Phase 2 — System Analysis & Modeling

Project: Online Course Registration System

---

1. Expanded Use Case Scenarios (Detailed)

UC01 — Draft Multi-Term Academic Plan

- Brief: A student builds a multi-term course plan (for future terms); the system checks academic progression rules and saves the plan as a scenario.
- Primary Actor: Student
- Secondary Actors: Degree Audit Engine, Catalog Service
- Preconditions: The student has declared a program/major, and the degree requirements mapped to the catalog are available.
- Postconditions: An Academic Plan is saved and linked to the student's profile; any validation warnings are shown.

Main Success Scenario:

1. The student opens the Academic Planning tool and selects target terms (e.g., Fall 2026 – Spring 2028).
2. The student drags/drops or selects courses into term-specific buckets.
3. For each added course, the system verifies that prerequisites will be met in earlier terms of the plan (optimistic validation) and flags any missing prerequisites.
4. The Degree Audit Engine assesses the overall plan for degree requirement coverage and total credits.
5. The student names and saves the plan; the system stores it as a scenario.
6. The system displays any warnings and suggests alternative courses where applicable.

Alternate Flows:

- 3a. The student marks a course as "Planned (external transfer)" — the system treats it as completed for planning purposes.

Exception Flows:

- 4a. Degree audit service is unavailable — the system saves the plan but marks the audit as pending.

UC02 — Search & Select Courses

- Brief: The student explores the course catalog, applies filters, reviews course sections, and adds selected sections to their Enrollment Cart.
- Primary Actor: Student
- Secondary Actor: System Catalog Service
- Preconditions: The current term's catalog is loaded; the student is authenticated (enrollment eligibility not required).
- Postconditions: The Enrollment Cart is updated with selected sections (or an error is displayed).

Main Success Scenario:

1. The student opens the Course Catalog page.
2. The system shows the default course list for the active term.
3. The student applies filters (e.g., department, time, delivery mode, instructor).
4. The system returns filtered results with available sections and seat availability.
5. The student views detailed information (prerequisites, credits, schedule) for a specific section.
6. The student clicks "Add to Cart" for a chosen section.
7. The system performs light validation (e.g., section existence, seat availability) and adds the section to the Enrollment Cart.
8. The system confirms the addition and displays the updated cart contents.

Alternate Flows:

- 3a. The student searches by course code — the system scrolls directly to that course.
- 6a. The student selects "View Similar Sections" and picks a different section to add.

Exception Flows:

- 4a. No results found — the system suggests broadening the filters.
- 7a. The section is full — the system offers a waitlist option.

UC03 — Execute Course Registration

- Brief: The student submits their Enrollment Cart for final registration; the system performs full validation and either enrolls the student, places them on a waitlist, or rejects the request.
- Primary Actor: Student
- Secondary Actors: Registration Engine, SIS (Student Information System), Notification Service
- Preconditions: The student is within their registration window, has no enrollment holds, and the Enrollment Cart contains at least one section.

- Postconditions: Enrollment status is updated per section (Enrolled, Waitlisted, or Failed); notifications are generated.

Main Success Scenario:

1. The student goes to the Enrollment Cart and clicks “Submit Registration”.
2. The system validates the student’s session and confirms the registration window is active.
3. For each section in the cart, the Registration Engine runs validations in this sequence:
   a. Checks prerequisites and co-requisites.
   b. Detects schedule conflicts with already-registered courses or other submitted sections.
   c. Validates credit/load limits and term-specific policies.
   d. Verifies section capacity.
4. If a section passes all checks, a seat is reserved and the student is marked “Enrolled”.
5. If a section fails only due to capacity, the student is offered waitlist placement.
6. If a section fails a hard rule (e.g., unmet prerequisite or active hold), the system logs the failure reason.
7. Once all sections are processed, the SIS is updated with final enrollment changes.
8. The system generates a consolidated result (success/failure per section) and notifies the student.

Alternate Flows:

- 3a. The student requests an override — the system forwards an exception request for instructor/admin approval (see UC12).
- 4a. Two students compete for the last seat — transaction locking ensures the first request succeeds; the second may be waitlisted.

Exception Flows:

- 2a. Session expired — the system prompts re-authentication and retries the submission.
- 7a. SIS update fails — the system rolls back local reservations and asks the student to retry; the event is logged.

UC04 — Withdraw from Course

- Primary Actor: Student
- Preconditions:
  - Student is authenticated.
  - Student is enrolled in at least one course.
  - The withdrawal period is open.
- Postconditions:
  - The course is removed from the student’s schedule.

    - SIS is updated.
    - The student receives confirmation.

Main Success Scenario:

1. The student navigates to the “My Schedule” page.
2. The system displays currently enrolled courses.
3. The student selects a course and clicks “Withdraw”.
4. The system checks if withdrawal is allowed per academic policy.
5. The system prompts the student to confirm.
6. The student confirms the action.
7. The enrollment record is updated to reflect withdrawal.
8. Tuition/financial adjustments are applied if needed.
9. The SIS is updated.
10. A withdrawal confirmation is sent to the student.

Alternate Flows:

- 4a. Withdrawal deadline has passed → system denies request and displays a message.
- 8a. No financial impact → skip step 8.

Exception Flows:

- SIS update fails → rollback withdrawal and notify the student.

UC05 — View Personal Schedule & Audit

- Primary Actor: Student
- Preconditions: Student is authenticated.
- Postconditions: Student sees the latest schedule and degree audit report.

Main Success Scenario:

1. The student accesses the “My Schedule & Audit” page.
2. The system retrieves registered courses for the current term.
3. The system fetches degree audit data.
4. The system displays a weekly schedule view alongside audit progress.

Alternate Flow:

- 3a. Audit service unavailable → system shows schedule only and warns the student.

Exception Flow:

- 2a. No courses found → display “Not enrolled this term”.

UC06 — Manage Course Section Data (Admin)

- Primary Actor: Administrator
- Preconditions: Admin is authenticated and has configuration privileges.
- Postconditions: A course section is added, updated, or deleted.

Main Success Scenario:

1. Admin opens “Course Section Management”.
2. The system displays existing sections.
3. Admin chooses to Add, Edit, or Delete a section.
4. Admin enters/updates section details (capacity, schedule, instructor, room).
5. The system validates the input.
6. The system saves the changes.
7. The catalog is updated, and relevant faculty are notified.

Alternate Flows:

- 3a. Admin edits an existing section.
- 3b. Admin deletes a section → system checks for enrolled students first.

Exception Flows:

- 5a. Validation error → system highlights invalid fields.
- 7a. Notification fails → log the event, but save the section anyway.

UC07 — Define Registration Period (Admin)

- Primary Actor: Administrator
- Preconditions: Admin is authenticated; the academic term exists.
- Postconditions: The registration window is saved and activated.

Main Success Scenario:

1. Admin opens “Registration Period Settings”.
2. The system lists existing terms.
3. Admin selects a term.
4. Admin enters start and end dates.
5. The system validates that start < end.
6. The system saves the registration window.
7. The window becomes active for student use.

Alternate Flow:

- 3a. Term not found → admin creates a new term first.

Exception Flow:

- 5a. Date overlaps with another period → system rejects the entry.

UC08 — Manage System User Roles (Admin)

- Primary Actor: Administrator
- Preconditions: Admin is authenticated.
- Postconditions: A user's role is updated (added, removed, or modified).

Main Success Scenario:

1. Admin goes to "User Role Management".
2. The system shows a list of users.
3. Admin selects a user.
4. Admin changes the role (Student, Faculty, or Admin).
5. The system validates role assignment policies.
6. The system saves the change.
7. The change is logged for audit purposes.

Exception Flows:

- 5a. Admin tries to downgrade their own role → system denies the action.
- 6a. Save operation fails → system displays an error.

UC09 — Generate System Report

- Primary Actor: Administrator
- Preconditions: Admin is authenticated; reporting service is available.
- Postconditions: A report is generated and available for download.

Main Success Scenario:

1. Admin opens the "Reports" module.
2. The system displays available report templates.
3. Admin selects a report type (e.g., enrollment summary, waitlist stats).
4. Admin configures filters (term, department, date range).
5. The system retrieves the necessary data.
6. The system compiles and generates the report.

7. A download link is provided.

Exception Flow:

- 5a. Data source unavailable → system shows an error message.

UC10 — Post Course Announcement (Faculty)

- Primary Actor: Faculty
- Preconditions: Faculty is authenticated and assigned to the course.
- Postconditions: Enrolled students receive the announcement.

Main Success Scenario:

1. Faculty opens the "Course Announcements" page.
2. The system lists courses assigned to the faculty.
3. Faculty selects a course.
4. Faculty enters announcement content.
5. The system validates that content is not empty.
6. The system posts the announcement.
7. All enrolled students are notified.

Exception Flow:

- 3a. Faculty selects a course they don't teach → system denies access.

UC11 — View Student Roster (Faculty)

- Primary Actor: Faculty
- Preconditions: Faculty is authenticated and assigned to the course section.
- Postconditions: Faculty sees the current student roster.

Main Success Scenario:

1. Faculty opens "Student Roster".
2. The system shows the faculty's assigned course sections.
3. Faculty selects a section.
4. The system retrieves enrolled students.
5. The system displays the roster with student details.

Exception Flow:

- 4a. No students enrolled → display an empty roster.

UC12 — Review Enrollment Exception Request (Faculty)

- Brief: Faculty evaluates override requests and approves or rejects waivers for capacity or prerequisites.
- Primary Actor: Faculty
- Secondary Actors: Registration Engine, Notification Service
- Preconditions: A student has submitted an active exception request for one of the faculty's courses.
- Postconditions: Request status is updated (Approved/Rejected); if approved, the student is notified and allowed to enroll.

Main Success Scenario:

1. Faculty opens the Exception Requests queue and selects a request.
2. The system displays student info, reason for request, and course details.
3. Faculty clicks Approve or Reject and may add comments.
4. The system updates the request status; if approved, it creates a temporary override enabling self-enrollment.
5. The student receives a notification.

Exception Flows:

- 3a. Faculty lacks permission → system denies action and escalates to an administrator.

---

2. Conceptual Classes (Analysis-Level)

The following core conceptual classes were derived from the requirements, each with defined responsibilities:

1. User (abstract)
    - Represents shared attributes of system users; base class for Student, Faculty, and Administrator.
    - Key conceptual attributes: userId, name, email, role
2. Student (extends User)
    - Manages academic plans, enrollment cart, registered courses, and holds.
    - Key conceptual attributes: studentId, program, completedCourses, currentPlans
3. Faculty (extends User)
    - Manages course sections, views rosters, and reviews exception requests.
4. Administrator (extends User)
    - Configures academic terms, manages course data, and sets system policies.

5. Course
    - Holds catalog-level metadata and maps to degree requirements.
    - Conceptual attributes: courseCode, title, creditHours, prerequisites
6. CourseSection
    - Represents a term-specific offering of a Course (section number, schedule, capacity, instructor).
    - Conceptual attributes: sectionId, term, capacity, schedule, instructor, enrolledCount
7. Semester / Term
    - Defines academic periods and associated registration windows.
8. Schedule / MeetingTime
    - Captures class meeting days/times/locations for conflict detection.
9. EnrollmentCart
    - Temporary container for a student's selected sections during a registration session.
10. Registration (Conceptual Transaction)
    - Represents a registration attempt for a student-section pair; includes status (ENROLLED, WAITLISTED, FAILED) and reasons.
11. AcademicPlan
    - Stores a multi-term sequence of planned courses for degree auditing and advising.
12. DegreeAuditEngine (conceptual service)
    - Validates academic plans against degree requirements.
13. ExceptionRequest
    - Records override requests (prerequisite or capacity) and tracks their lifecycle.
14. NotificationService (conceptual)
    - Generates and delivers notices for registration outcomes, approvals, and warnings.
15. SISAdapter (conceptual integration component)
    - Interfaces with the Student Information System to synchronize authoritative enrollment and student records.

---

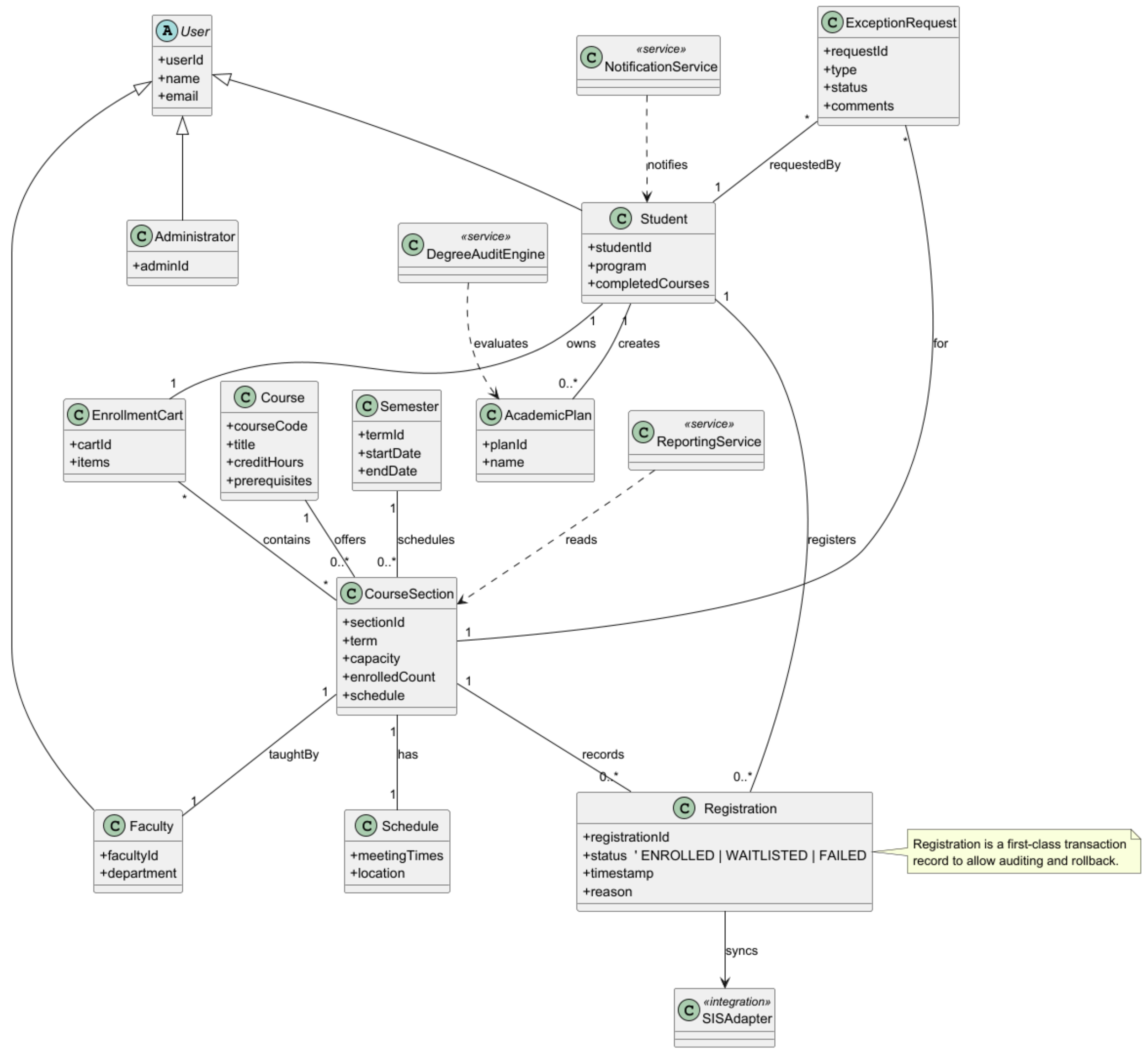
3. Relationships (Analysis-Level)

- Inheritance: User ← Student, Faculty, Administrator
- Aggregation: Course (1) — (0..*) CourseSection
  (A Course may have many CourseSection instances across terms.)
- CourseSection (1) — (*) Registration
  (Each section has multiple registration records.)
- Student (1) — (*) Registration
  (A student may have multiple registration records over time.)

- Student (1) — (1) EnrollmentCart
  (Each student has one active cart per session.)
- EnrollmentCart (*) — (*) CourseSection
  (Cart holds zero or more selected sections.)
- AcademicPlan (1) — (*) Course (or CourseSection reference)
  (Plan contains planned courses for future terms.)
- CourseSection (1) — (1) Faculty
  (Each section is taught by one primary instructor.)
- ExceptionRequest (*) — (1) CourseSection and (*) — (1) Student
  (Requests link a student to a specific section.)
- Dependency: DegreeAuditEngine uses AcademicPlan and Student.completedCourses
- Integration: SISAdapter synchronizes Registration outcomes with the SIS

---

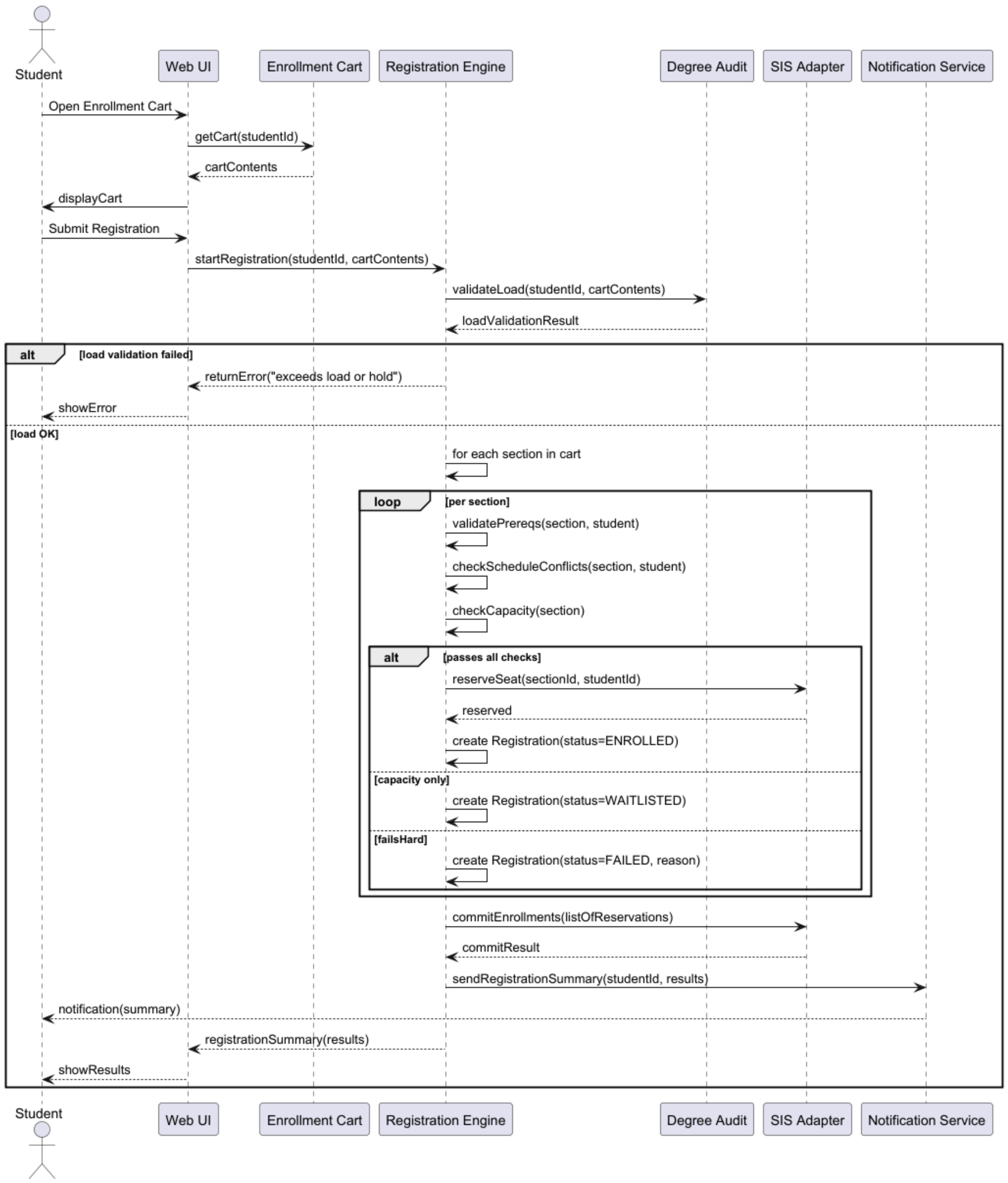
4. Analysis-Level Class Diagram (PlantUML)



```
@startuml
' Analysis-level (conceptual) class diagram for Online Course Registration System

skinparam classAttributeIconSize 0

' --- Conceptual classes ---
abstract class User {
 + userId
 + name
 + email
}

class Student {
 + studentId
 + program
 + completedCourses
}
class Faculty {
 + facultyId
 + department
}
class Administrator {
 + adminId
}
class Course {
 + courseCode
 + title
 + creditHours
 + prerequisites
}
class CourseSection {
 + sectionId
 + term
 + capacity
 + enrolledCount
 + schedule
}
class Semester {
 + termId
 + startDate
 + endDate
}
class Schedule {
 + meetingTimes
 + location
}
class EnrollmentCart {
 + cartId
 + items
}
class Registration {
 + registrationId
 + status  ' ENROLLED | WAITLISTED | FAILED
 + timestamp
 + reason
}
class AcademicPlan {
 + planId
 + name
}
class ExceptionRequest {
 + requestId
 + type
 + status
 + comments
}
class DegreeAuditEngine <<service>>
class SISAdapter <<integration>>
class NotificationService <<service>>
class ReportingService <<service>>

' --- Inheritance ---
User <|-- Student
User <|-- Faculty
User <|-- Administrator

' --- Associations & multiplicities ---
Course "1" -- "0..*" CourseSection : offers
Semester "1" -- "0..*" CourseSection : schedules
CourseSection "1" -- "1" Schedule : has
CourseSection "1" -- "0..*" Registration : records
Student "1" -- "0..*" Registration : registers
Student "1" -- "1" EnrollmentCart : owns
EnrollmentCart "*" -- "*" CourseSection : contains
Student "1" -- "0..*" AcademicPlan : creates
CourseSection "1" -- "1" Faculty : taughtBy
ExceptionRequest "*" -- "1" CourseSection : for
ExceptionRequest "*" -- "1" Student : requestedBy

' --- Dependencies / uses ---
Registration --> SISAdapter : syncs
DegreeAuditEngine ..> AcademicPlan : evaluates
NotificationService ..> Student : notifies
ReportingService ..> CourseSection : reads

' --- Notes ---
note right of Registration
 Registration is a first-class transaction
 record to allow auditing and rollback.
end note

@enduml
```

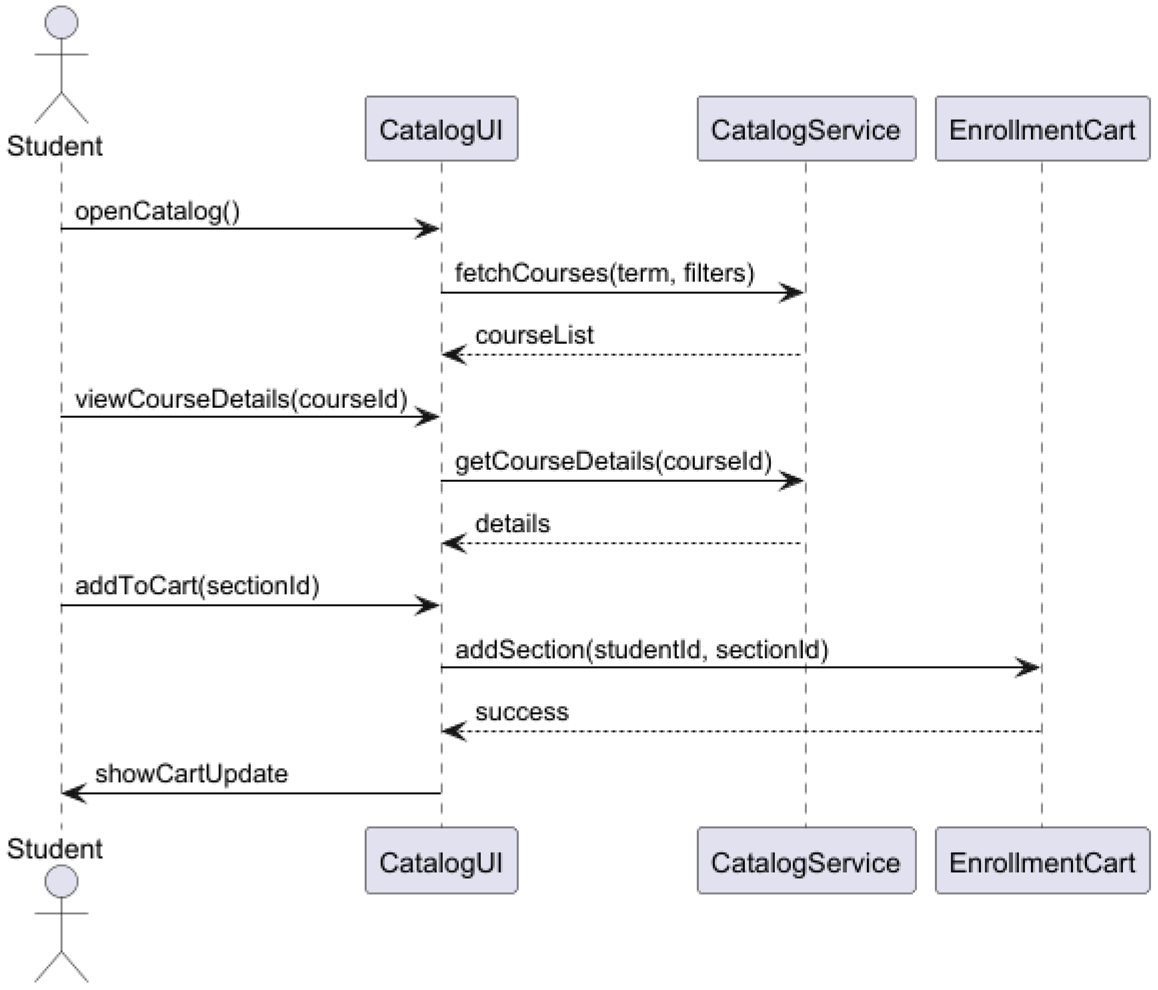
5. Sequence Diagrams (PlantUML) for core use cases

5.1 Sequence —Course Registration (UC03)


Student
Web UI
Enrollment Cart
Registration Engine
Degree Audit
SIS Adapter
Notification Service
Open Enrollment Cart
getCart(studentId)
cartContents
displayCart
Submit Registration
startRegistration(studentId, cartContents)
validateLoad(studentId, cartContents)
loadValidationResult
alt
[load validation failed]
returnError("exceeds load or hold")
showError
[load OK]
for each section in cart
loop
[per section]
validatePrereqs(section, student)
checkScheduleConflicts(section, student)
checkCapacity(section)
alt
[passes all checks]
reserveSeat(sectionId, studentId)
reserved
create Registration(status=ENROLLED)
[capacity only]
create Registration(status=WAITLISTED)
[failsHard]
create Registration(status=FAILED, reason)
commitEnrollments(listOfReservations)
commitResult
sendRegistrationSummary(studentId, results)
notification(summary)
registrationSummary(results)
showResults

```
@startuml
actor Student
participant "Web UI" as UI
participant "Enrollment Cart" as Cart
participant "Registration Engine" as Engine
participant "Degree Audit" as Audit
participant "SIS Adapter" as SIS
participant "Notification Service" as Notify

Student -> UI : Open Enrollment Cart
UI -> Cart : getCart(studentId)
Cart --> UI : cartContents
UI -> Student : displayCart

Student -> UI : Submit Registration
UI -> Engine : startRegistration(studentId, cartContents)
Engine -> Audit : validateLoad(studentId, cartContents)
Audit --> Engine : loadValidationResult

alt load validation failed
 Engine --> UI : returnError("exceeds load or hold")
 UI --> Student : showError
else load OK
 Engine -> Engine : for each section in cart
 loop per section
   Engine -> Engine : validatePrereqs(section, student)
   Engine -> Engine : checkScheduleConflicts(section, student)
   Engine -> Engine : checkCapacity(section)
   alt passes all checks
     Engine -> SIS : reserveSeat(sectionId, studentId)
     SIS --> Engine : reserved
     Engine -> Engine : create Registration(status=ENROLLED)
   else capacity only
     Engine -> Engine : create Registration(status=WAITLISTED)
   else failsHard
     Engine -> Engine : create Registration(status=FAILED, reason)
   end
 end
 Engine -> SIS : commitEnrollments(listOfReservations)
 SIS --> Engine : commitResult
 Engine -> Notify : sendRegistrationSummary(studentId, results)
 Notify --> Student : notification(summary)
 Engine --> UI : registrationSummary(results)
 UI --> Student : showResults
end
@enduml
```

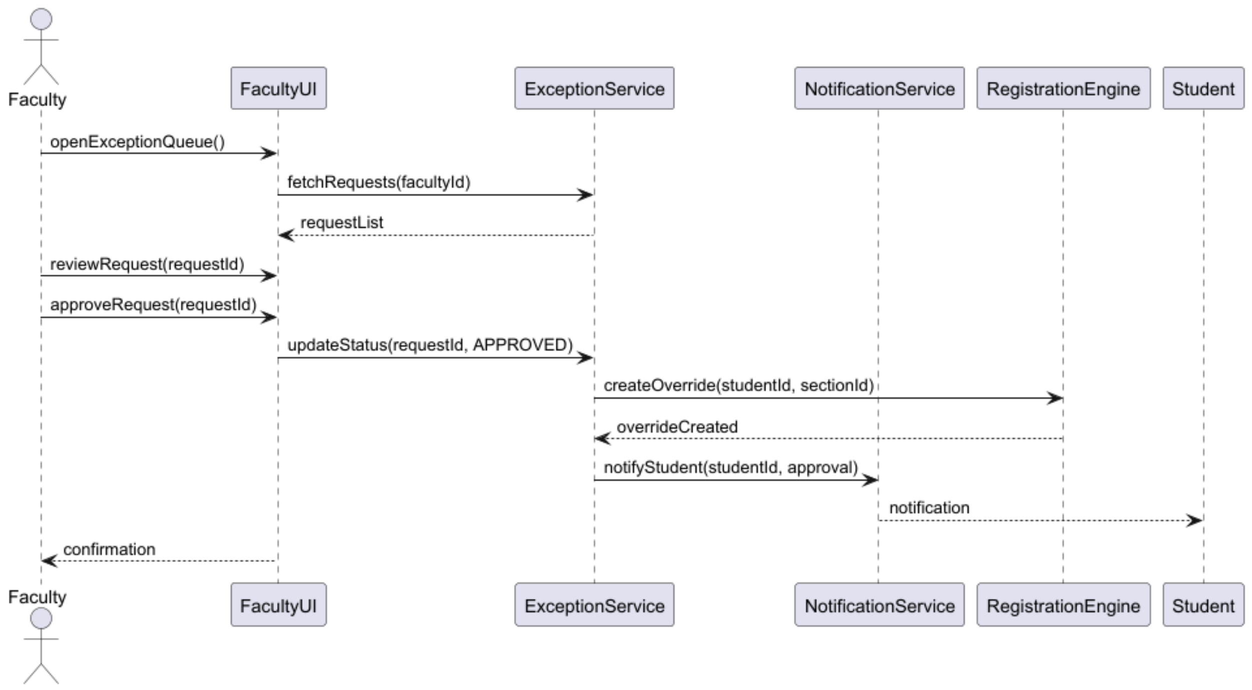
5.2 Sequence — Search & Select Courses (UC02)



```
@startuml
actor Student
participant CatalogUI
participant CatalogService
participant EnrollmentCart

Student -> CatalogUI: openCatalog()
CatalogUI -> CatalogService: fetchCourses(term, filters)
CatalogService --> CatalogUI: courseList
Student -> CatalogUI: viewCourseDetails(courseId)
CatalogUI -> CatalogService: getCourseDetails(courseId)
CatalogService --> CatalogUI: details
Student -> CatalogUI: addToCart(sectionId)
CatalogUI -> EnrollmentCart: addSection(studentId, sectionId)
EnrollmentCart --> CatalogUI: success
CatalogUI -> Student: showCartUpdate
```

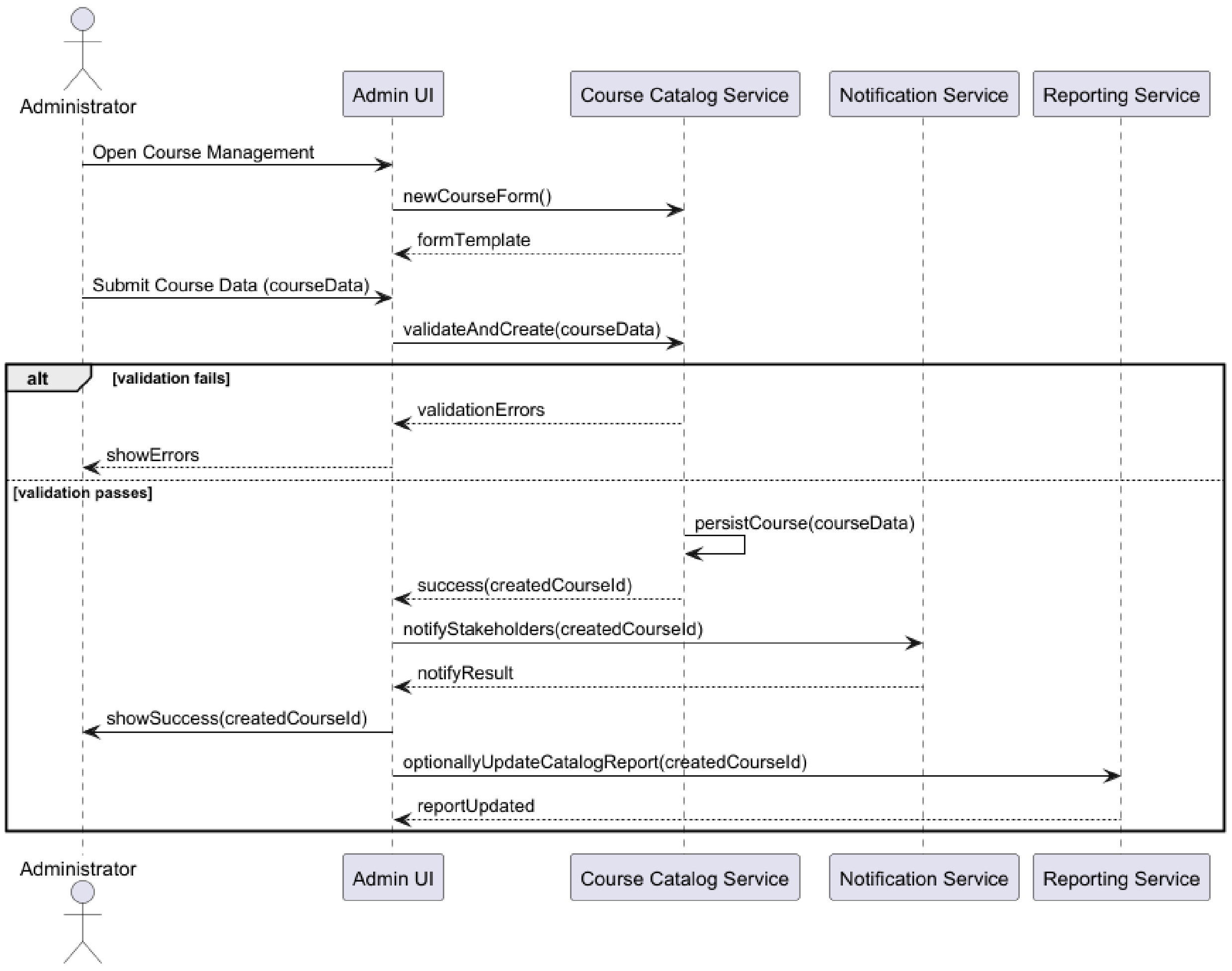
```
@enduml
```

5.3 Sequence — Review Enrollment Exception Request (UC12)



```
@startuml
actor Faculty
participant FacultyUI
participant ExceptionService
participant NotificationService
participant RegistrationEngine

Faculty -> FacultyUI: openExceptionQueue()
FacultyUI -> ExceptionService: fetchRequests(facultyId)
ExceptionService --> FacultyUI: requestList
Faculty -> FacultyUI: reviewRequest(requestId)
Faculty -> FacultyUI: approveRequest(requestId)
FacultyUI -> ExceptionService: updateStatus(requestId, APPROVED)
ExceptionService -> RegistrationEngine: createOverride(studentId, sectionId)
RegistrationEngine --> ExceptionService: overrideCreated
ExceptionService -> NotificationService: notifyStudent(studentId, approval)
NotificationService --> Student: notification
FacultyUI --> Faculty: confirmation
@enduml
```

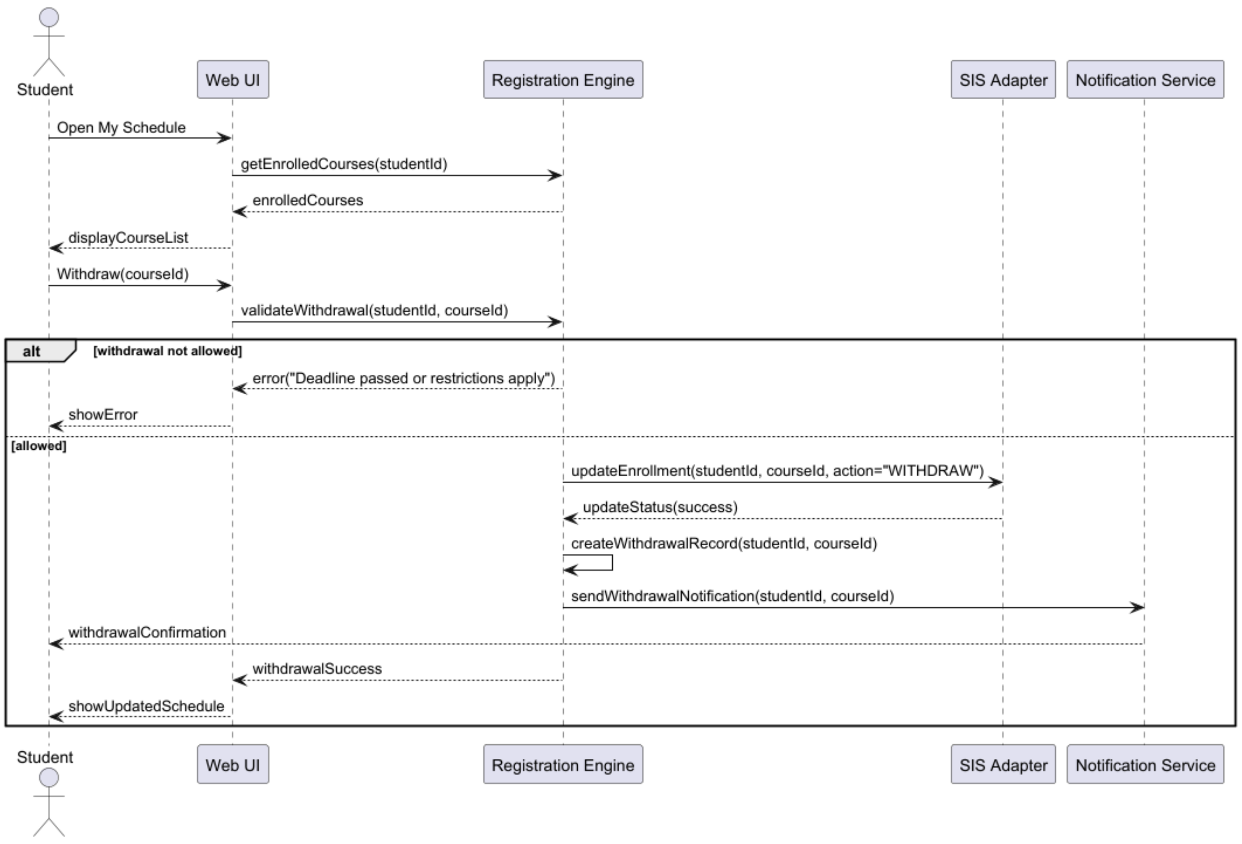
5.4 Sequence — Add New Course (Admin)



```
@startuml
actor Administrator
participant "Admin UI" as AdminUI
participant "Course Catalog Service" as Catalog
participant "Notification Service" as Notify
participant "Reporting Service" as Reports

Administrator -> AdminUI : Open Course Management
AdminUI -> Catalog : newCourseForm()
Catalog --> AdminUI : formTemplate

Administrator -> AdminUI : Submit Course Data (courseData)
AdminUI -> Catalog : validateAndCreate(courseData)
alt validation fails
```

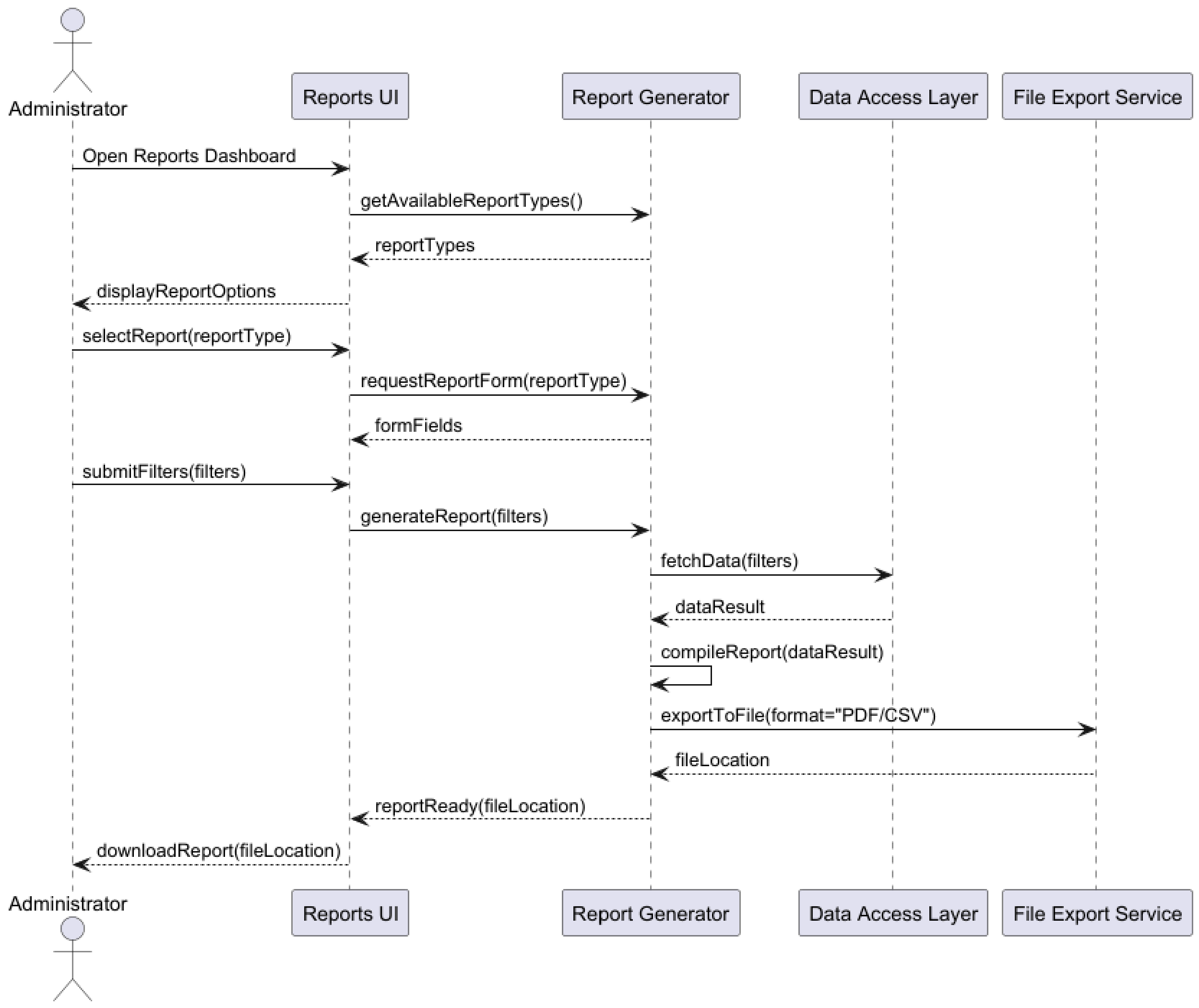
```
 Catalog --> AdminUI : validationErrors
 AdminUI --> Administrator : showErrors
else validation passes
 Catalog -> Catalog : persistCourse(courseData)
 Catalog --> AdminUI : success(createdCourseId)
 AdminUI -> Notify : notifyStakeholders(createdCourseId)
 Notify --> AdminUI : notifyResult
 AdminUI -> Administrator : showSuccess(createdCourseId)
 AdminUI -> Reports : optionallyUpdateCatalogReport(createdCourseId)
 Reports --> AdminUI : reportUpdated
end
@enduml
```

5.5 Sequence — Withdraw From Course:


Student
Web UI
Registration Engine
SIS Adapter
Notification Service
Open My Schedule
getEnrolledCourses(studentId)
enrolledCourses
displayCourseList
Withdraw(courseId)
validateWithdrawal(studentId, courseId)
alt [withdrawal not allowed]
error("Deadline passed or restrictions apply")
showError
[allowed]
updateEnrollment(studentId, courseId, action="WITHDRAW")
updateStatus(success)
createWithdrawalRecord(studentId, courseId)
sendWithdrawalNotification(studentId, courseId)
withdrawalConfirmation
withdrawalSuccess
showUpdatedSchedule

```
@startuml
actor Student
participant "Web UI" as UI
participant "Registration Engine" as Engine
participant "SIS Adapter" as SIS
participant "Notification Service" as Notify

Student -> UI : Open My Schedule
UI -> Engine : getEnrolledCourses(studentId)
Engine --> UI : enrolledCourses
UI --> Student : displayCourseList

Student -> UI : Withdraw(courseId)
UI -> Engine : validateWithdrawal(studentId, courseId)

alt withdrawal not allowed
  Engine --> UI : error("Deadline passed or restrictions apply")
  UI --> Student : showError
else allowed
  Engine -> SIS : updateEnrollment(studentId, courseId, action="WITHDRAW")
  SIS --> Engine : updateStatus(success)

  Engine -> Engine : createWithdrawalRecord(studentId, courseId)
  Engine -> Notify : sendWithdrawalNotification(studentId, courseId)
  Notify --> Student : withdrawalConfirmation
  Engine --> UI : withdrawalSuccess
  UI --> Student : showUpdatedSchedule
end

@enduml
```

5.6 Sequence Diagram — Generate System Report

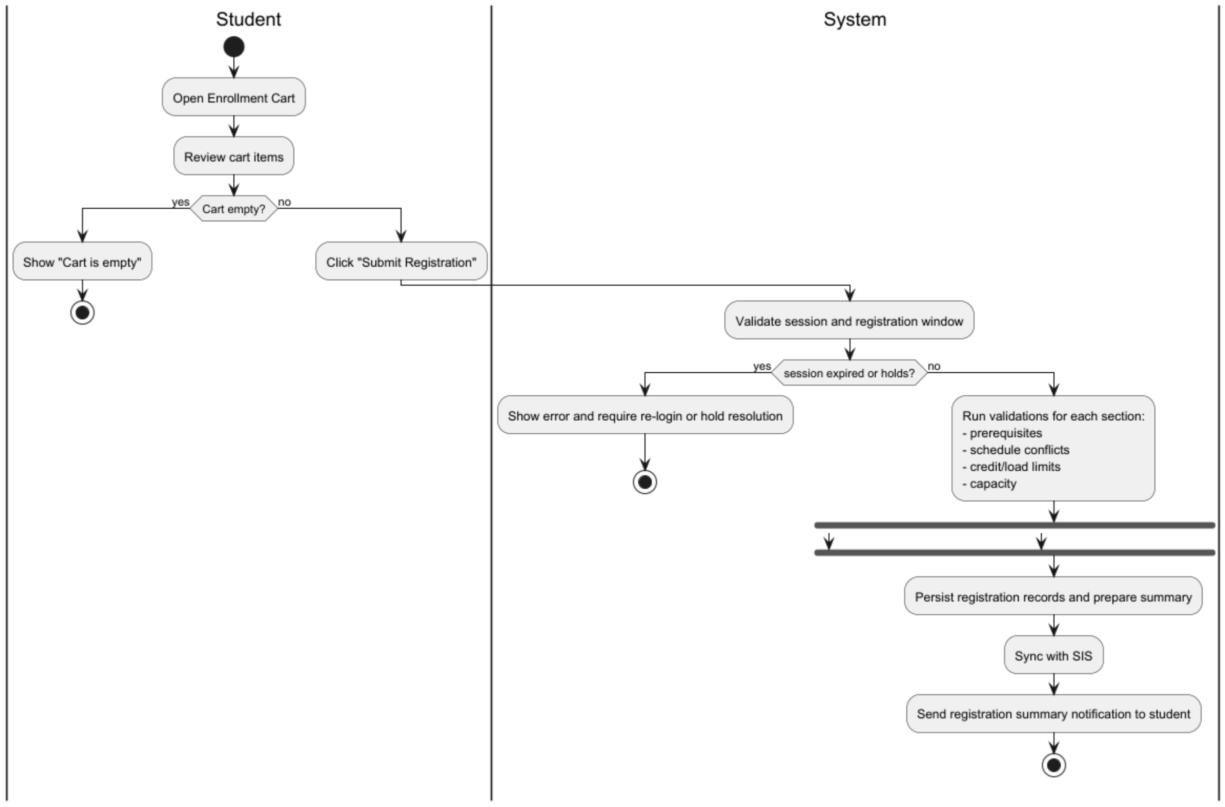


```
@startuml
actor Administrator
participant "Reports UI" as UI
participant "Report Generator" as ReportGen
participant "Data Access Layer" as DAL
participant "File Export Service" as Exporter

Administrator -> UI : Open Reports Dashboard
UI -> ReportGen : getAvailableReportTypes()
ReportGen --> UI : reportTypes
```

```
UI --> Administrator : displayReportOptions

Administrator -> UI : selectReport(reportType)
UI -> ReportGen : requestReportForm(reportType)
ReportGen --> UI : formFields

Administrator -> UI : submitFilters(filters)
UI -> ReportGen : generateReport(filters)

ReportGen -> DAL : fetchData(filters)
DAL --> ReportGen : dataResult

ReportGen -> ReportGen : compileReport(dataResult)
ReportGen -> Exporter : exportToFile(format="PDF/CSV")
Exporter --> ReportGen : fileLocation

ReportGen --> UI : reportReady(fileLocation)
UI --> Administrator : downloadReport(fileLocation)

@enduml
```

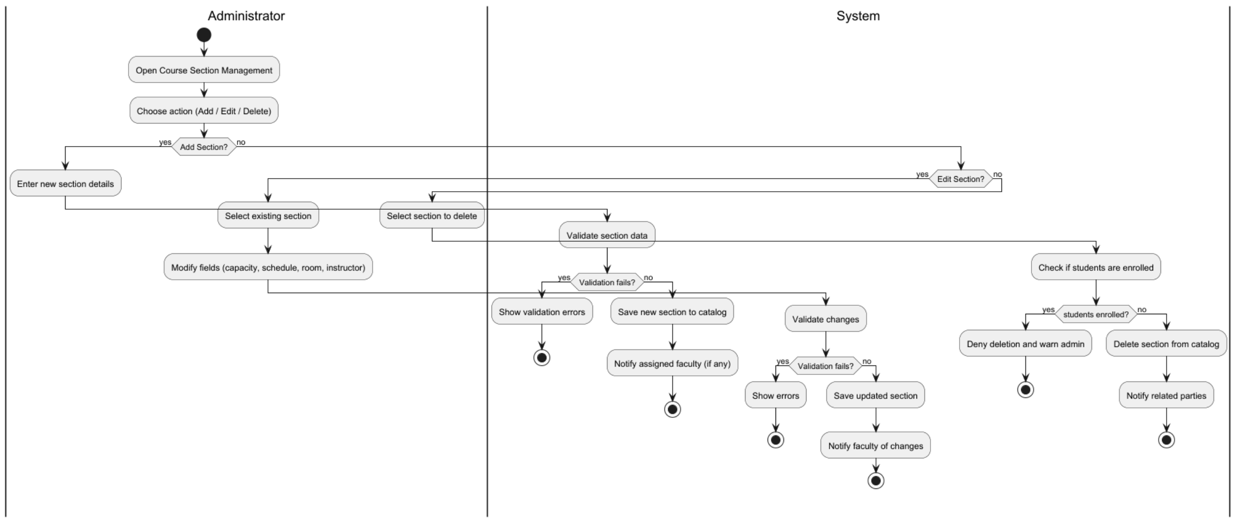
6. Activity Diagrams (PlantUML)

6.1 Activity — Course Registration Process


Student
System
Open Enrollment Cart
Review cart items
Cart empty?
yes
no
Show "Cart is empty"
Click "Submit Registration"
Validate session and registration window
session expired or holds?
yes
no
Show error and require re-login or hold resolution
Run validations for each section:
- prerequisites
- schedule conflicts
- credit/load limits
- capacity
Persist registration records and prepare summary
Sync with SIS
Send registration summary notification to student

```
@startuml
|Student|
start
:Open Enrollment Cart;
:Review cart items;

if (Cart empty?) then (yes)
 :Show "Cart is empty";
 stop
else (no)
 :Click "Submit Registration";
 |System|
 :Validate session and registration window;

 if (session expired or holds?) then (yes)
   :Show error and require re-login or hold resolution;
   stop
 else (no)
   :Run validations for each section:\n- prerequisites\n- schedule conflicts\n- credit/load limits\n- capacity;

   fork
     -> System: enrollBranch
     :Enroll sections that pass all checks;
   fork again
     -> System: waitlistBranch
     :Place sections with capacity issues on waitlist;
   end fork

   :Persist registration records and prepare summary;
   :Sync with SIS;
   :Send registration summary notification to student;
   stop
 endif

endif
@enduml
```

6.2 Activity — Manage Course Section Data (Admin)


Administrator
System
Open Course Section Management
Choose action (Add / Edit / Delete)
yes
Add Section?
no
Enter new section details
yes
Edit Section?
no
Select existing section
Select section to delete
Validate section data
Modify fields (capacity, schedule, room, instructor)
Check if students are enrolled
yes
Validation fails?
no
Show validation errors
Save new section to catalog
Validate changes
yes
students enrolled?
no
Deny deletion and warn admin
Delete section from catalog
Notify assigned faculty (if any)
yes
Validation fails?
no
Show errors
Save updated section
Notify related parties
Notify faculty of changes

```
@startuml
|Administrator|
start
:Open Course Section Management;
:Choose action (Add / Edit / Delete);
```

```
if (Add Section?) then (yes)
 :Enter new section details;
 |System|
 :Validate section data;
 if (Validation fails?) then (yes)
   :Show validation errors;
   stop
 else (no)
   :Save new section to catalog;
   :Notify assigned faculty (if any);
   stop
 endif

else (no)
 if (Edit Section?) then (yes)
   |Administrator|
   :Select existing section;
   :Modify fields (capacity, schedule, room, instructor);
   |System|
   :Validate changes;
   if (Validation fails?) then (yes)
     :Show errors;
     stop
   else (no)
     :Save updated section;
     :Notify faculty of changes;
     stop
   endif

 else (no)
   |Administrator|
   :Select section to delete;
   |System|
   :Check if students are enrolled;
   if (students enrolled?) then (yes)
     :Deny deletion and warn admin;
     stop
   else (no)
     :Delete section from catalog;
     :Notify related parties;
     stop
   endif

 endif
endif

@enduml
```

7. Explanatory Notes & Modeling Decisions

Analysis vs. Design:
This phase emphasizes *what* the system does and the core domain concepts involved — not *how* it is implemented. Therefore, concrete data types, method signatures, and implementation details are intentionally omitted. These will be introduced during Phase 3 (Design Stage).

Registration as a First-Class Concept:
Instead of handling enrollment as a simple field or boolean status, **Registration** is modeled as a transactional entity. This supports:

- Historical tracking
- Auditing
- Waitlist management
- Rollback of failed attempts

Separation of Catalog and Sections:

- **Course** stores catalog-level details (title, prerequisites, credit hours).
- **CourseSection** represents term-specific offerings (schedule, capacity, instructor).

This enables catalog reuse across terms and improves flexibility in academic scheduling.

Exception Requests Workflow:
ExceptionRequest is modeled as its own entity linking a student and a course section while capturing:

- Reason for request
- Approval lifecycle

- Status tracking
- Audit logs

SIS as an Integration Adapter:
The **SISAdapter** acts as a boundary component to keep internal logic decoupled from the external Student Information System. SIS remains the authoritative record while controlled synchronization updates registration outcomes.

Degree Audit as a Service:
The **DegreeAuditEngine** encapsulates complex rules for degree progress evaluation. It is invoked during academic planning and select registration checks, preventing business rules from being scattered across domain entities.