Online Course Registration System

SW301 - Course Project – Phase 2 - Analysis Model Report,

Ahmed Wael 202201415
Ahmed Sameh 202202151
Seif Eldin 202201510

Expanded use case descriptions

Use Case 1: Drop Course
Goal: Student removes enrollment for a Section.
Primary actor: Student
Supporting actors: Notification Service, Registrar (for late drops)
Preconditions:

- Student is authenticated.
- Drop period is open (or if late, a special workflow applies).

Postconditions:

- Registration status set to Dropped (or record removed depending on audit rules).
- If there is a waitlist, the first waitlist entry is promoted.
- Notification to student (and optionally instructor).

Normal Flow:

1. Student initiates drop for specific Section.
2. System validates drop window and rule checks.
3. System updates Registration → Dropped.
4. If waitlist exists, system enrolls first waitlist student and notifies them.
5. Notification & audit logged.

Alternate flows:

- A1: Drop window passed → system flags as late drop, create request for approval from Registrar/Instructor.
- A2: Financial/fee implications → system alerts student to financial consequences.

Use Case 2: Request Overload
Goal: Student requests Overload exception.
Primary actor: Student
Supporting actors: Instructor, Administrator, Notification Service
Preconditions:

- Student is authenticated and identifies specific reason.

Postconditions:

- Overload Request entity is created with status Pending.
- Instructor/Admin receives notification and can approve/deny.

Normal Flow:

1. Student opens override request form, provides justification and attachments.
2. System stores request and notifies Instructor.
3. Instructor reviews and either Approves or Denies.
    - If approved, system makes admin do the enrollment prosses and notify student.
    - If decline, notify student.

Alternate flows:

- A1: Instructor send to Admin.
- A2: If section full and approval given, admin may create temporary seat or move waitlist.

Use Case 3: Add New Course
Goal: Create a new Course and/or Section for a given semester.
Primary Actor: Administrator
Supporting Actors:

- Course Catalog Database
- Notification Service

Preconditions:

- Actor is authenticated with permission to create courses.
- Semester exists in the system.

Postconditions:

- A new Course record exists and/or a new Section under an existing Course.
- Instructor assigned to the section.

- Notification generated.

Normal Flow:

1. Admin opens “Create Course / Section” form.
2. Admin enters course details (title, credits, description, prerequisites).
3. Admin enters section details (instructor, room, time, capacity, term).
4. System validates input fields.
5. System checks whether similar course Code already exists.
    - If exists → skip to section creation.
6. System creates new Course in catalog.
7. System creates Section under Course for selected term.
8. System assigns instructor.
9. System sends success notification.
10. System logs creation event.

Alternate Flows

1. A2: Invalid Data
    - Missing prerequisites or invalid capacity → system returns error.
2. A3: Time Conflict with Instructor Schedule
    - System warns instructor and asks for confirmation.

Use Case 4: Manage User Accounts (Admin)
Goal: Administrator handles user lifecycle: create, update, disable accounts.
Primary Actor: Administrator
Supporting Actors: Authentication Service**,** Notification Service**,** User Database
Preconditions:

- Admin authenticated with permissions.
- System operational.

Postconditions:

- User account created/updated/disabled.
- Notification sent.

Normal Flow:

1. Admin opens User Management panel.
2. Admin selects action (Create / Update / Disable).

3. Admin enters user details:
    - name, email, role (Student, Instructor, Admin)
    - program (for students)
4. System validates fields (email uniqueness, role constraints).
5. System applies action:
    - Create user record
    - Update user data
    - Set status = Disabled
6. System sends notification email.
7. System logs admin action.

Alternate Flows:

1. A1: User Already Exists
    - System rejects creation and offers to update existing user.
2. A2: Invalid Role Assignment
    - System blocks attempt
3. A3: Disabled User Attempt to Login
    - Authentication service returns “Account Disabled”.

Analysis-level Class Diagram

User (abstract)

- attributes: userid, name, email, role, status
- operations: authenticate(), authorize()

Student (extends User)

- attributes: studentId, program, maxCredits
- operations: ViewTranscript(), RequestOverload(),RegisterCourse(), DropCourse(), ViewSchedule()

Instructor (extends User)

- attributes: instructorId, office, bio
- operations: ApproveRequest(), ViewCourseStudents()

Administrator (extends User)

- attributes: adminId, permissions
- operations: SetRegistrationWindow(), ManageAccounts()

Course

- attributes: courseCode, title, description, credits, prerequisites, department
- operations: IsPrerequisiteDone(student)

Section (CourseOffering)

- attributes: sectionId, term, scheduleTimes, location, capacity, enrolledCount, instructor
- operations: hasSeat(), enrollStudent(), dropStudent()

Semester (Term)

- attributes: termId, name, startDate, endDate, registrationWindow

Transcript

- attributes: student, completedCourses (Course + grade)

OverloadRequest

- attributes: requestId, student, reason, attachments, status, reviewer, timestamp
- operations: approve(), deny()

Schedule (student schedule)

- attributes: student, list<Section>

Notification

- attributes: notificationId, recipient(User), message, type, status
- operations: send()

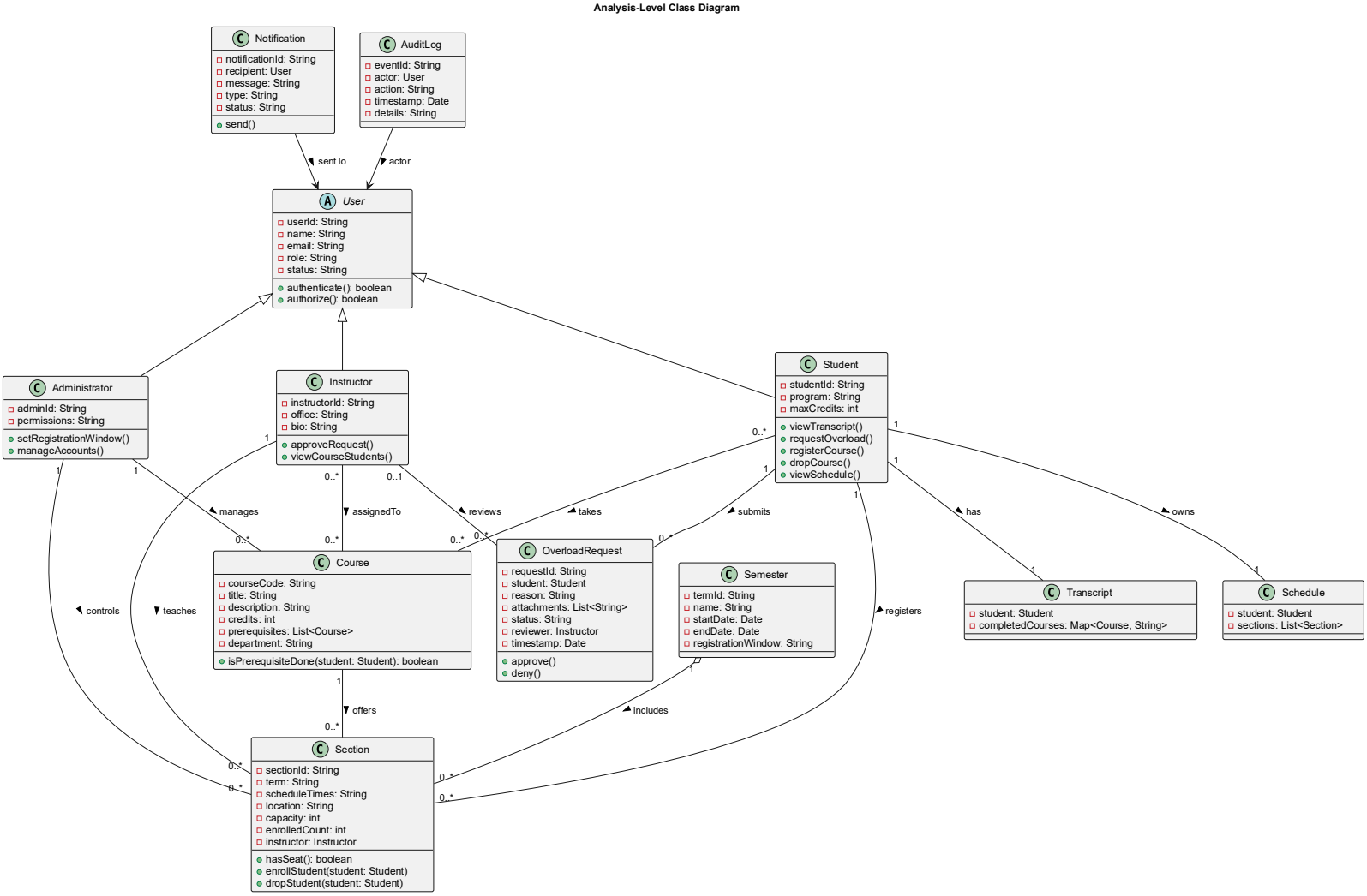
AuthenticationService

- operations: verifyCredentials(), require2FA()

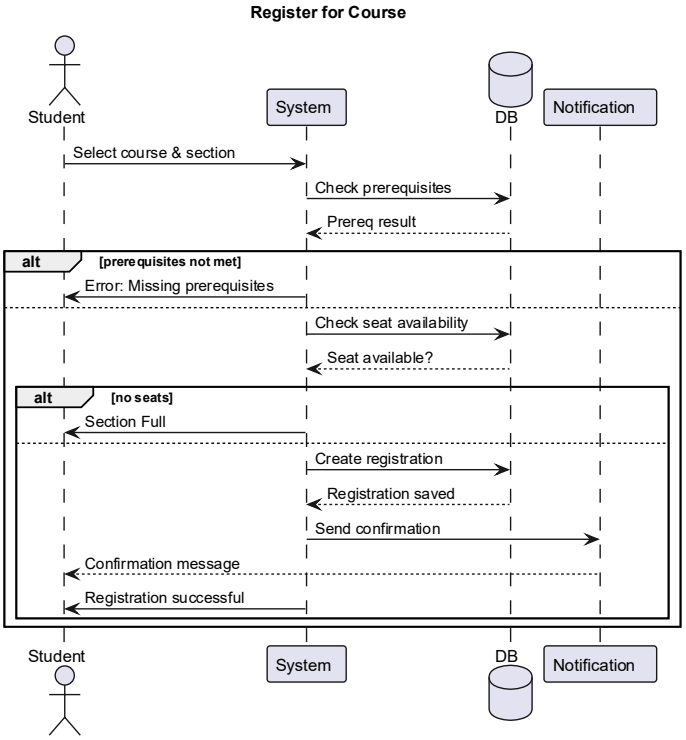
AuditLog

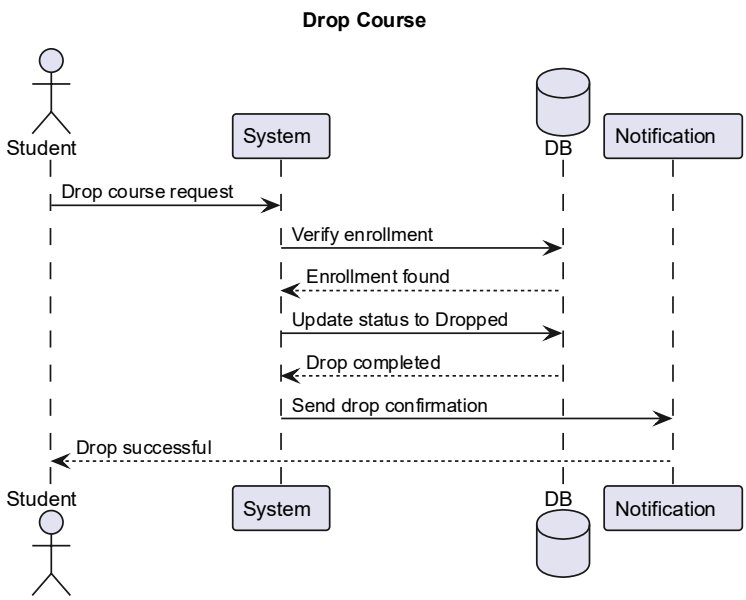
- attributes: eventId, actor, action, timestamp, details

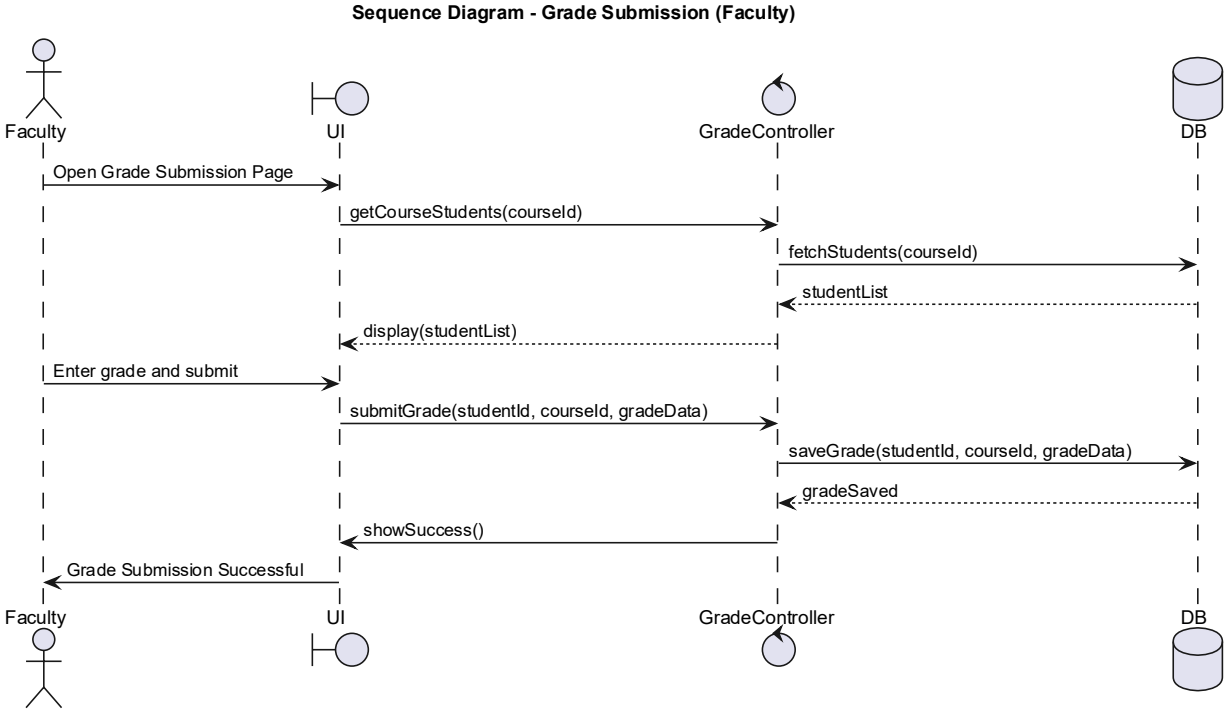
Analysis-Level Class Diagram



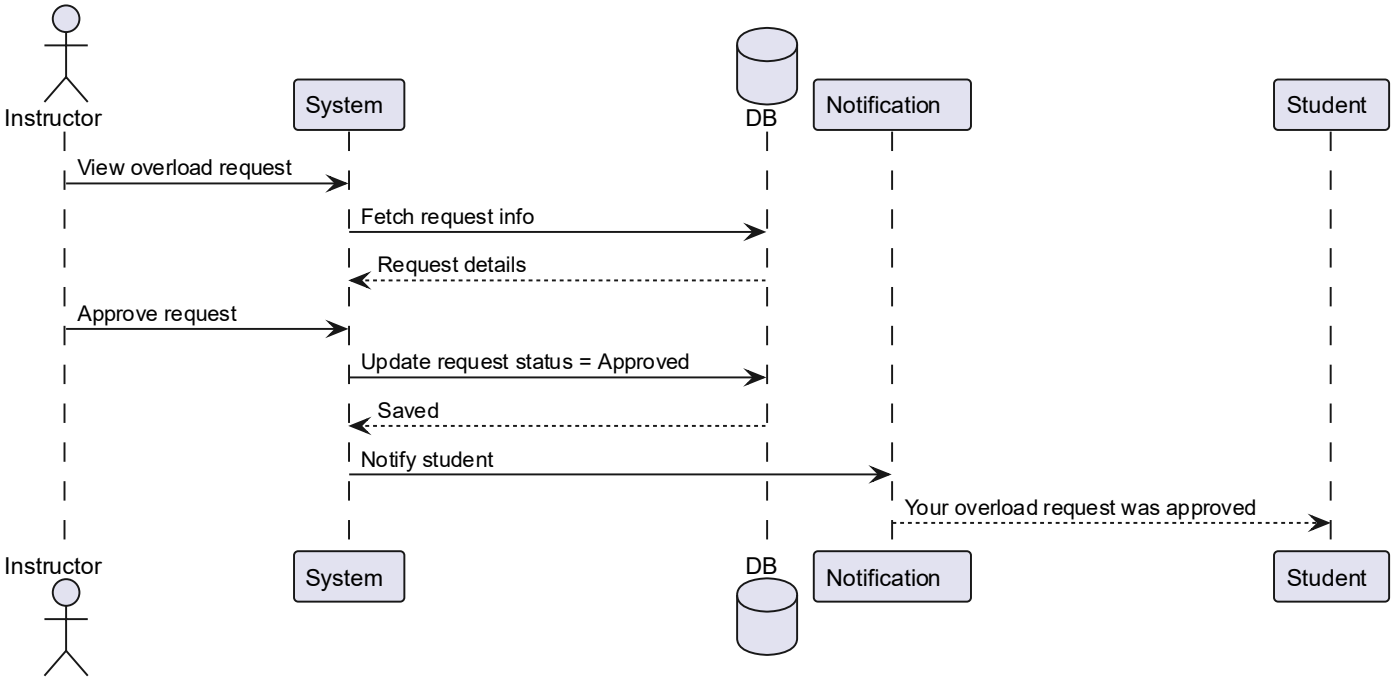
Sequence Diagrams



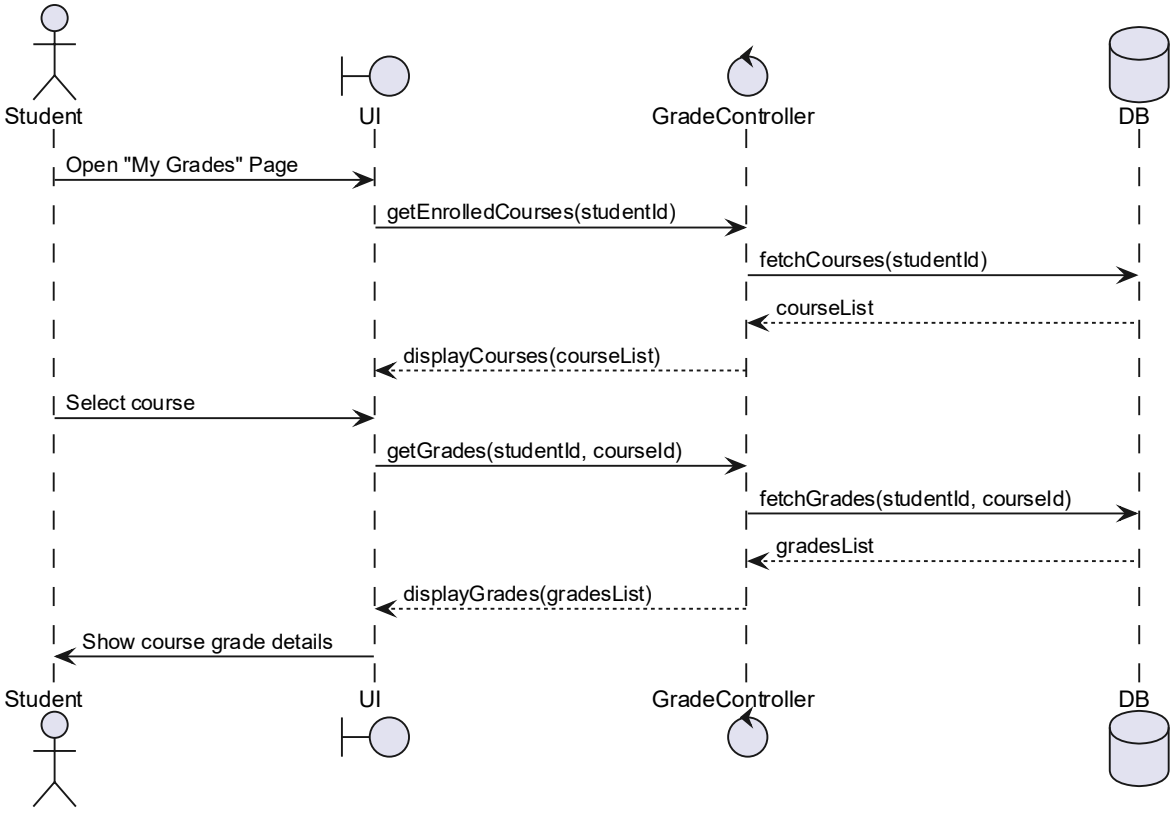




Approve Overload Request

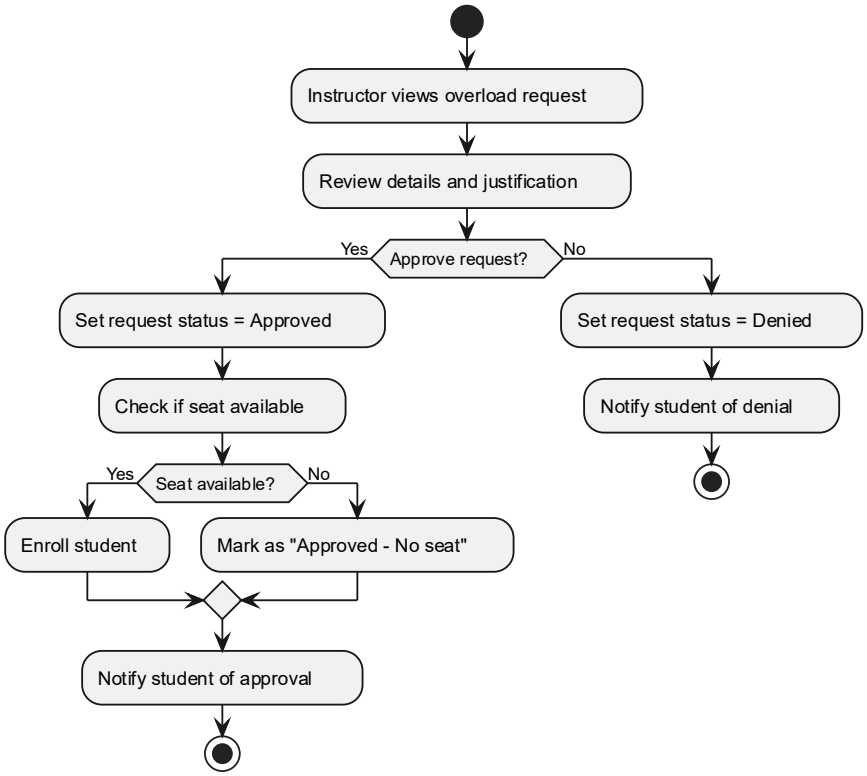


Sequence Diagram - Student Views Grades

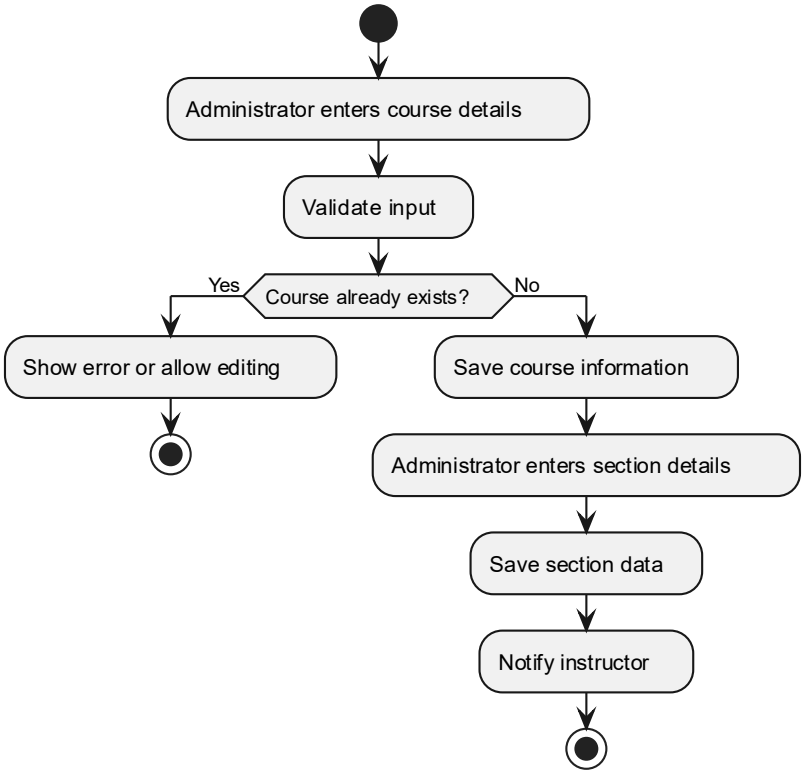


Activity Diagrams

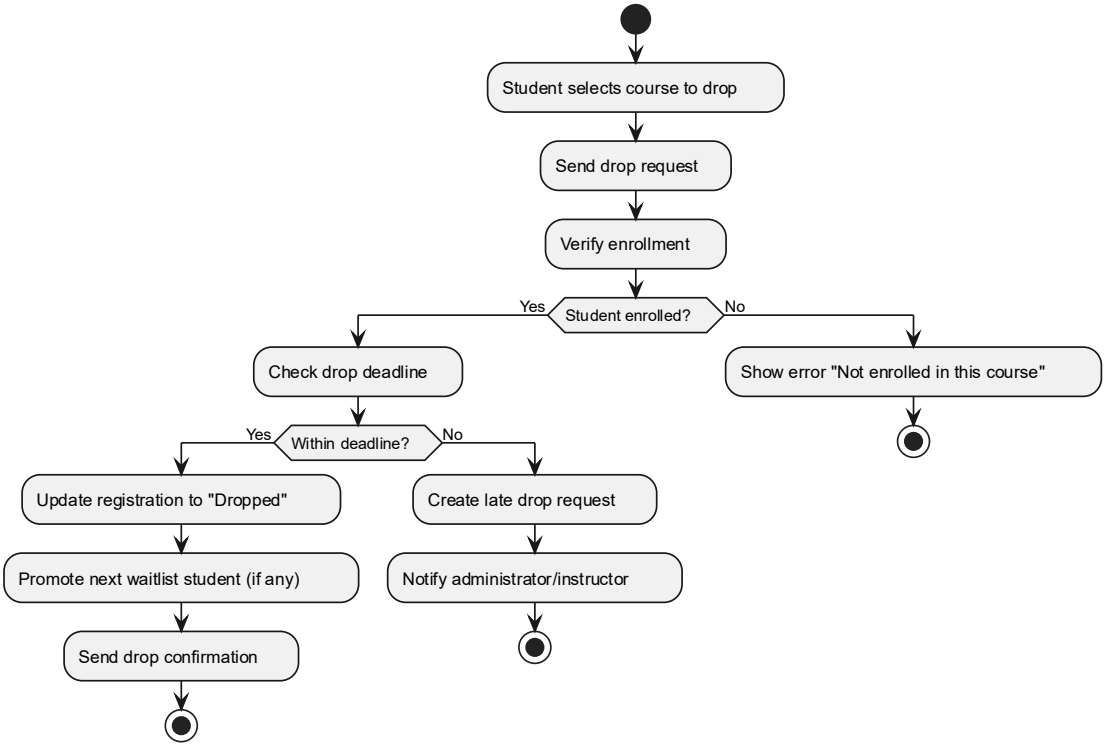
Activity Diagram - Approve Overload Request



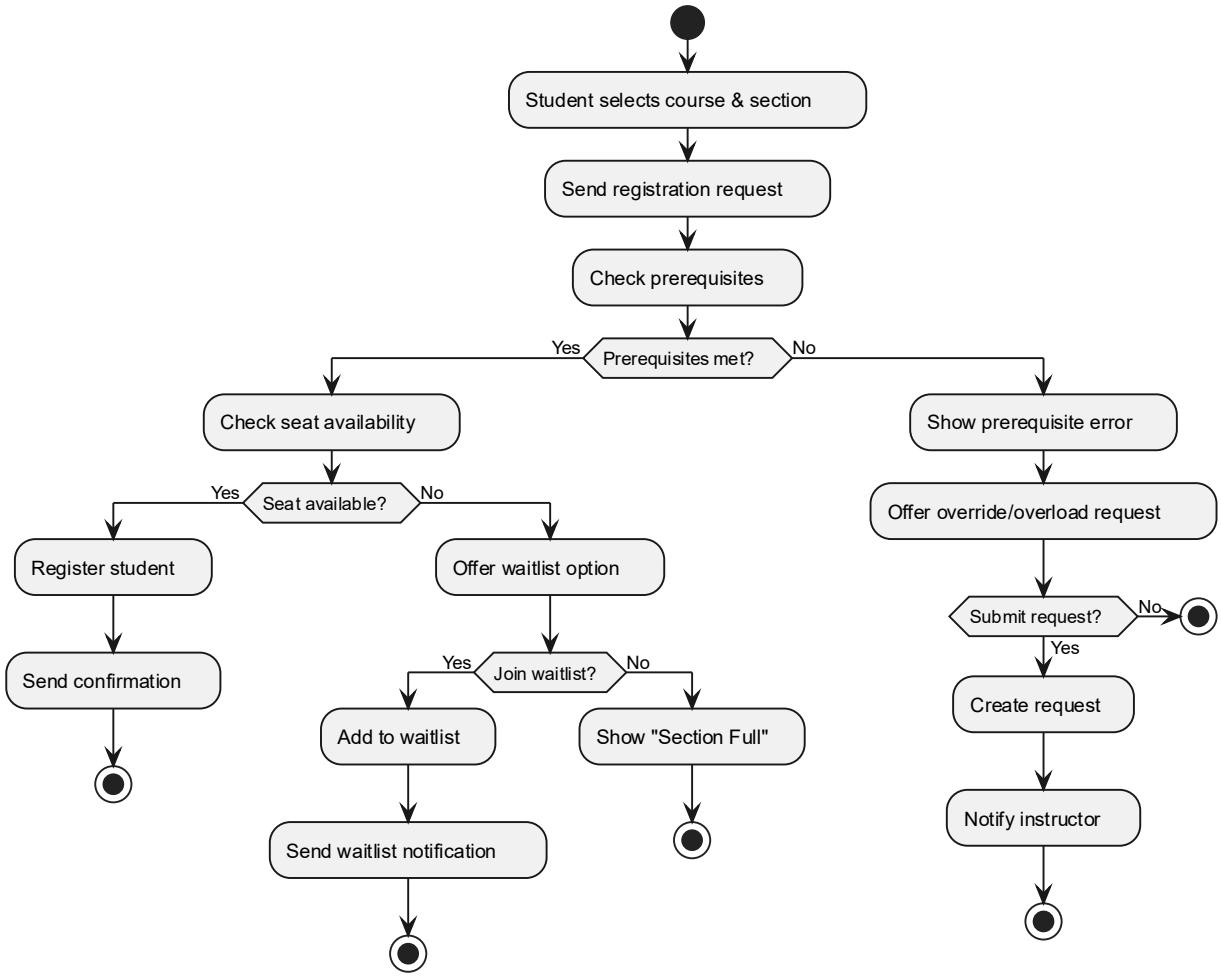
Activity Diagram - Create New Course



Activity Diagram - Drop Course



Activity Diagram - Register for Course



Explanatory notes on modeling decisions

Why separate Course and Section?
A Course is the abstract catalog entry (courseCode, credits, prereqs). Section (CourseOffering) holds term-specific data (time, instructor, capacity). This is fundamental to support multiple offerings per term and different instructors/times.

Why was the User class made abstract with Student, Instructor, and Administrator as subclasses?
Because all users share common information (userId, name, email, role) and actions (authenticate, authorize). Using an abstract parent class avoids repeating the same attributes and keeps the design cleaner.

Why do we use a Registration class instead of linking Student directly to Section?
Registration stores details like enrollment status, timestamp, and drop information. It also solves the many-to-many relationship between students and sections.

Why include Transcript and Schedule as separate classes?
Transcript stores completed courses and grades, which helps check prerequisites. Schedule stores the student's current sections, which helps detect time conflicts. This keeps Student simpler and avoids mixing too many responsibilities.

Why is Overload Request modeled as its own class?
Overload requests require details (reason, attachments, status) and a review/approval process. A separate class helps track the full workflow without complicating Registration.

Why link Student to Section instead of Student to Course?
Students register for sections, not courses. Course completion is tracked indirectly through sections.