## **Final Consolidated Report**

Object-Oriented Analysis and Design – Fall 2025

Online Course Registration System



**Basmala Salah Mohamed**

**Rana Abdelhamid Dief**

**GitHub Repository:  
<https://github.com/RanaDief/OOAD_Project>**

# 

# 1. System Overview

The Online Course Registration System (OCRS) is a university-wide platform that enables students to browse available courses, register for classes, and manage their academic schedules. The system also supports instructors who manage their courses and administrators who oversee academic terms, user accounts, and course configurations.

The system aims to streamline the registration process, enforce academic rules (e.g., prerequisites, capacity limits, schedule conflicts), and ensure that all users interact with an efficient, secure, and user-friendly environment.

# 2. Project Objectives

The objectives of the OCRS are to:

* Provide students with a clear and accessible interface to register, drop, and review courses.
* Enable instructors to manage course information and view enrolled students.
* Support administrators in managing course offerings, user accounts, and registration regulations.
* Ensure validation of enrollment rules such as capacity, prerequisites, and scheduling.
* Enhance usability, reliability, and overall efficiency of the university registration workflow.

# 

# 

# 

# 

# 3. Actors and Their Roles

|  |  |
| --- | --- |
| Actor | Description |
| Student | |  | | --- | | Browses courses, registers/drops courses, views personal schedule. |  |  | | --- | |  | |
| Instructor | |  | | --- | | Updates course details and views enrolled students. |  |  | | --- | |  | |
| Administrator | Manage courses, users, academic terms, and system rules. |

### 4. Functional Requirements

**FR1:** Student can browse the course catalog.

**FR2:** Student can search/filter courses by department, level, instructor, etc.

**FR3:** Student can register for a course.

**FR4:** System validates prerequisites before registration.

**FR5:** System checks course capacity.

**FR6:** System detects scheduling conflicts.

**FR7:** Student can drop a course.

**FR8:** Student can view their weekly schedule.

**FR9:** Instructor can modify course information.

**FR10:** Instructor can view enrolled students.

**FR11:** Administrator can create, edit, or delete courses.

**FR12:** Administrator can manage user accounts.

**FR13:** System sends notifications for successful/failed registrations.

**FR14:** System logs all registration activities.

### 5. Non-Functional Requirements

**NFR1 — Usability:** Interface should be user-friendly and easy to navigate.

**NFR2 — Performance:** Course search and registration validation must respond within 3 seconds.

**NFR3 — Reliability:** The system must be available during all registration periods with minimal downtime.

**NFR4 — Security:** Only authenticated users can access the system; sensitive data must be protected.

**NFR5 — Scalability:** The system should support high user volume during peak times.

**NFR6 — Maintainability:** Code and system structure must be easily maintainable for future upgrades.

### 6. System Constraints & Assumptions

**Constraints:**

Registration actions must follow academic regulations.

Course data must be accurate and updated by administrators.

**Assumptions:**

Users have valid university accounts.

Internet connection is available.

Course schedules and prerequisites are predefined.

### 

### 7. Use Case List (Brief Descriptions)

|  |  |
| --- | --- |
| Use Case | Description |
| UC-1 Browse Courses | |  | | --- | | Student views the available course catalog. |  |  | | --- | |  | |
| UC-2 Register for Course | |  | | --- | | Student registers for a chosen course after validations. |  |  | | --- | |  | |
| UC-3 Drop Course | |  | | --- | | The student removes an enrolled course. |  |  | | --- | |  | |
| UC-4 View Schedule | |  | | --- | | Students view their weekly course schedule. |  |  | | --- | |  | |
| UC-5 Edit Course Information | |  | | --- | | The instructor updates course details. |  |  | | --- | |  | |
| UC-6 View Enrolled Students | |  | | --- | | The instructor views list of enrolled students. |  |  | | --- | |  | |
| UC-7 Manage Courses | The administrator manages course creation, editing, and removal |

### 

### 8. Use Case Diagram



### 

### 1. Detailed Use Case Scenarios:

**UC1 — Browse Courses**

**Actor:** Student

**Preconditions:** Student is logged in.

**Main Flow:**

1. Student opens the “Course Catalog”.

2. System displays all available courses.

3. Student may filter by department, level, day, or instructor.

4. Student selects any course to view details.

**Postconditions:** None (view-only).

**Exceptions:** None.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**UC2 — Register for Course**

**Actor:** Student  
 **Preconditions:**

· Student is logged in.

· Registration period is open.

**Main Flow:**

1. Student selects a course from the catalog.

2. System retrieves course info.

3. System validates prerequisites.

4. System checks for schedule conflicts.

5. System checks course capacity.

6. If all validations pass → System registers the student.

7. System updates the student schedule.

8. System sends confirmation message.

**Postconditions:**

· Course added to student’s schedule.

· Course capacity decreases.

**Exceptions:**

· Missing prerequisites → Registration denied.

· Time conflict → Registration denied.

· Course full → Registration denied.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**UC3 — Drop Course**

**Actor:** Student  
**Preconditions:**

· Student is logged in.

· Add/Drop period is open.

**Main Flow:**

1. Student opens list of registered courses.

2. Student selects a course to drop.

3. System removes the course from schedule.

4. System updates capacity.

5. System sends confirmation.

**Postconditions:**

· Course removed from schedule.

**Exceptions:**

· Drop period closed → Denied.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**UC4 — View Schedule**

**Actor:** Student

**Preconditions:** Student is logged in.

**Main Flow:**

1. Student opens “My Schedule”.

2. System retrieves all registered courses.

3. System displays the full weekly schedule.

4. Postconditions: None (view-only).

**Exceptions:** None.

**UC5 — Edit Course Information**

**Actor:** Instructor  
**Preconditions:**

· Instructor teaches the selected course.

**Main Flow:**

1. Instructor selects a course.

2. System displays current course details.

3. Instructor modifies description/materials/requirements.

4. System saves updates.

**Postconditions:**

· Course information updated.

**Exceptions:**

· Instructor attempts editing unrelated course → Denied.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**UC6 — View Enrolled Students**

**Actor:** Instructor

**Preconditions:** Instructor teaches the course.

**Main Flow:**

1. Instructor chooses one of their courses.

2. System retrieves list of enrolled students

3. Instructor views names, IDs, and emails.

**4. Postconditions:** None (view-only).

**Exceptions:** Instructor views a course not assigned to them → denied.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**UC7 — Manage Courses**

**Actor:** Administrator  
**Preconditions:**

· Admin is logged in.

**Main Flow:**

1. Admin opens course management dashboard.

2. Admin chooses Add/Edit/Delete.

3. System performs the requested operation.

4. System updates database.

5. System displays confirmation.

**Postconditions:**

· Course catalog updated.

**Exceptions:**

Removing a course during active registration → Warning.

## **2. Conceptual Classes**

* **Student:** Registers for courses, views schedule.
* **Instructor:** Manages course content and views enrolled students.
* **Administrator:** Manages courses, users, and terms.
* **Course:** Contains title, description, capacity, schedule, and instructor.
* **Registration:** Represents enrollment of a student in a course.
* **Schedule:** Student’s organized weekly plan of courses.
* **Prerequisite:** Required course that must be completed before enrollment.
* **Semester/Term:** Defines the academic period.
* **Notification:** Communicates confirmation or failure messages.
* **TimeSlot:** Represents meeting time(s) for a course.
* **UserAccount:** General user attributes inherited by Student, Instructor, Admin.

## **3. Class Relationships**

### Associations

* Student ↔ Registration (1 student has many registrations)
* Course ↔ Registration (1 course can have many students)
* Course ↔ Instructor (1 instructor teaches many courses)
* Course ↔ TimeSlot (1..\* time slots)
* Student ↔ Schedule (1 schedule per student)
* Administrator ↔ Course (manages courses)
* Administrator ↔ UserAccount (creates/edits accounts)

### Aggregations

* Schedule contains TimeSlots
* Course contains CourseMaterials

### Inheritance

* UserAccount  
   ↳ Student  
   ↳ Instructor  
   ↳ Administrator

## 

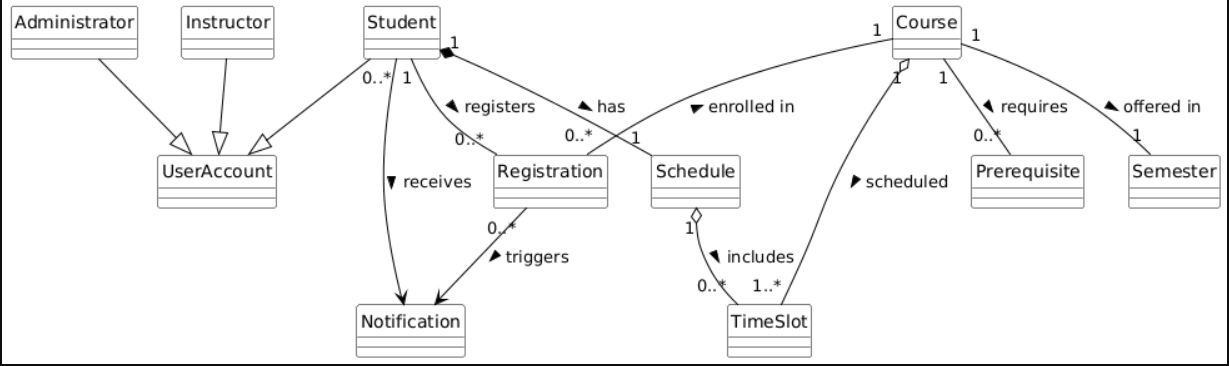
## 

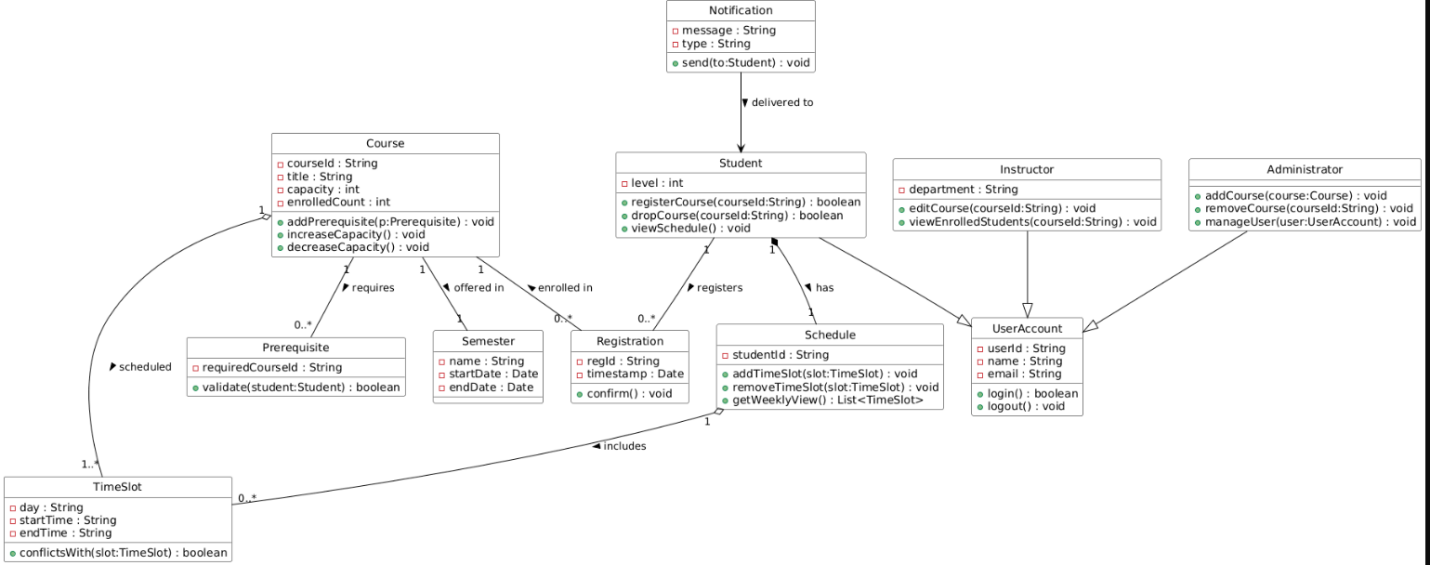
## 

## 

## 

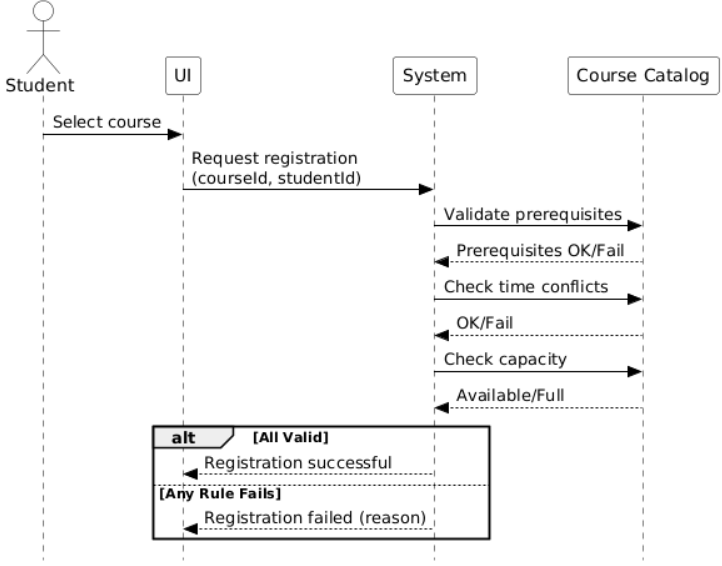
## **4. Analysis-Level Class Diagram**



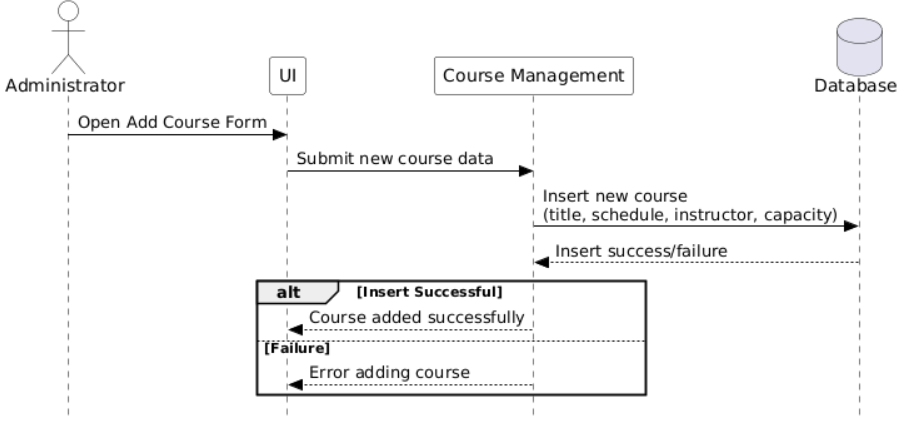


## **5. Sequence Diagrams**

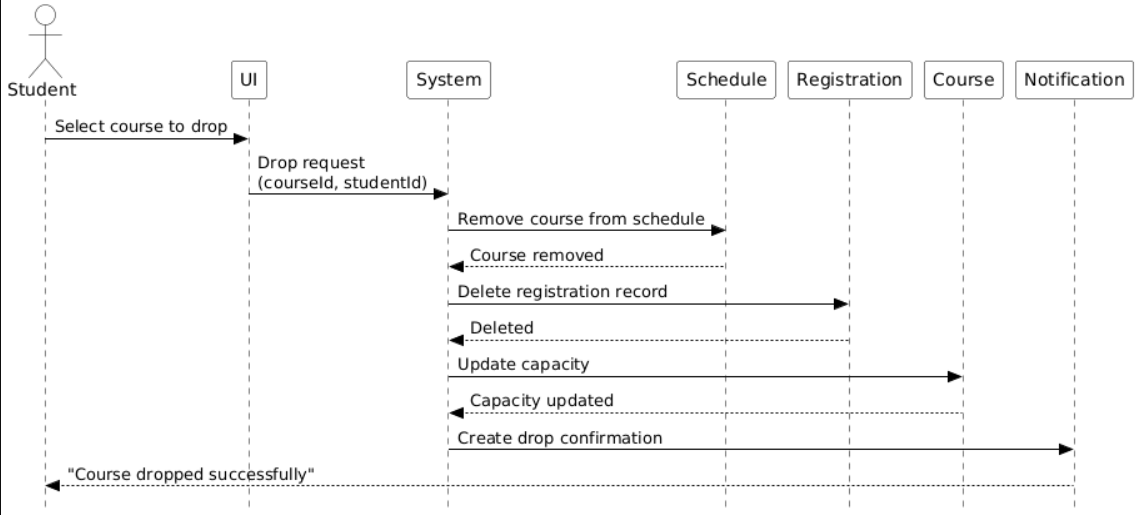
**SEQUENCE DIAGRAM 1 (Register for Course)**



**SEQUENCE DIAGRAM 2 (Add New Course)**



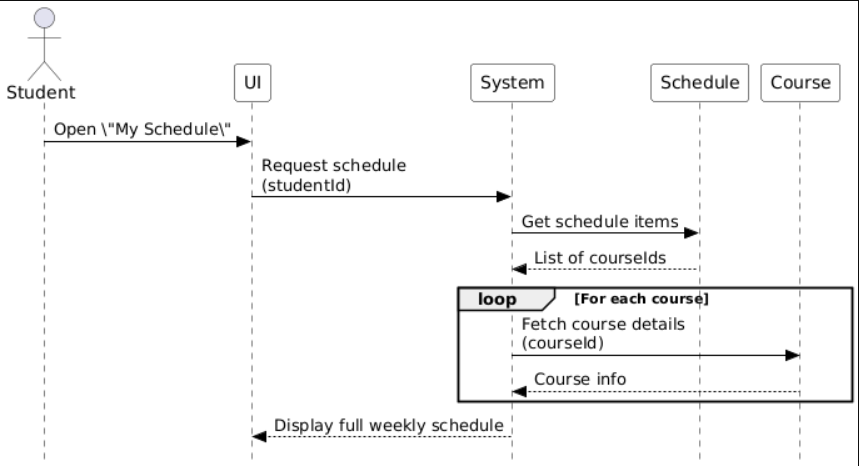
# **SEQUENCE DIAGRAM 3 (Drop Course)**



## 

# 

# **Sequence Diagram 4—View Schedule**



# 

# 

# 

# 

# 

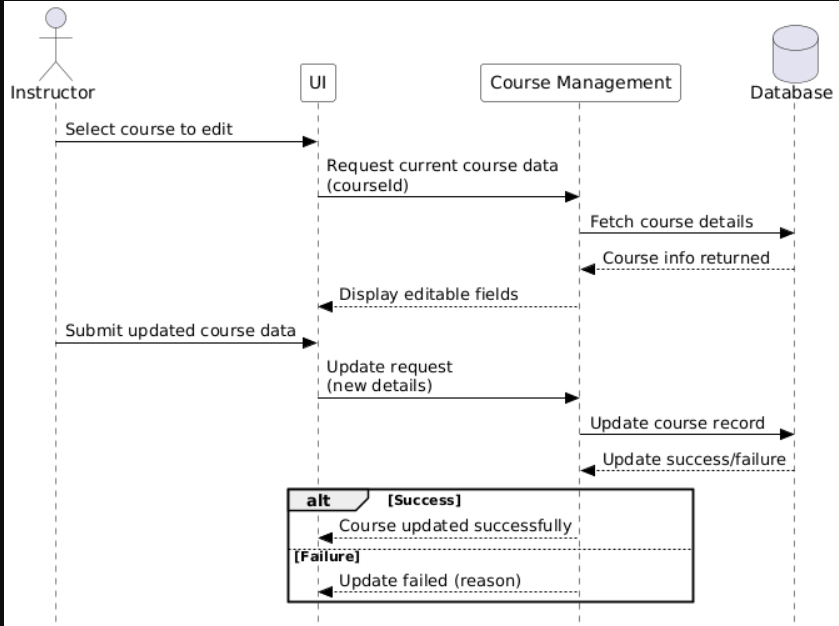
# 

# 

# 

# 

# **Sequence Diagram 5—Edit Course Information (Instructor)**



# 

# 

# 

# 

# 

# 

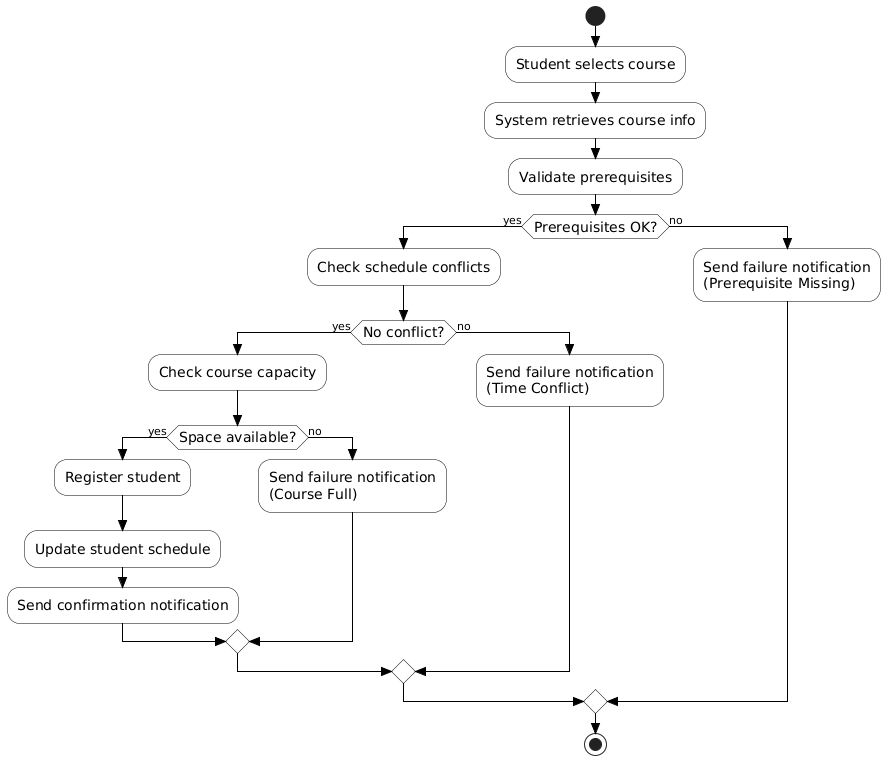
# **Sequence Diagram 6—View Enrolled Students (Instructor)**



## **6. Activity Diagram—Course Registration Process**

Include the workflow:

1. Student selects course
2. System retrieves course info
3. Validate prerequisites
4. Validate schedule conflict
5. Validate capacity
6. If all pass → register student
7. Update schedule
8. Send notification
9. End



## 

## 

## **7. Expanded Notes & Modeling Decisions**

### 1. Inheritance & User Hierarchy

The **UserAccount** class is used as a *generalized superclass* for all user types in the system.  
 This design decision is based on the observation that:

* Student, Instructor, and Administrator all share common attributes (Name, Email, UserID).
* They also share similar authentication behaviors (login/logout).
* Using inheritance avoids duplication, increases maintainability, and supports polymorphism if needed later in the design.

Thus:

* **Student**, **Instructor**, and **Administrator** extend **UserAccount**.
* This aligns with OO principles of *generalization and specialization*.

### 2. Registration as a Separate Entity

Registration is modeled as an independent **conceptual class** rather than an attribute inside Student or Course.

**Justification:**

* Students can register for *multiple* courses.
* Courses contain *multiple* students.
* This creates a **many-to-many relationship**.
* A dedicated **Registration** class resolves this many-to-many relation cleanly.

It also enables:

* Adding timestamps
* Tracking registration status
* Logging operations
* Supporting notifications
* Extending features in future semesters

This follows strong database and OO modeling conventions.

### 

### 3. Schedule Modeled as a Composition

A Student owns exactly one **Schedule**, and deleting a student should remove their schedule.

Therefore:

* The relationship is modelled as **composition**.
* The schedule cannot exist without its owner.
* It helps visualize the student’s weekly timetable.

This ensures a cohesive representation of the student’s academic workload.

### 

### 4. TimeSlot as an Independent Reusable Concept

The **TimeSlot** class is created to represent course meeting times (e.g., Monday 8–10am).

Reasons:

* Scheduling conflict detection requires comparing day/time periods.
* Multiple courses may share the same time slots.
* TimeSlots can be reused or mapped without duplicating data.

This supports clean scheduling validation.

### 

### 5. Course Aggregation with Time Slots

The relationship between Course and TimeSlot is modeled as **aggregation**:

* A course consists of one or more time slots.
* TimeSlots can exist independently of the course creation or deletion.

Example:  
 A time slot model can be reused across different courses.

This choice improves modularity and prevents unnecessary deletion dependencies.

### 6. Course Prerequisites

Prerequisites are modeled using a **Prerequisite** class instead of a simple list of course IDs.

Why:

* A course may require multiple prerequisites.
* Prerequisites may evolve into more complex validation rules (e.g., minimum grade).
* This design supports extension and improves flexibility.

This follows the OO principle of "anticipating change."

### 7. Semester as a Separate Class

The **Semester** class exists to:

* Assign courses to academic terms
* Support future features such as overlapping semesters
* Enable filtering courses by semester
* Facilitate future reporting requirements

This makes the model more future-proof.

### 8. Notification Added for Confirmation & Feedback

Notifications are used for essential user feedback:

* Successful registration
* Failed registration
* Drop confirmations
* Administrative updates

Having a dedicated **Notification** class allows:

* Logging messages
* Categorizing message types
* Future expansion (email/SMS)

### 9. Clear Separation of Responsibilities

The model reflects proper OO separation:

* **Course** handles academic content and capacity
* **Registration** handles enrollment linking
* **Schedule** manages weekly structure
* **Notification** handles system messages
* **UserAccount** hierarchy manages user roles

This separation improves readability, maintainability, and modularity.

### 10. Avoiding Premature Design Complexity

Phase 2 focuses on analysis-level modeling:

* Minimal attributes
* No data types unless necessary
* No method bodies
* Focus on conceptual understanding

This aligns strictly with UML analysis principles and avoids mixing design-level decisions prematurely.

**Fully Detailed Class Descriptions**

**Package: user**

**2.1 UserAccount (Superclass)**

**Purpose:** General identity and authentication information for all system users.

**Attributes**

 userId : String — Unique identifier for the user

 name : String — Full user name

 email : String — Official contact email

 password : String — Encrypted login password

**Methods**

 login(email:String, password:String) : boolean — Validates user credentials

 logout() : void — Ends the user session

**Role in System**

 Parent class for Student, Instructor, and Administrator

 Avoids duplication of shared data fields

 Supports polymorphism (UserAccount references)

**2.2 Student (extends UserAccount)**

**Purpose:** Represents a student who registers for courses and maintains a schedule.**Attributes**

 studentLevel : int — Year/level of the student (e.g., 1, 2, 3, 4)

**Methods**

 registerForCourse(courseId:String) : boolean — Attempts registration and returns status

 dropCourse(courseId:String) : boolean — Removes a registered course

 viewSchedule() : Schedule — Retrieves the student’s weekly schedule

**Relationships**

 Has **one Schedule** (composition)

 Has **many Registration records**

 Receives **Notifications**

**2.3 Instructor (extends UserAccount)**

**Purpose:** Manages academic course information and views student lists.

**Attributes**

 department : String — Department responsible for instructor’s courses

**Methods**

 editCourseInfo(courseId:String, data:Map) : boolean — Modify course description, materials,

etc.

 viewEnrolledStudents(courseId:String) : List<Student> — Retrieves student details

**Relationships**

 **Teaches multiple courses2.4 Administrator (extends UserAccount)**

**Purpose:** Manages courses, users, and registration rules for the system.

**Methods**

 addCourse(course:Course) : boolean

 editCourse(courseId:String, data:Map) : boolean

 deleteCourse(courseId:String) : boolean

 manageUser(user:UserAccount) : boolean

**Relationships**

 Interacts with **Course**, **UserAccount**, and system configurations.

**Package: course**

**2.5 Course**

**Purpose:** Represents an academic course available for registration.

**Attributes**

 courseId : String — Unique course identifier

 title : String — Course name

 description : String — Course summary and content

 capacity : int — Maximum students allowed

 currentEnrollment : int — Current number of enrolled students

**Methods** hasSpace() : boolean — Checks if enrollment < capacity

 incrementEnroll() : void — Increase student count

 decrementEnroll() : void — Decrease student count

**Relationships**

 1 Instructor teaches the course

 1 Semester defines when it is offered

 1..\* TimeSlots represent when it meets

 0..\* Prerequisites required for registration

 0..\* Registration records link students to the course

**2.6 Prerequisite**

**Purpose:** Encapsulates course prerequisite information.

**Attributes**

 requiredCourseId : String — Course ID that must be completed first

**Relationships**

 Multiple prerequisites may exist per course.

**2.7 TimeSlot**

**Purpose:** Represents the scheduled meeting time(s) of a course.

**Attributes**

 day : String — Day of the week startTime : Time — Start time

 endTime : Time — End time

**Relationships**

 Aggregated by Course

 Used by Schedule for conflict checking

**2.8 Semester**

**Purpose:** Identifies the academic term.

**Attributes**

 name : String — e.g., “Fall 2025”

 startDate : Date

 endDate : Date

**Relationships**

 Each course belongs to exactly one semester

**Package: registration**

**2.9 Registration**

**Purpose:** Represents an attempt or completion of course registration.

**Attributes**

 registrationId : String — Unique registration ID

 studentId : String courseId : String

 timestamp : DateTime — When the request was made

 status : RegistrationStatus — Pending, Approved, Rejected

**Methods**

 confirm() : void — Sets status to APPROVED

 fail(reason:String) : void — Sets status to REJECTED

**Relationships**

 Links Student ↔ Course

 Triggers Notifications

 Logged in the system for auditing

**2.10 RegistrationStatus (Enum)**

 PENDING

 APPROVED

 REJECTED

**Package: schedule**

**2.11 Schedule**

**Purpose:** Holds all registered courses for a student.

**Attributes**

 studentId : String — The owner of this schedule**Methods**

 addCourse(course:Course) : void

 removeCourse(course:Course) : void

 checkConflict(timeslot:TimeSlot) : boolean — Ensures no overlap

**Relationships**

 Composition with Student (Schedule cannot exist without Student)

 Contains many TimeSlots

**Package: notification**

**2.12 Notification**

**Purpose:** Communicates registration outcomes and updates.

**Attributes**

 message : String

 type : String — success, failure, warning

 timeSent : DateTime — Time sent

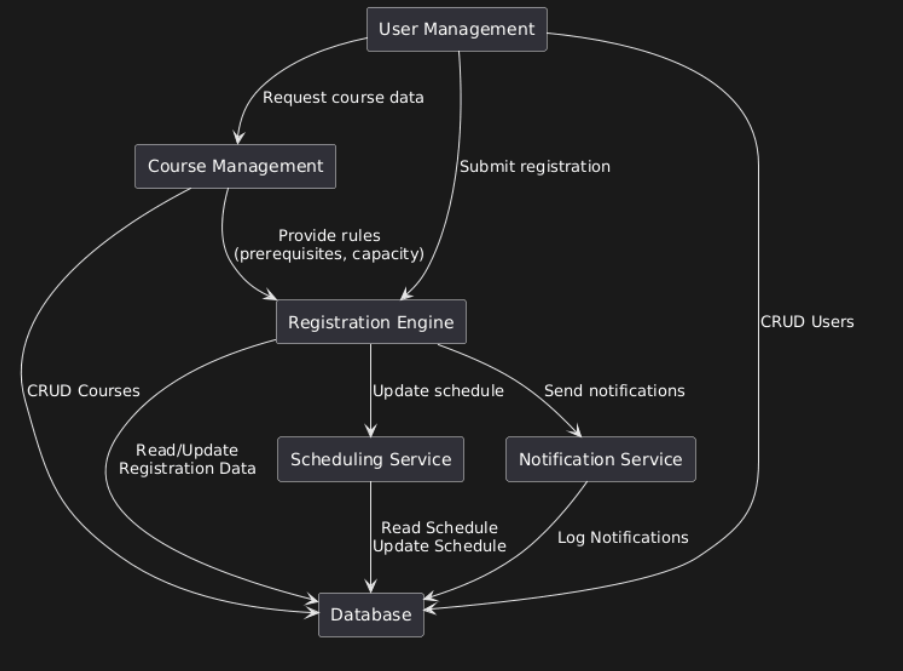
**Methods**

 send(toUserId:String) : void — Delivers the notification

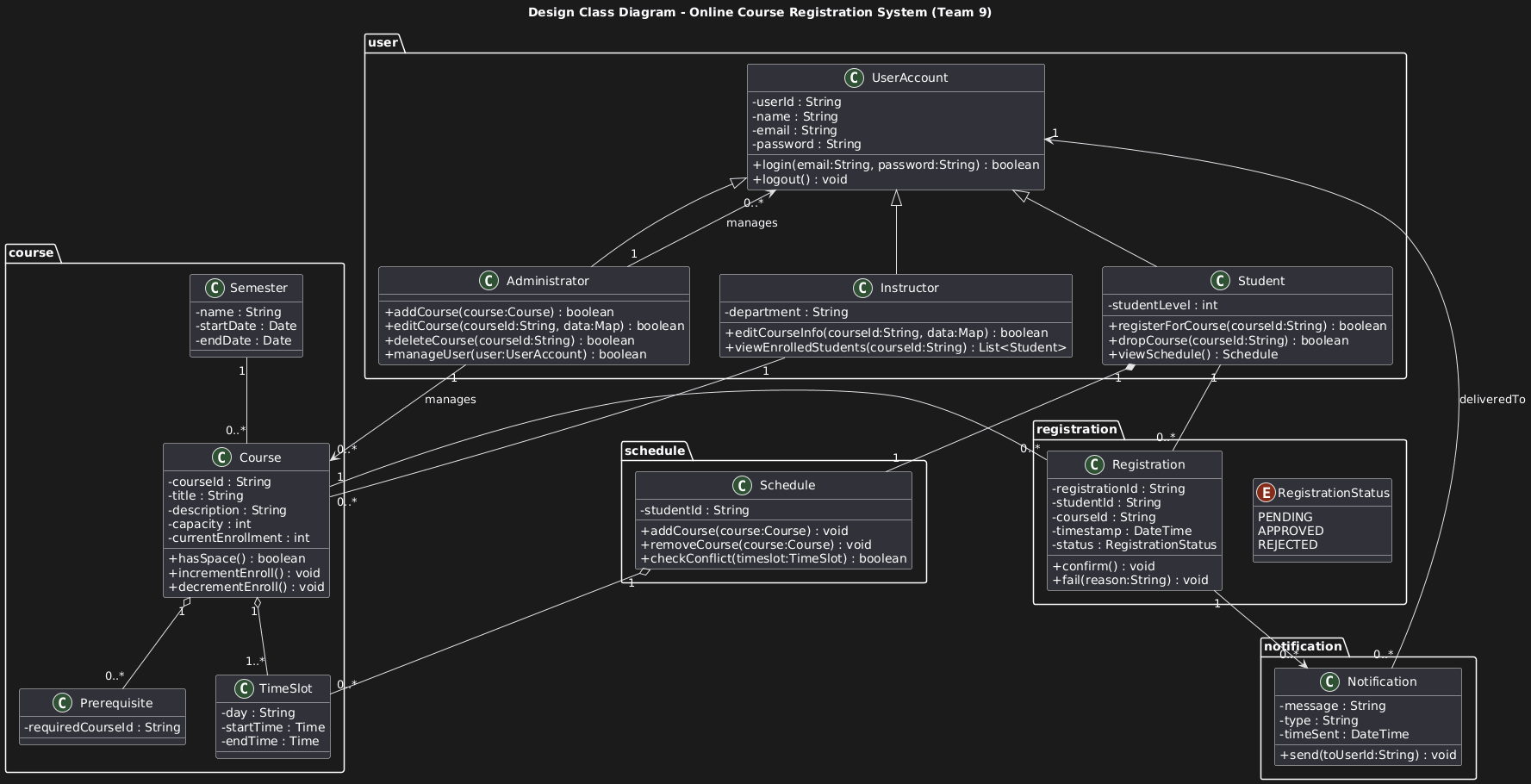
**Relationships**

 Created by Registration engine or Admin operations

 Delivered to Student, Instructor, or Admin

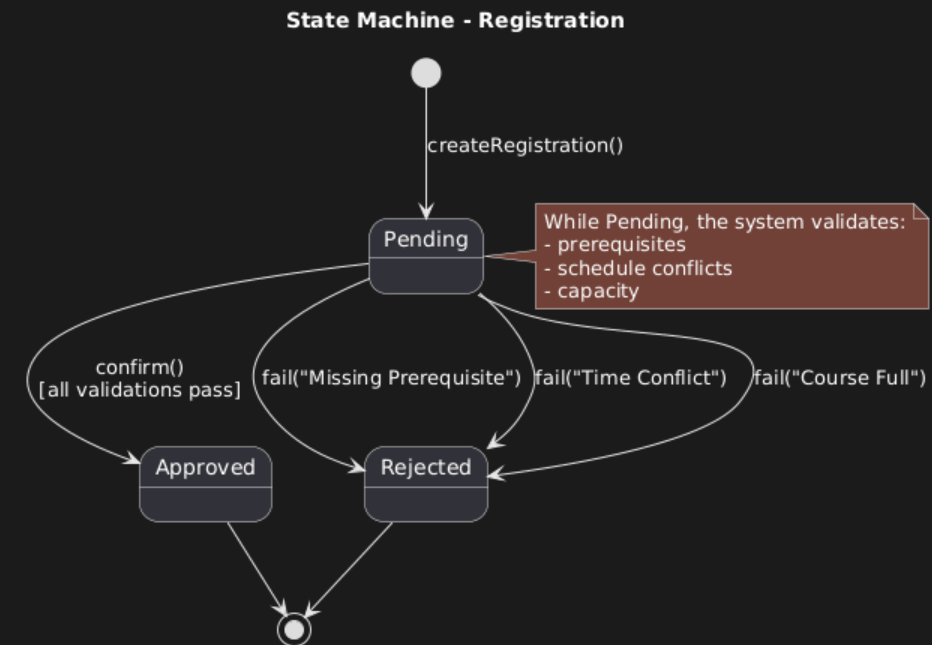
**Component Diagram****:**

**Class Design Diagram:**

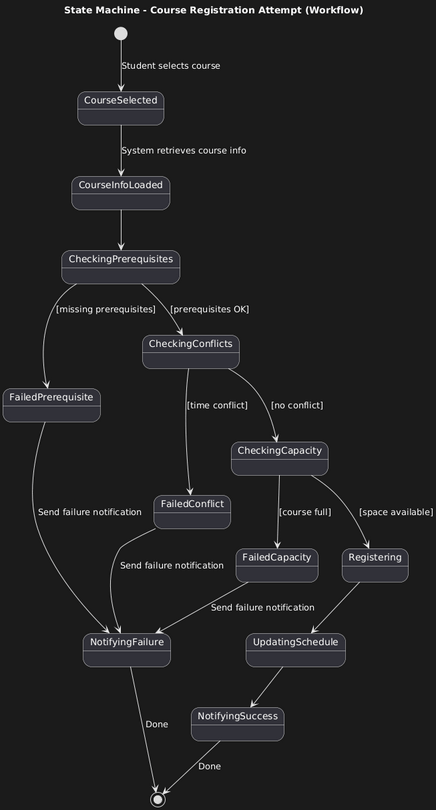
****

**State Machine Diagram:**

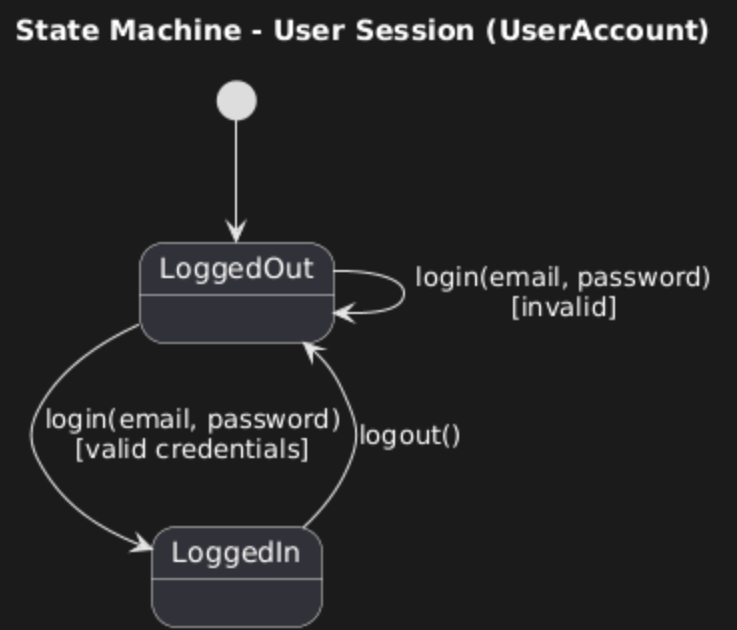
1. **Registration State Machine:**

****

1. **Course Registration Attempt (Process State Machine):**



1. **User Session State Machine:**

****