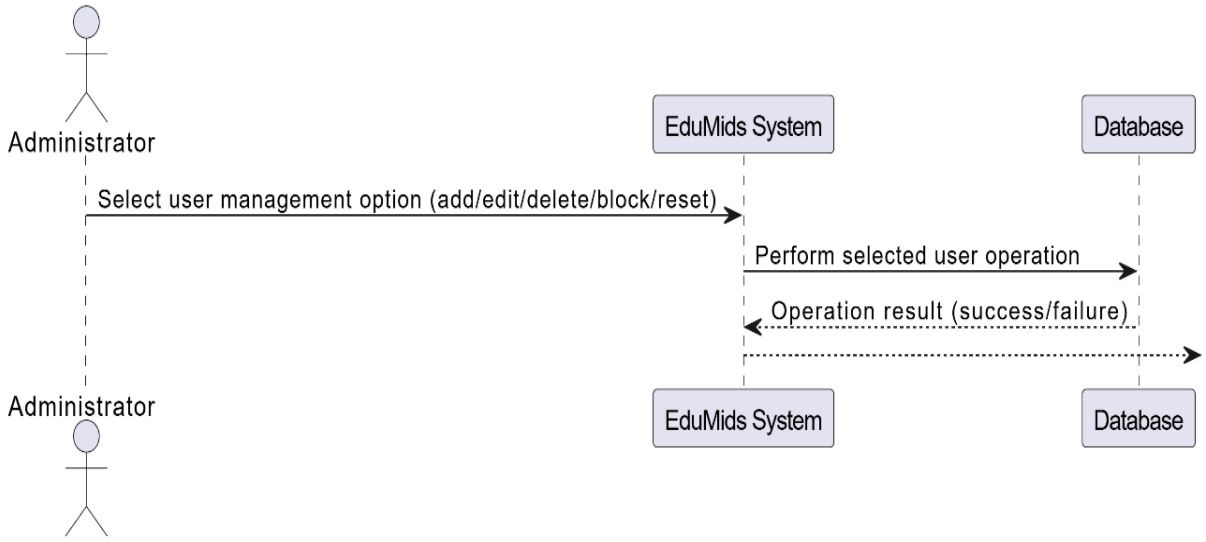
1.4.14 Sequence Diagram: UC_Logout



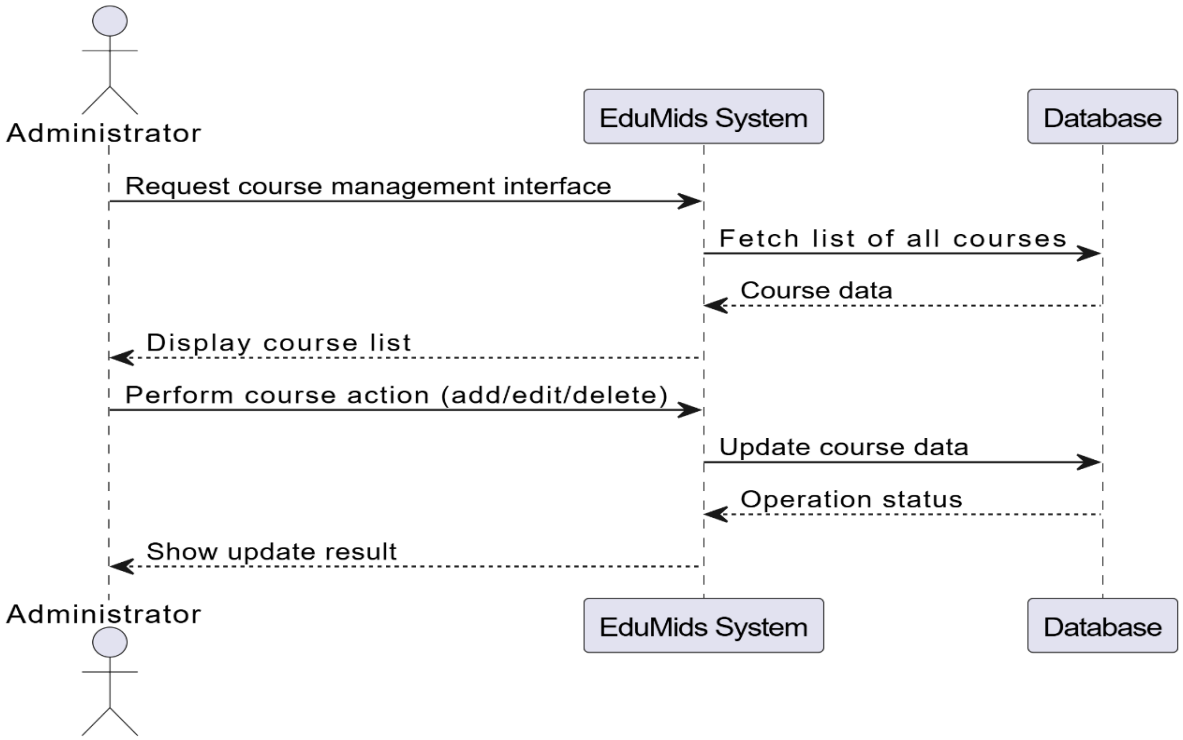
1.4.15 Sequence Diagram: Admin_Login



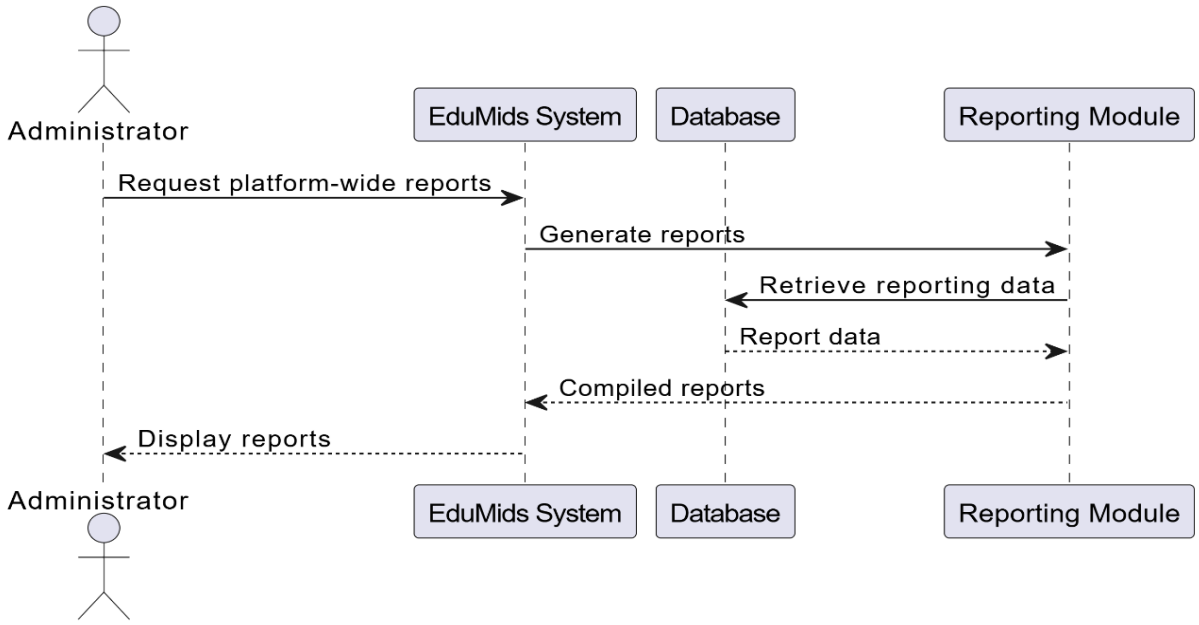
1.4.16 Sequence Diagram: Manage_Users



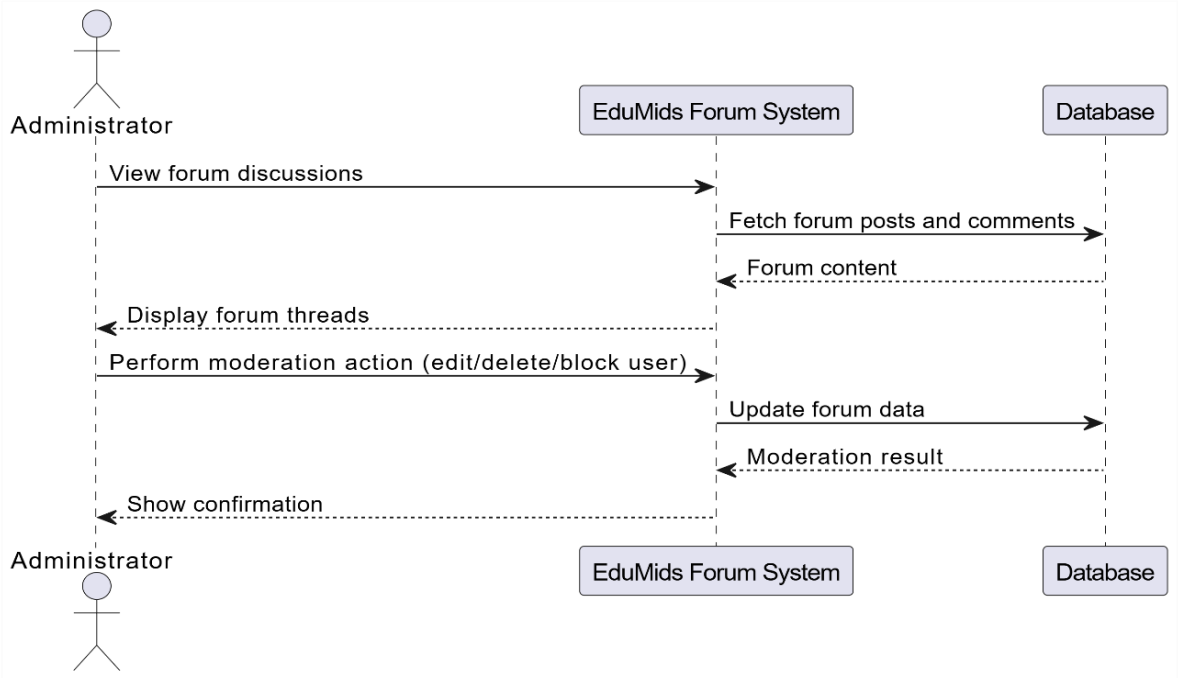
1.4.17 Sequence Diagram: Manage_Courses



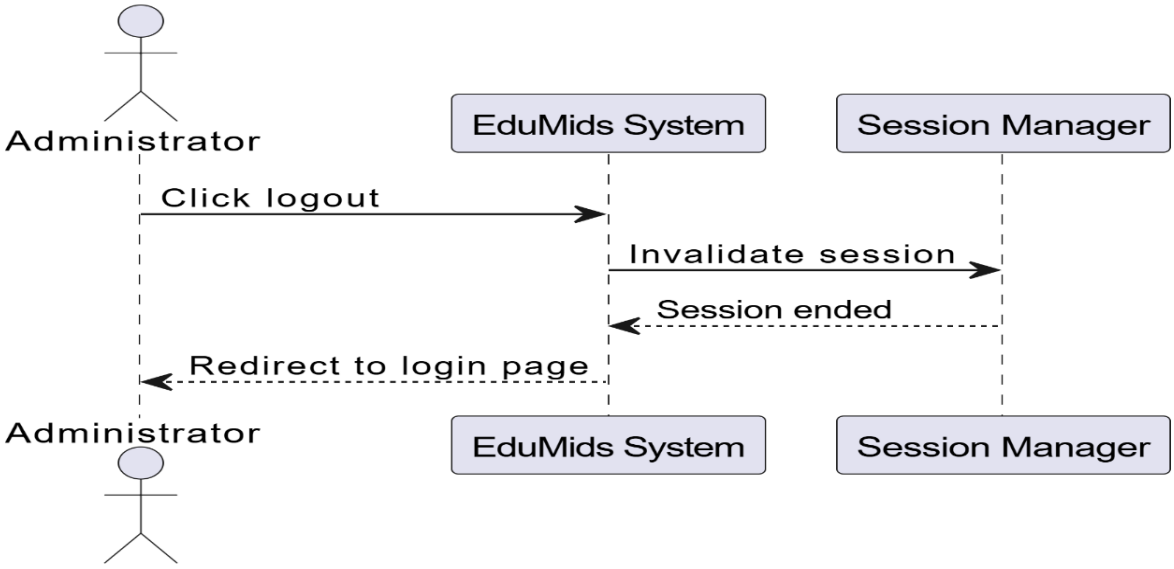
1.4.18 Sequence Diagram: View_System_Reports



1.4.19 Sequence Diagram: Moderate_Discussions



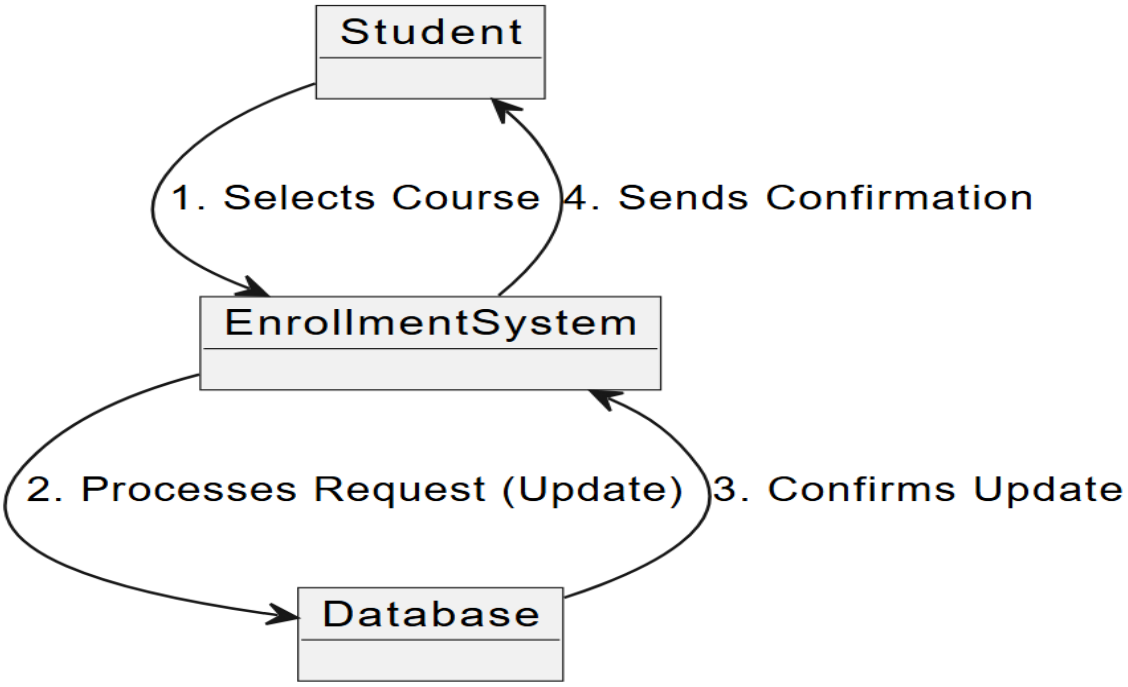
1.4.20 Sequence Diagram: Admin_Logout



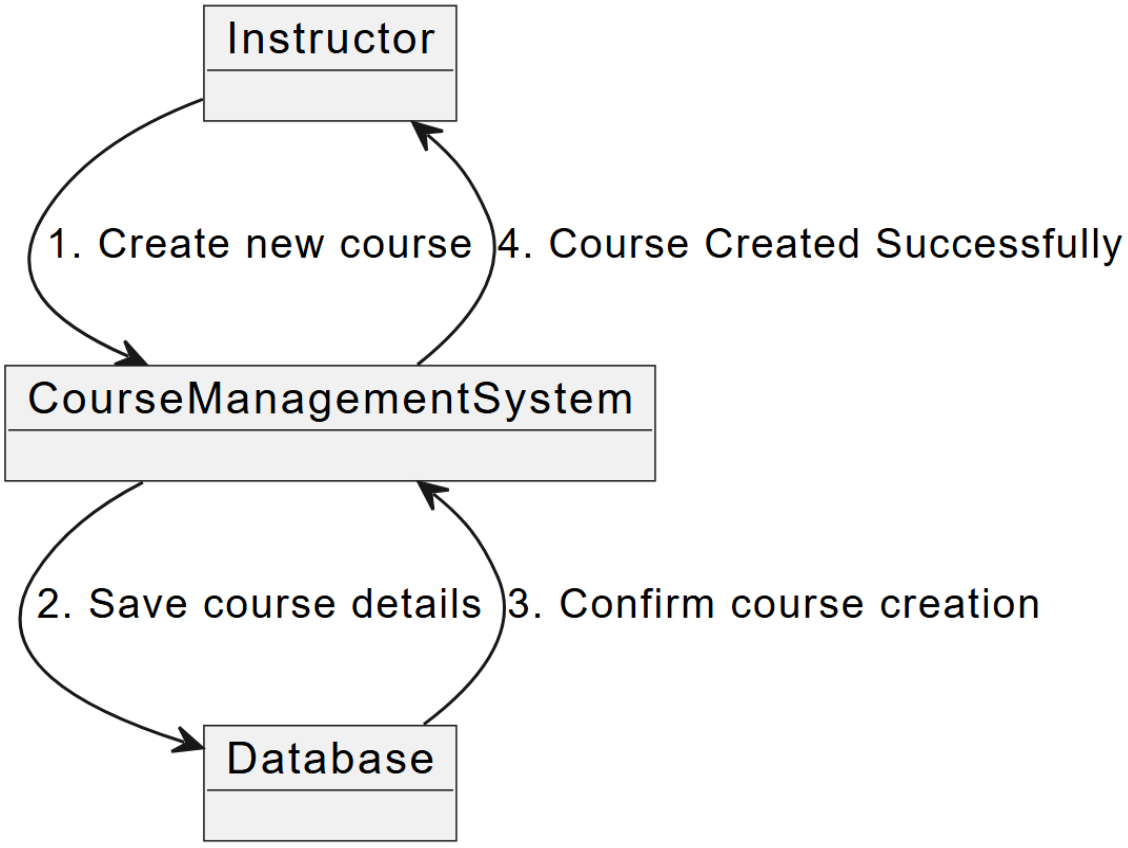
1.5 Collaboration Diagram

A **collaboration diagram** describes how various objects within the **EduMids** 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.

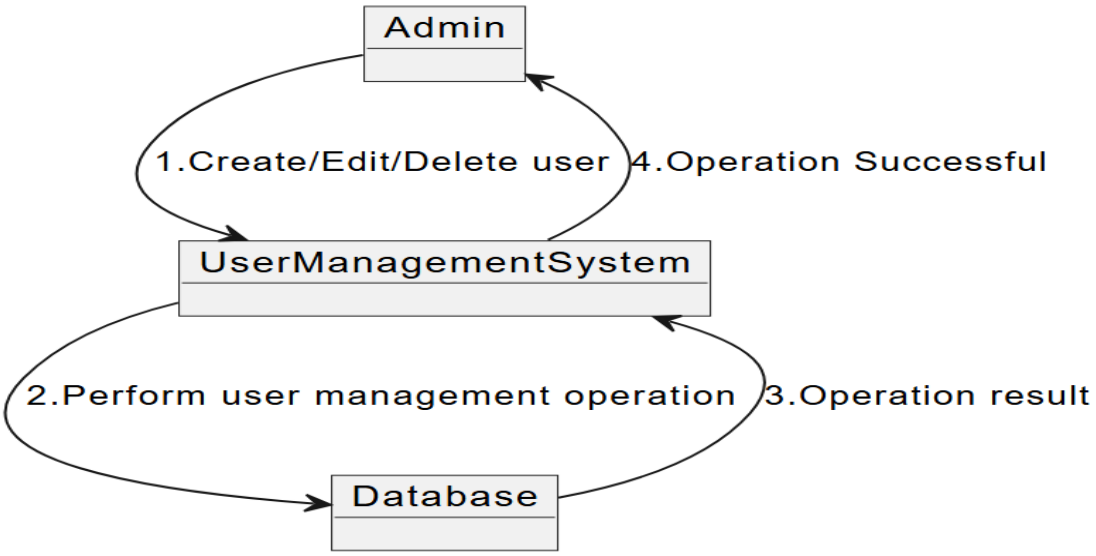
1.5.1 Collaboration Use Case: Enroll in Course



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 Course

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 **EduMids**-Your Pathway To khowledge 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 **EduMids**. 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 **EduMids**, 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 **EduMids**, entity classes would include:

- Course
- Student
- Assignment
- Quiz
- Grade

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

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 **EduMids** 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()

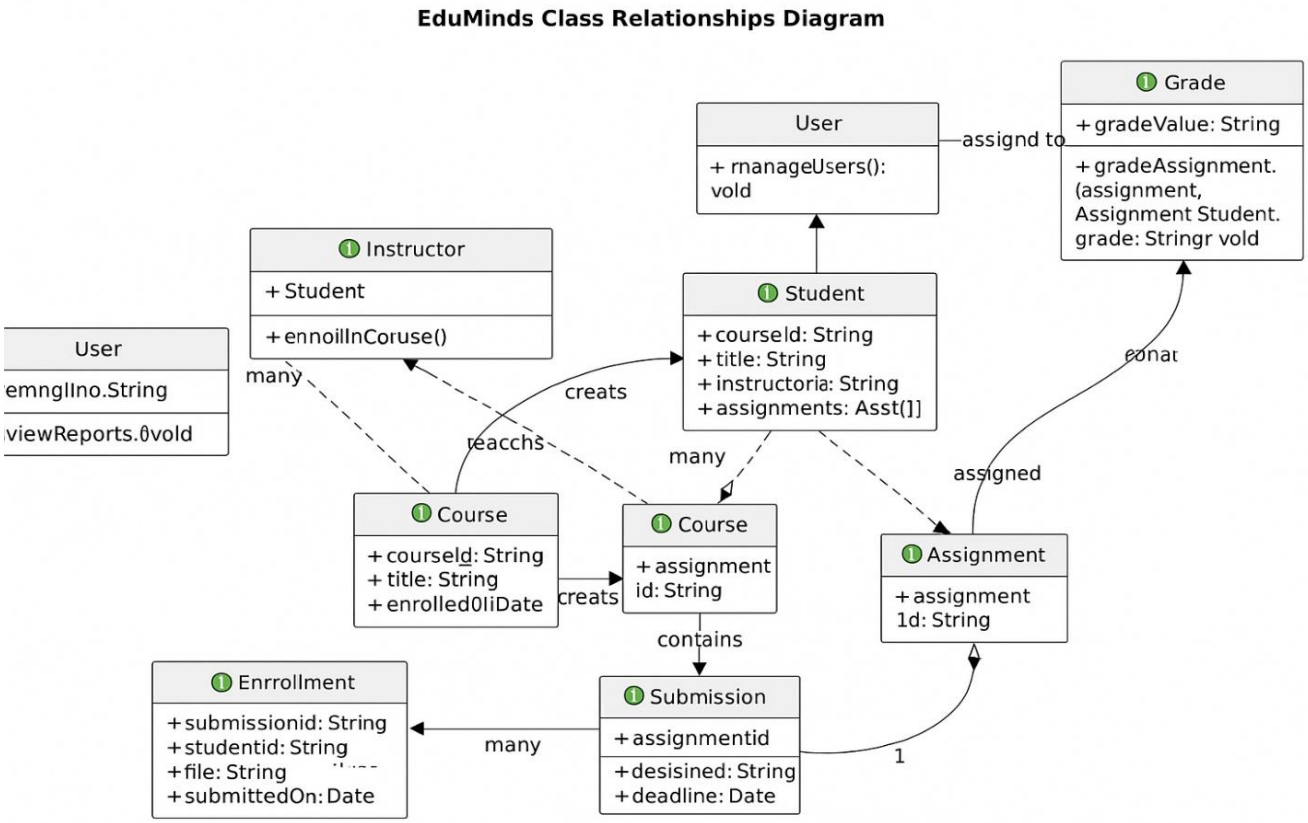
7.Class Visibility

- **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 **EduMids** 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.

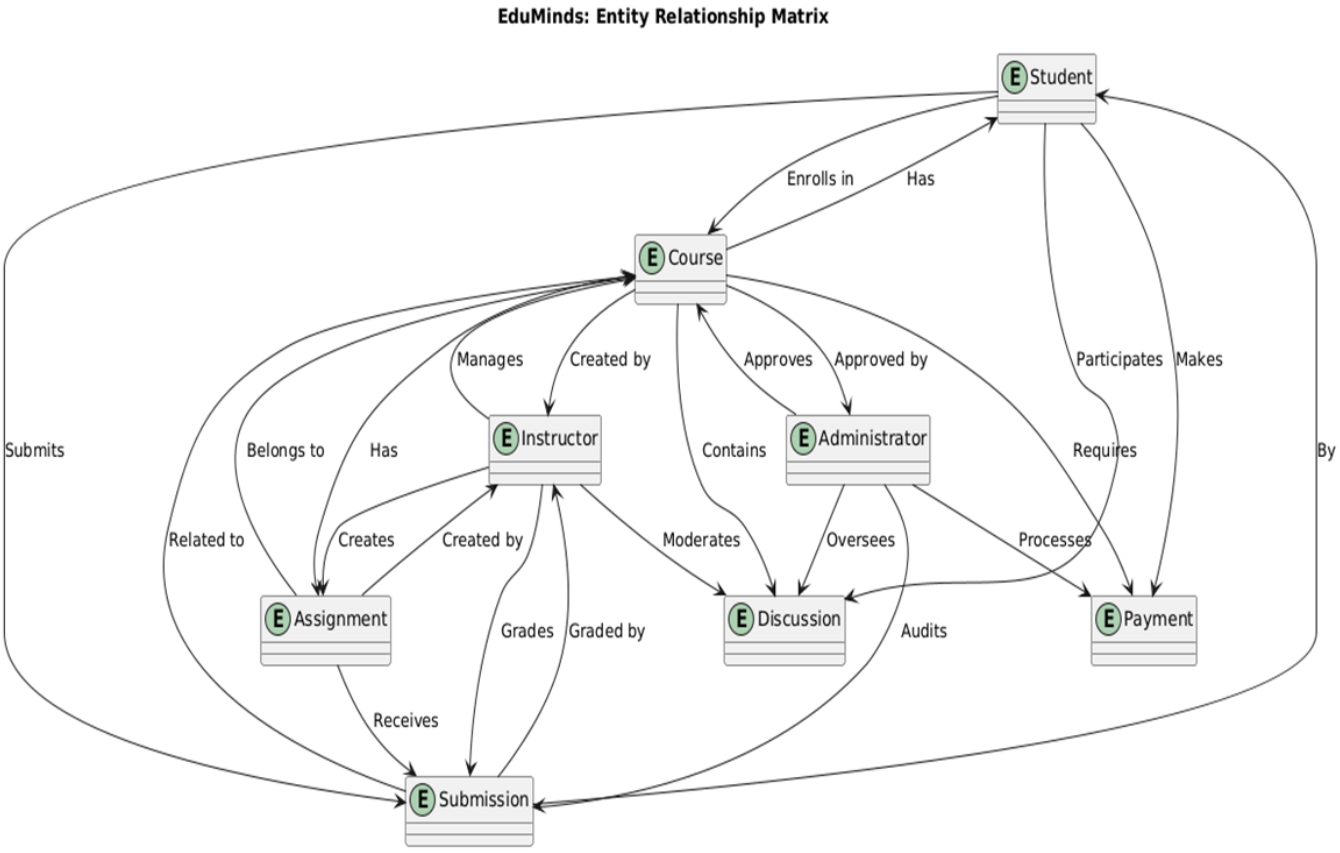
This model uses **primary keys**, **foreign keys**, **entity relationships**, and **cardinality** to represent the system's data behavior in a relational database system such as MongoDB (in practice with schema modeling) or SQL-based systems for this representation.

Identify Entities

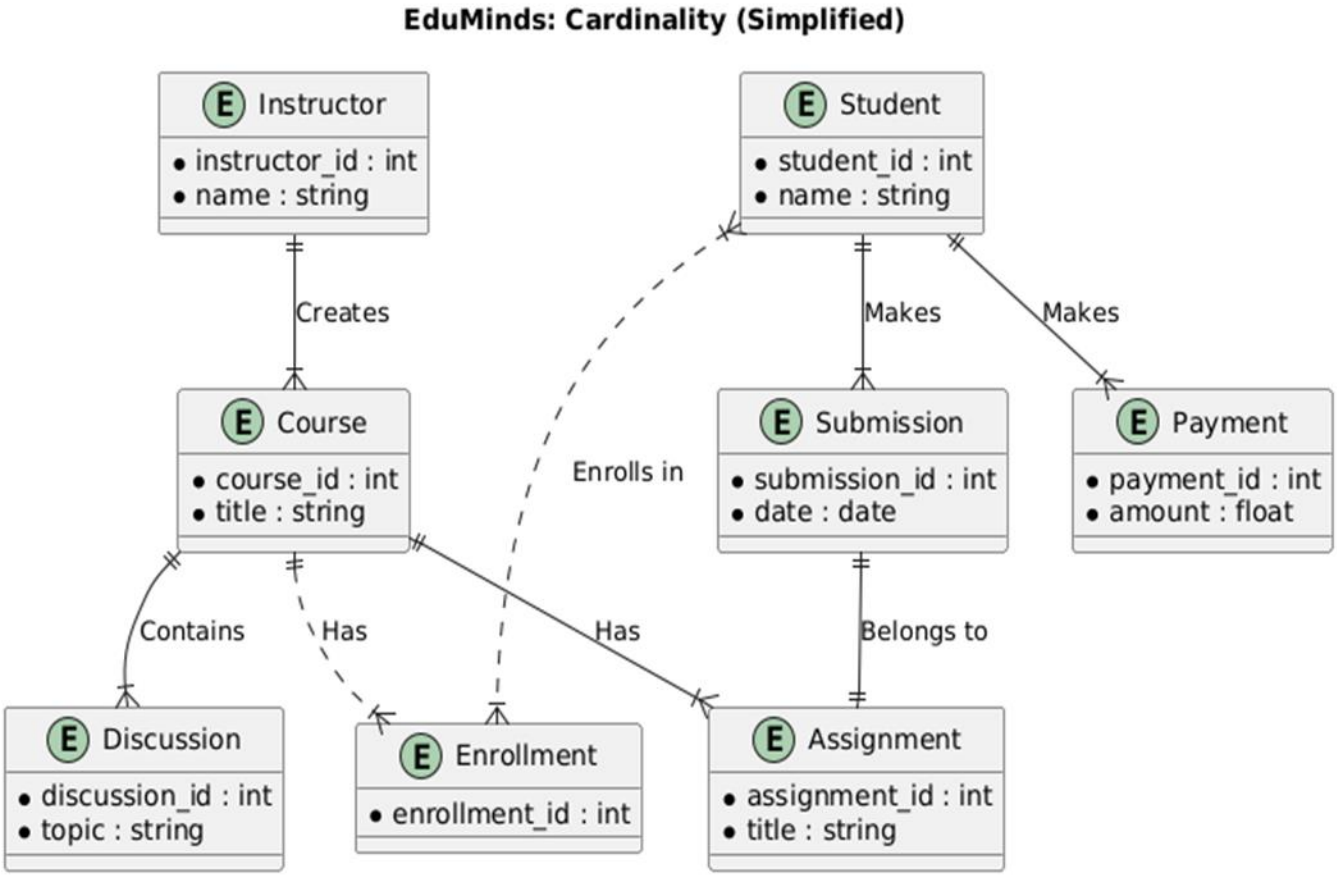
Entities for **EduMids** include:

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

Entity Relationship Matrix

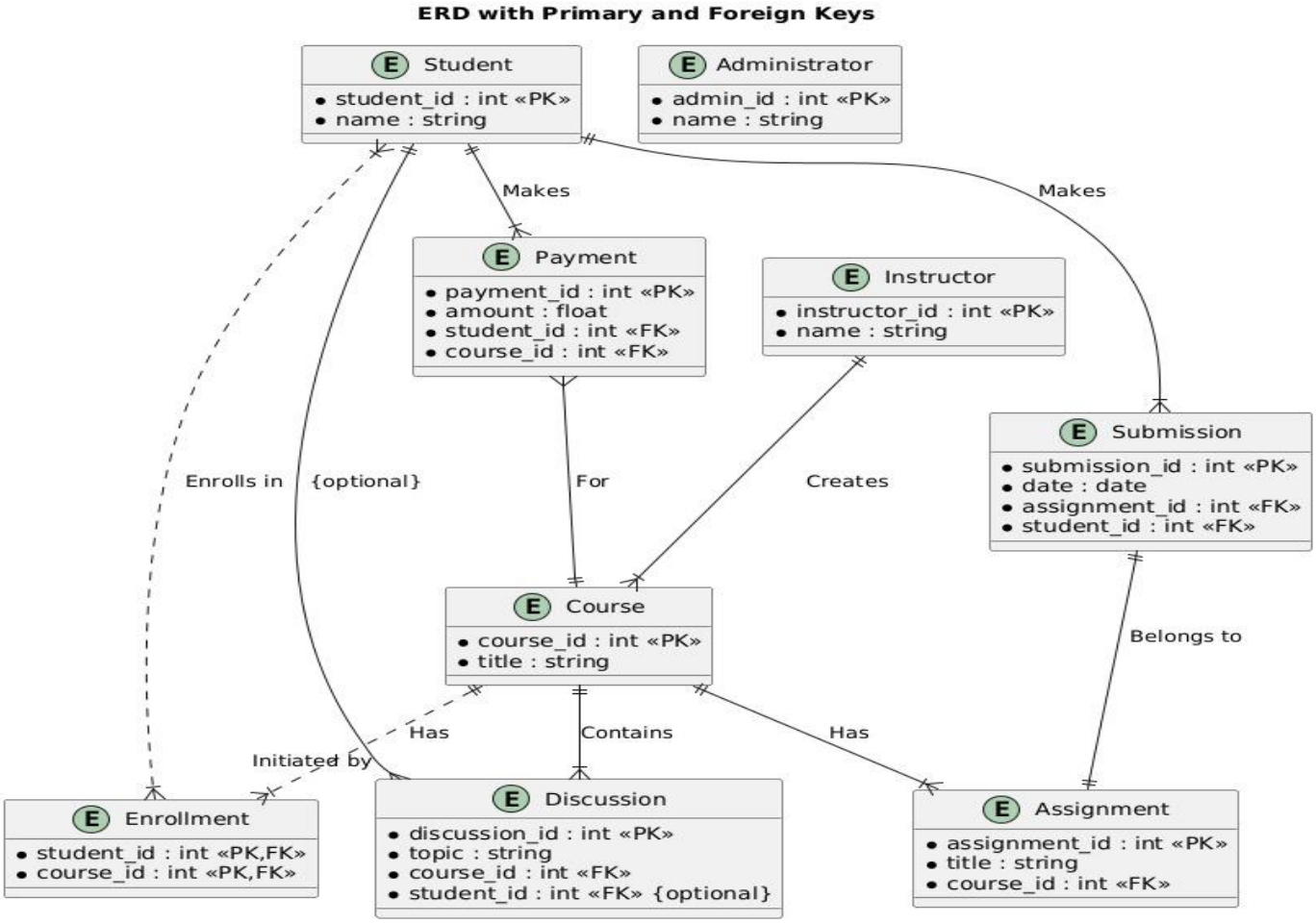


Cardinality (Simplified)



- 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



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