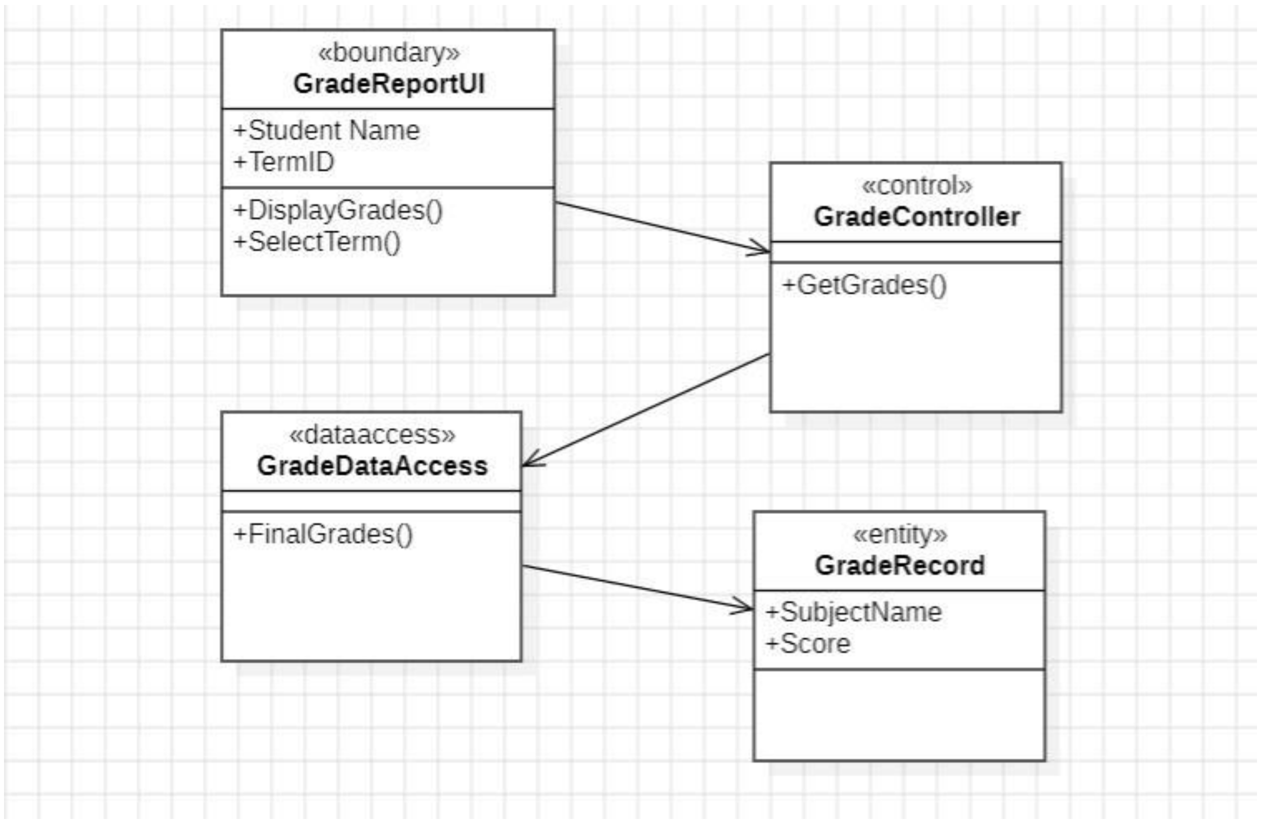
Object-Oriented Design

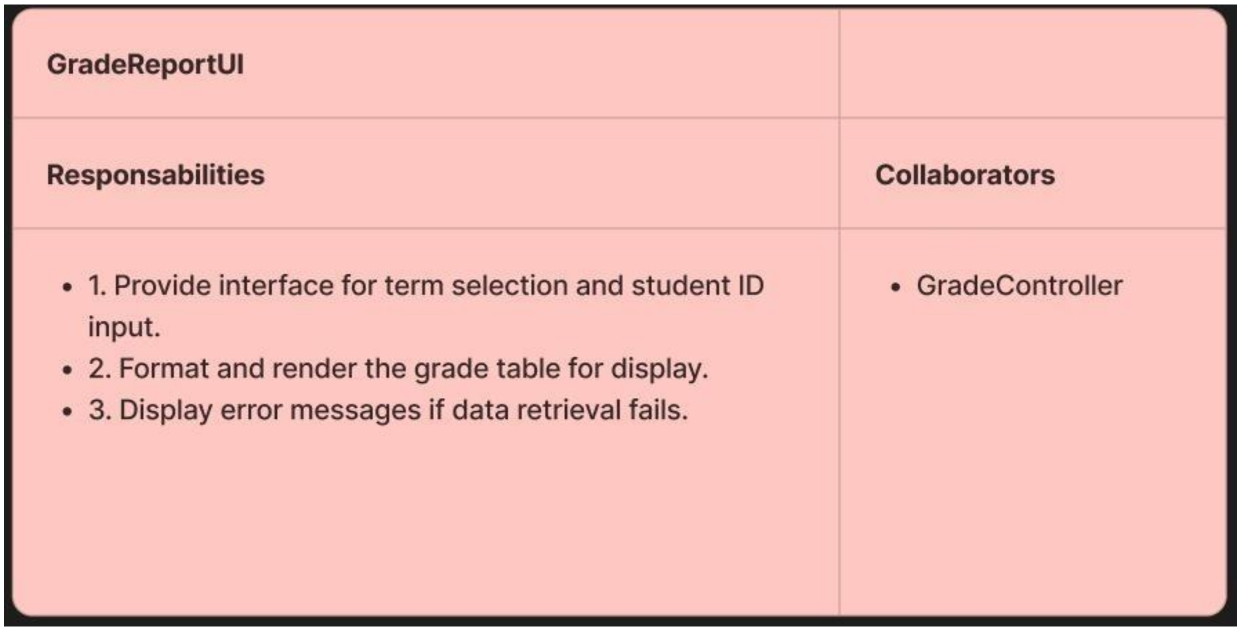
1. Use Case Name: View Grades (Simple)

- Scenario: Student or Parent checks academic performance for a specific term.
- Triggering event: User selects the “View Grades” option from the dashboard.
- Brief description: Retrieves and displays a student's grades and calculated GPA for a selected academic term.
- Actors: Primary: Student or Parent. Secondary: Registrar (data source).
- Stakeholders: Students; Parents; Teachers; Academic Advisors.
- Related use cases: Enter or Update Grades (grades must be entered first).
- Preconditions: * User is authenticated and logged into the system.
    - Grade records for the requested term have been published.
- Postconditions: * Grade report is displayed; GPA is calculated and shown.
- Flow of activities:
    1. Actor requests “View Grades” and selects a specific academic term.
    2. System (GradeController) validates the student session.
    3. System (GradeDataAccess) queries the database for matching GradeRecords.
    4. System calculates the term GPA based on retrieved scores and credits.
    5. System displays the grade list and GPA summary on the GradeReportUI.
- Exception conditions:
    - No grades found for selected term → display "No records available" message.
    - Session timeout → redirect to Login screen.

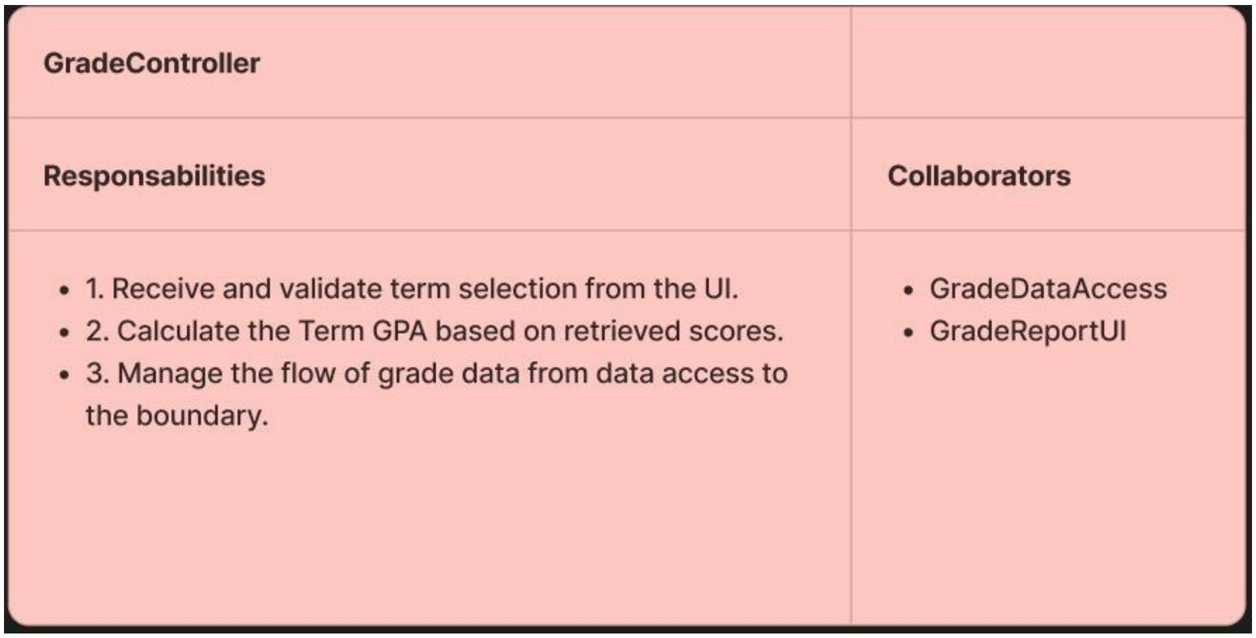
First Cut Design Class Diagram :



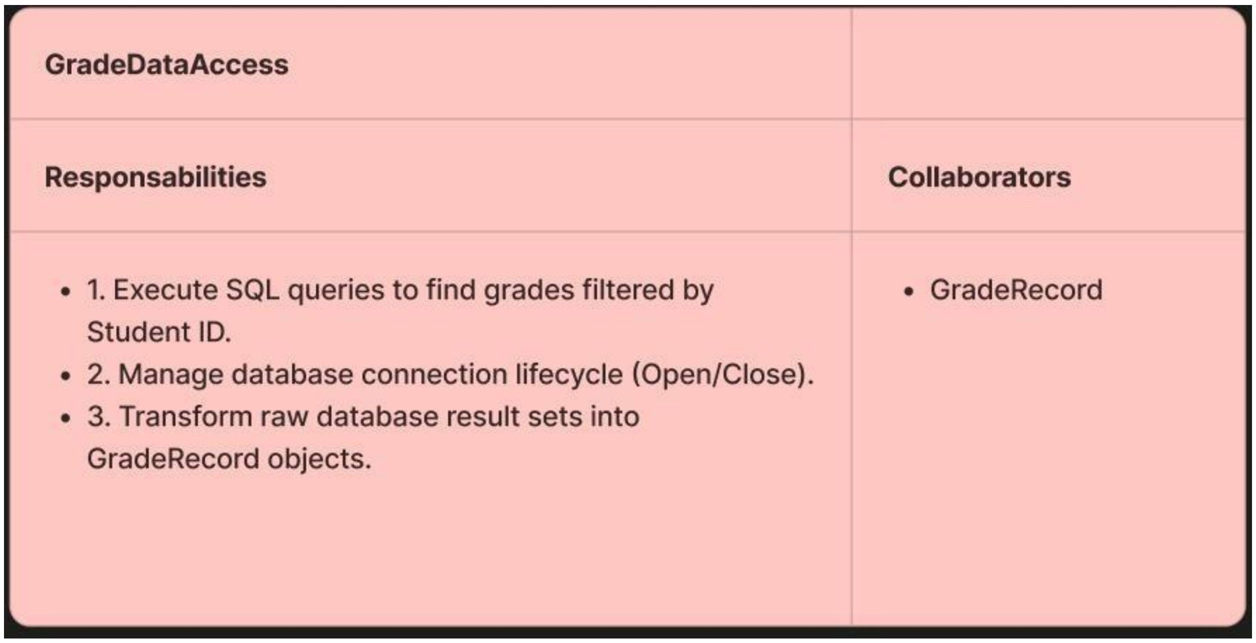
CRC Cards :



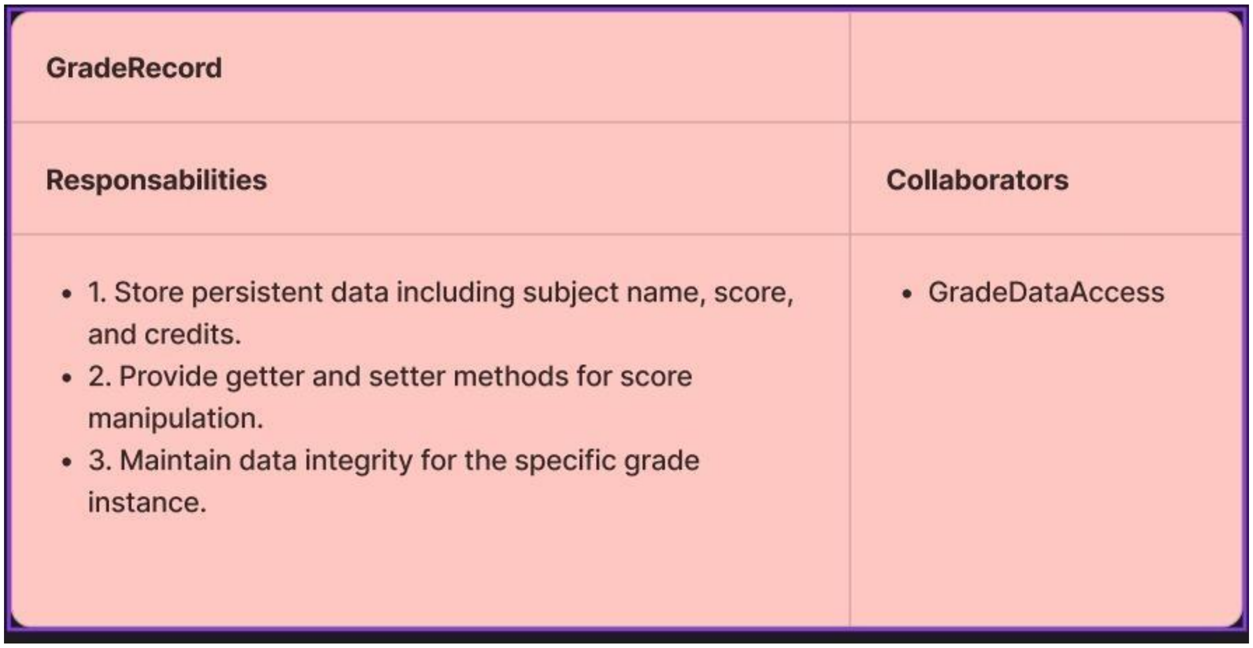
GradeReportUI	
Responsabilities	**Collaborators**
• 1. Provide interface for term selection and student ID input. • 2. Format and render the grade table for display. • 3. Display error messages if data retrieval fails.	• GradeController



GradeController	
Responsabilities	**Collaborators**
• 1. Receive and validate term selection from the UI. • 2. Calculate the Term GPA based on retrieved scores. • 3. Manage the flow of grade data from data access to the boundary.	• GradeDataAccess • GradeReportUI

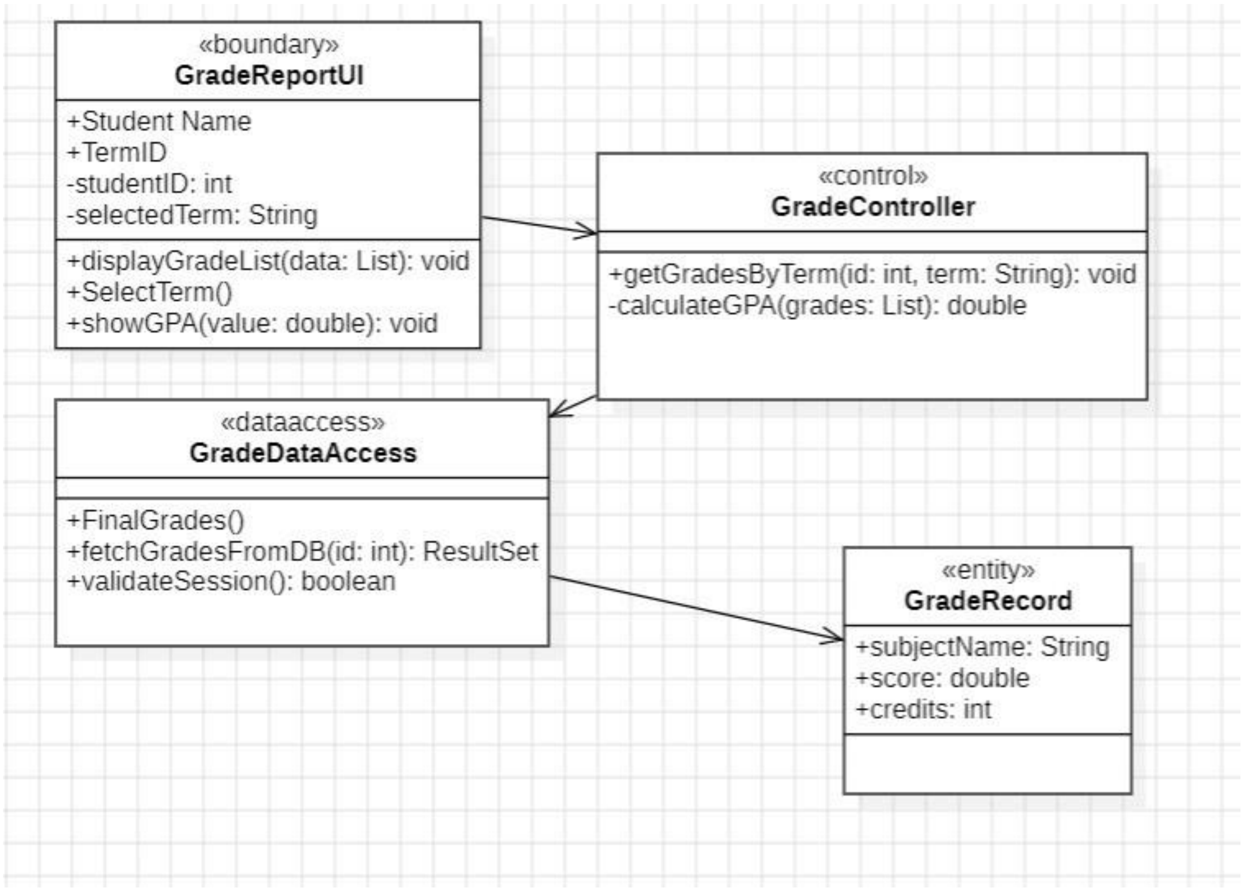


GradeDataAccess	
Responsabilities	**Collaborators**
• 1. Execute SQL queries to find grades filtered by Student ID. • 2. Manage database connection lifecycle (Open/Close). • 3. Transform raw database result sets into GradeRecord objects.	• GradeRecord



GradeRecord	
Responsabilities	**Collaborators**
• 1. Store persistent data including subject name, score, and credits. • 2. Provide getter and setter methods for score manipulation. • 3. Maintain data integrity for the specific grade instance.	• GradeDataAccess

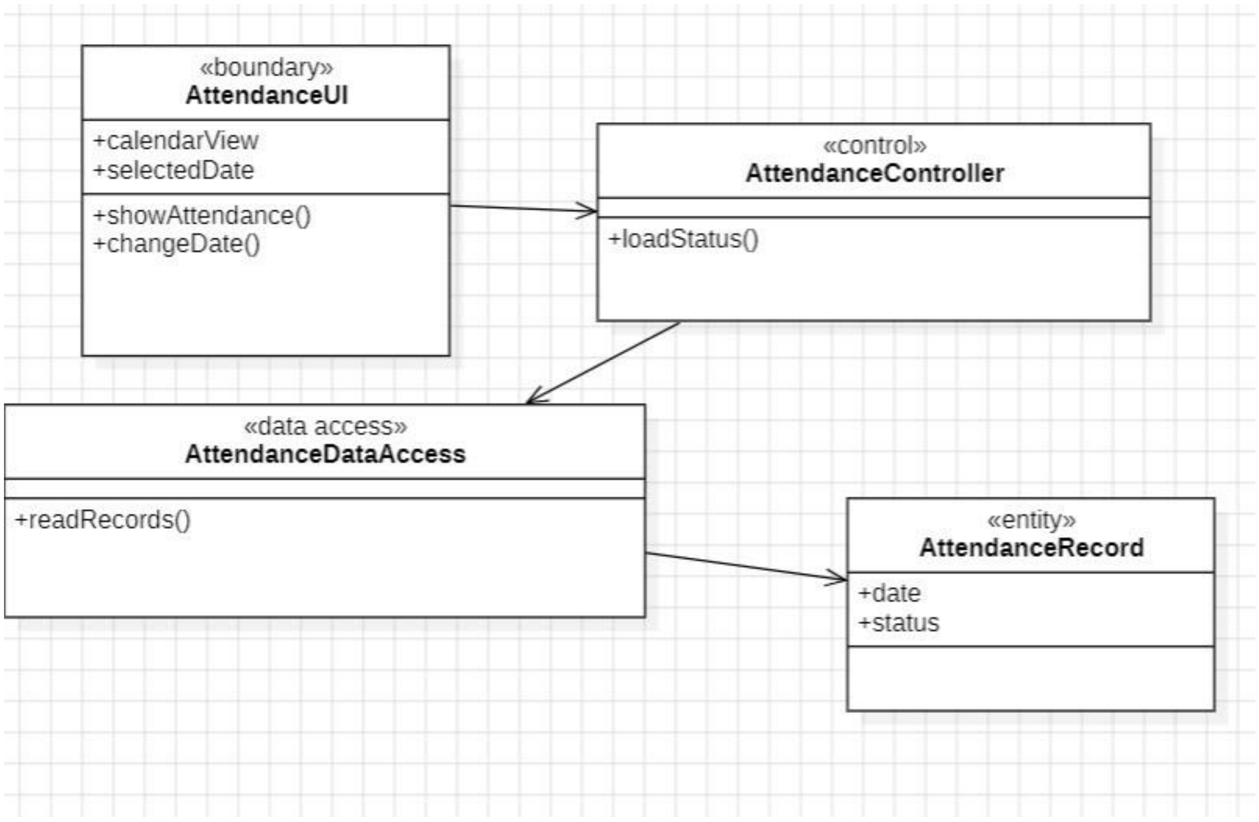
Updated Class Diagram :



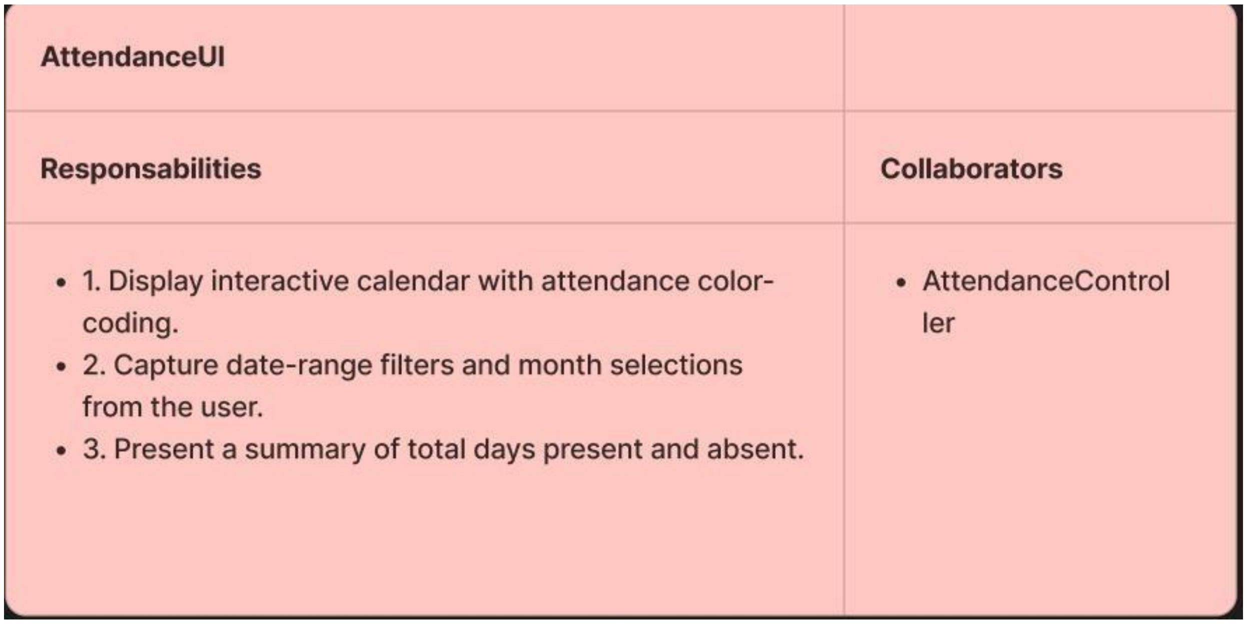
2. Use Case Name: View Attendance (Simple)

- Scenario: User monitors daily attendance status (Present/Absent).
- Triggering event: User selects “Attendance History.”
- Brief description: Displays a calendar or list view of a student's attendance records within a chosen date range.
- Actors: Primary: Student or Parent. Secondary: Teacher (who recorded it).
- Stakeholders: Students; Parents; School Administration.
- Related use cases: Record Attendance (attendance must be recorded first).
- Preconditions: * User is authorized to view the specific student's data.
    - AttendanceUI is initialized.
- Postconditions: * Attendance grid is rendered; total presence/absence summary is provided.
- Flow of activities:
    1. Actor selects a month or date range in the AttendanceUI.
    2. System (AttendanceController) requests records for that specific range.
    3. System (AttendanceDataAccess) retrieves AttendanceRecords from the DB.
    4. System computes the attendance percentage.
    5. System highlights absent days in red and present days in green on the UI.
- Exception conditions:
    - Invalid date range (e.g., end date before start date) → show error message.
    - Database connection failure → notify user to try again later.

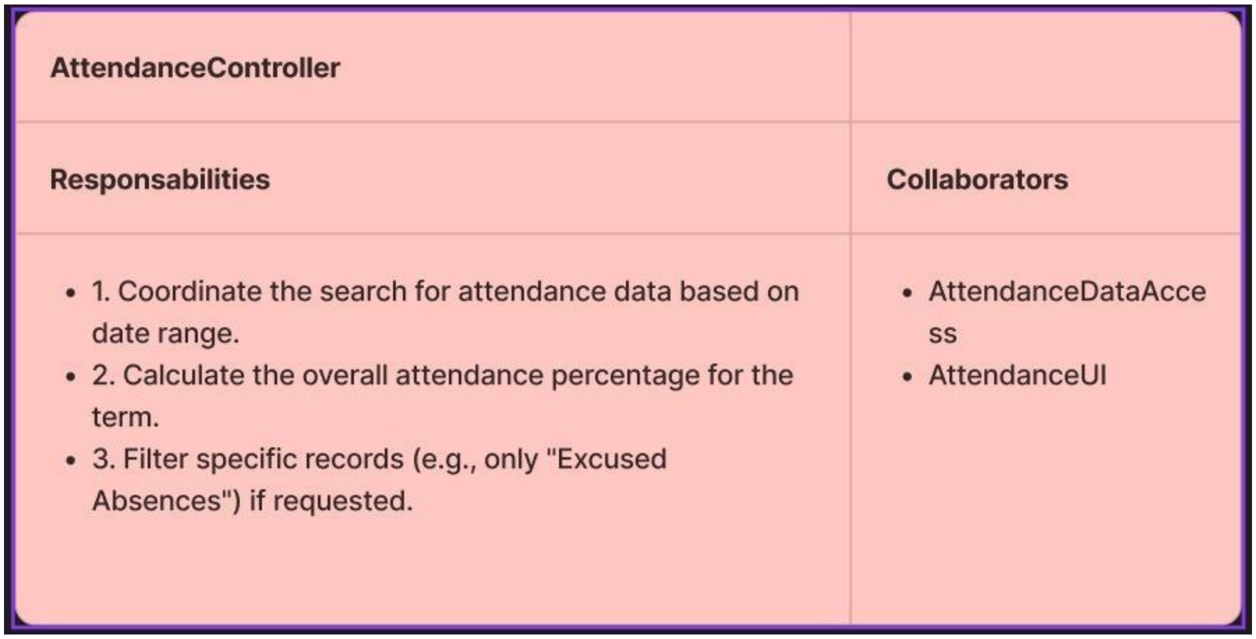
First Cut Design Class Diagram :



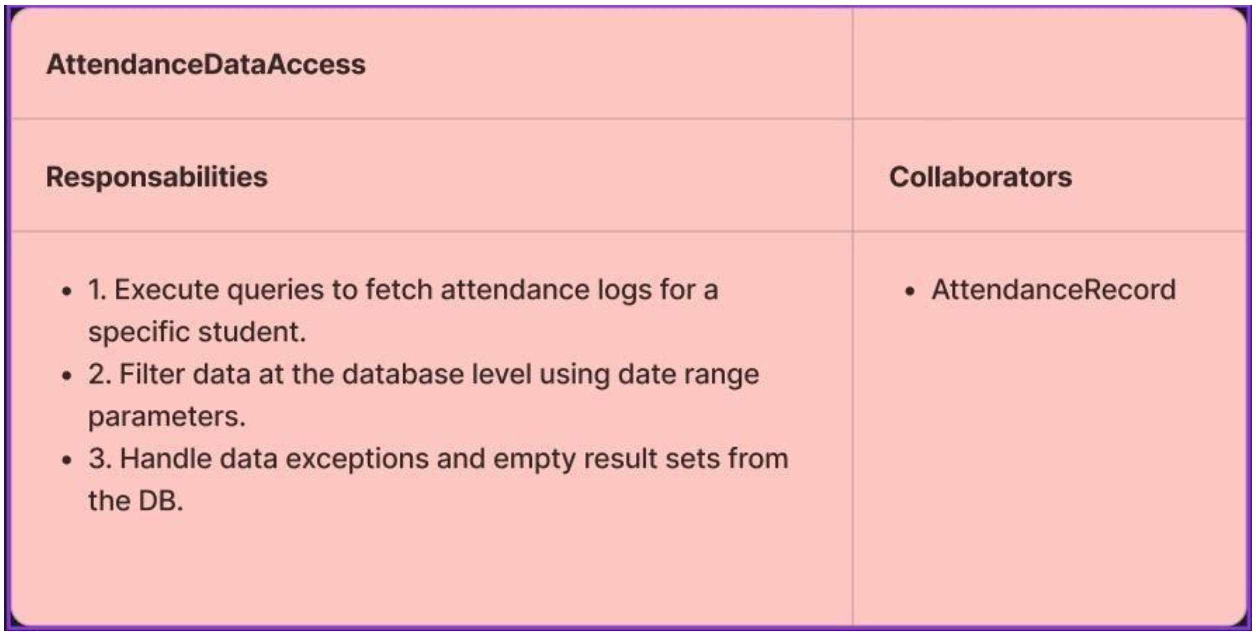
CRC Cards :



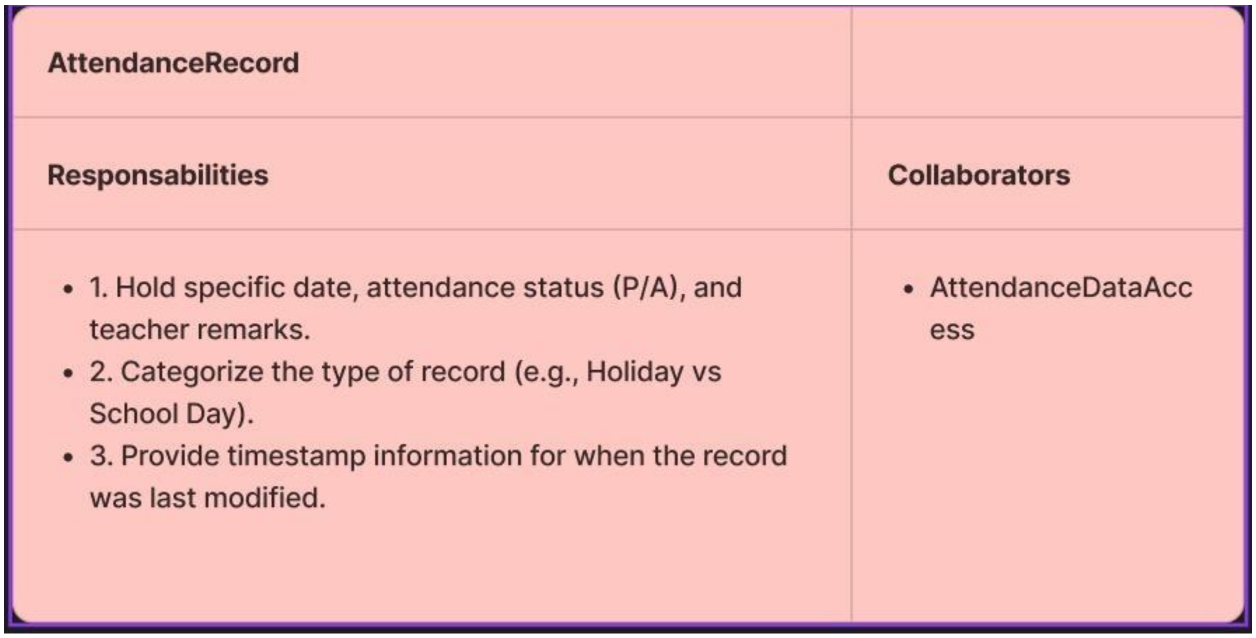
AttendanceUI	
Responsabilities	**Collaborators**
• 1. Display interactive calendar with attendance color-coding. • 2. Capture date-range filters and month selections from the user. • 3. Present a summary of total days present and absent.	• AttendanceController



AttendanceController	
Responsabilities	**Collaborators**
• 1. Coordinate the search for attendance data based on date range. • 2. Calculate the overall attendance percentage for the term. • 3. Filter specific records (e.g., only "Excused Absences") if requested.	• AttendanceDataAccess • AttendanceUI

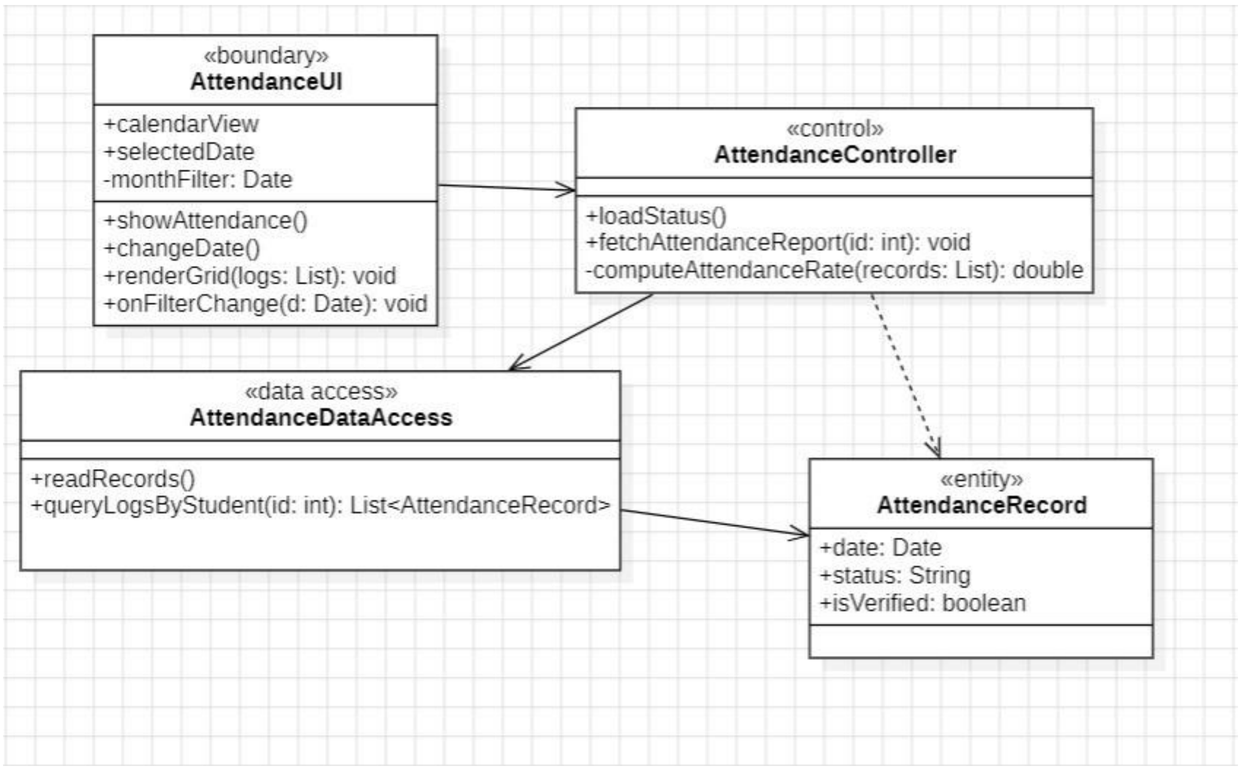


AttendanceDataAccess	
Responsabilities	**Collaborators**
• 1. Execute queries to fetch attendance logs for a specific student. • 2. Filter data at the database level using date range parameters. • 3. Handle data exceptions and empty result sets from the DB.	• AttendanceRecord



AttendanceRecord	
Responsabilities	**Collaborators**
• 1. Hold specific date, attendance status (P/A), and teacher remarks. • 2. Categorize the type of record (e.g., Holiday vs School Day). • 3. Provide timestamp information for when the record was last modified.	• AttendanceDataAccess

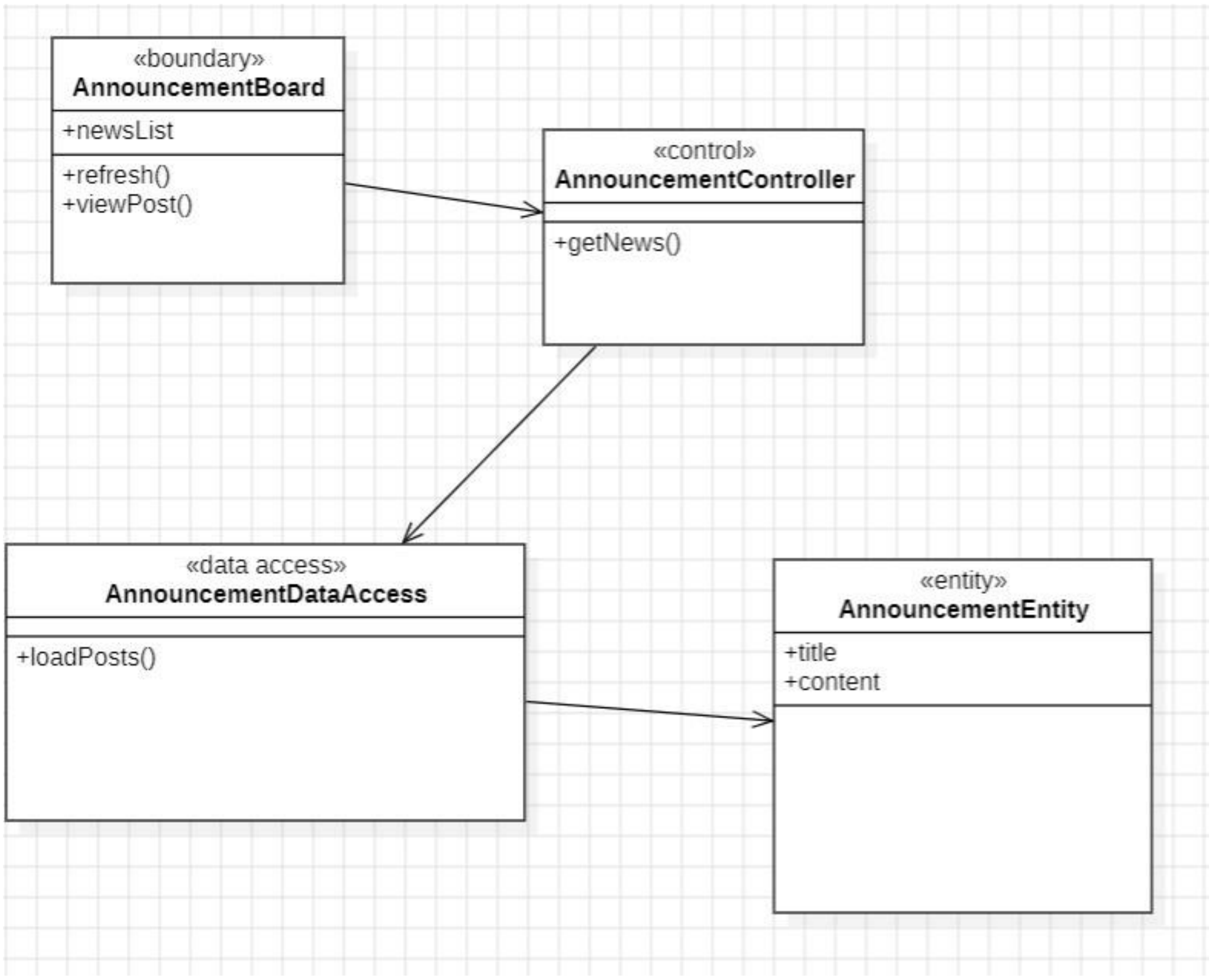
Updated Diagram :



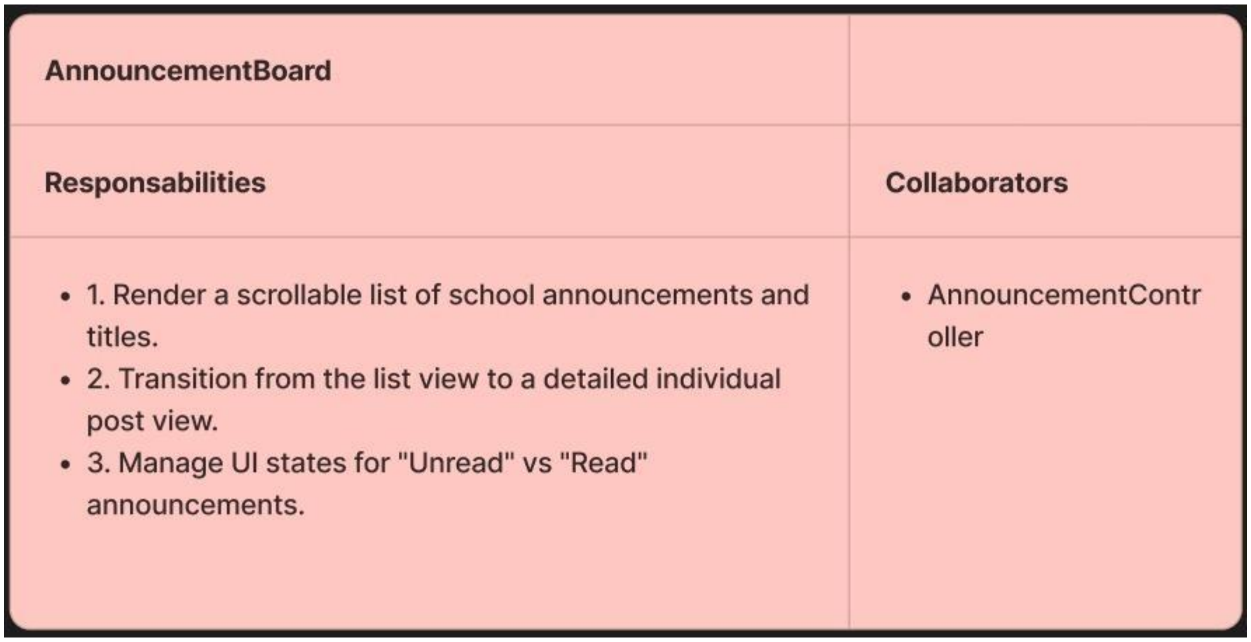
3. Use Case Name: View Announcements (Simple)

- Scenario: User reads school news, updates, or urgent notices.
- Triggering event: User navigates to the “Announcements” section or clicks a notification.
- Brief description: Provides a filtered list of announcements relevant to the user's role (Student, Parent, or Teacher).
- Actors: Primary: All Users (Student, Parent, Teacher, Admin).
- Stakeholders: Entire School Community.
- Related use cases: Announcements and Communication (creation of the post).
- Preconditions: * User is logged in (to determine role-based filtering).
- Postconditions: * List of active announcements is displayed; unread posts are marked.
- Flow of activities:
    1. Actor requests to see the AnnouncementBoard.
    2. System (AnnouncementController) checks the user’s role.
    3. System (AnnouncementDataAccess) fetches non-expired posts for that role.
    4. System sorts posts by urgency (priority) and date.
    5. System displays the titles; Actor clicks a title to view full content.
- Exception conditions:
    - No active announcements → show "No new announcements" message.

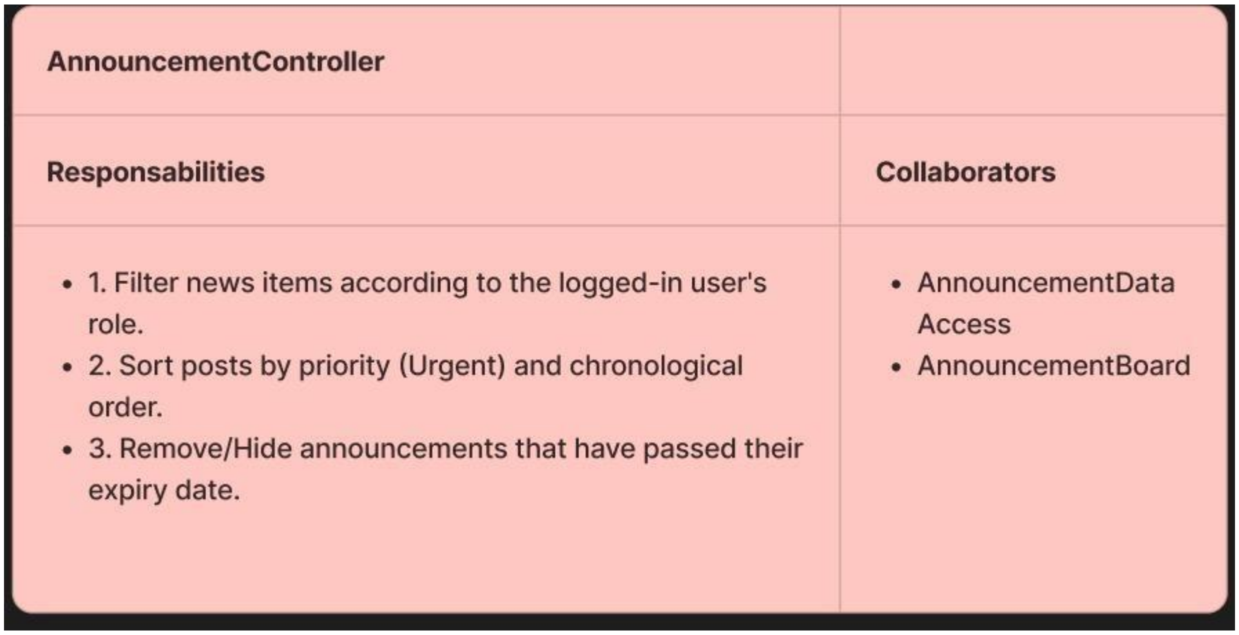
First Cut Design Class Diagram :



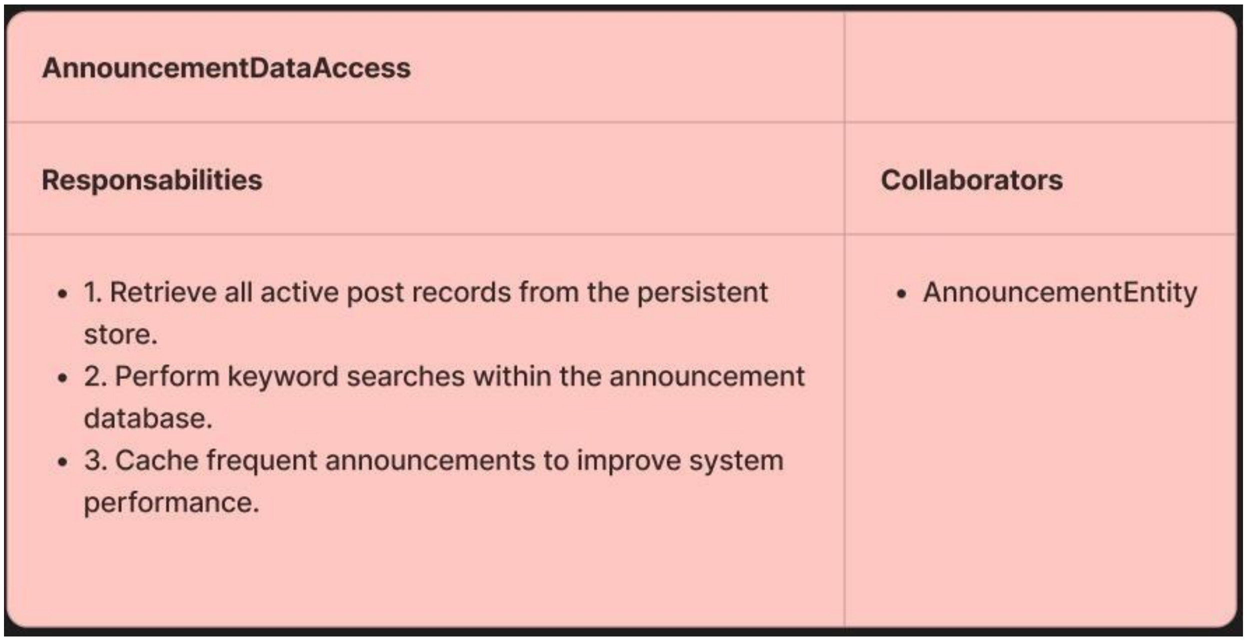
CRC Cards :



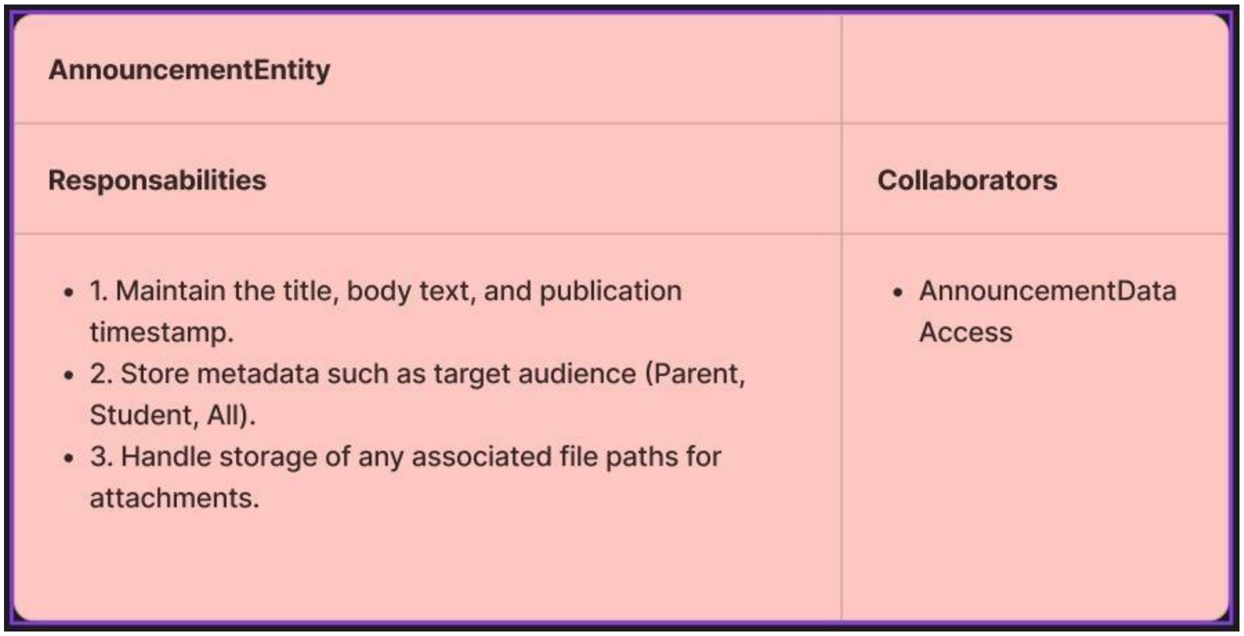
AnnouncementBoard	
Responsabilities	**Collaborators**
• 1. Render a scrollable list of school announcements and titles. • 2. Transition from the list view to a detailed individual post view. • 3. Manage UI states for "Unread" vs "Read" announcements.	• AnnouncementController



AnnouncementController	
Responsabilities	**Collaborators**
• 1. Filter news items according to the logged-in user's role. • 2. Sort posts by priority (Urgent) and chronological order. • 3. Remove/Hide announcements that have passed their expiry date.	• AnnouncementDataAccess • AnnouncementBoard

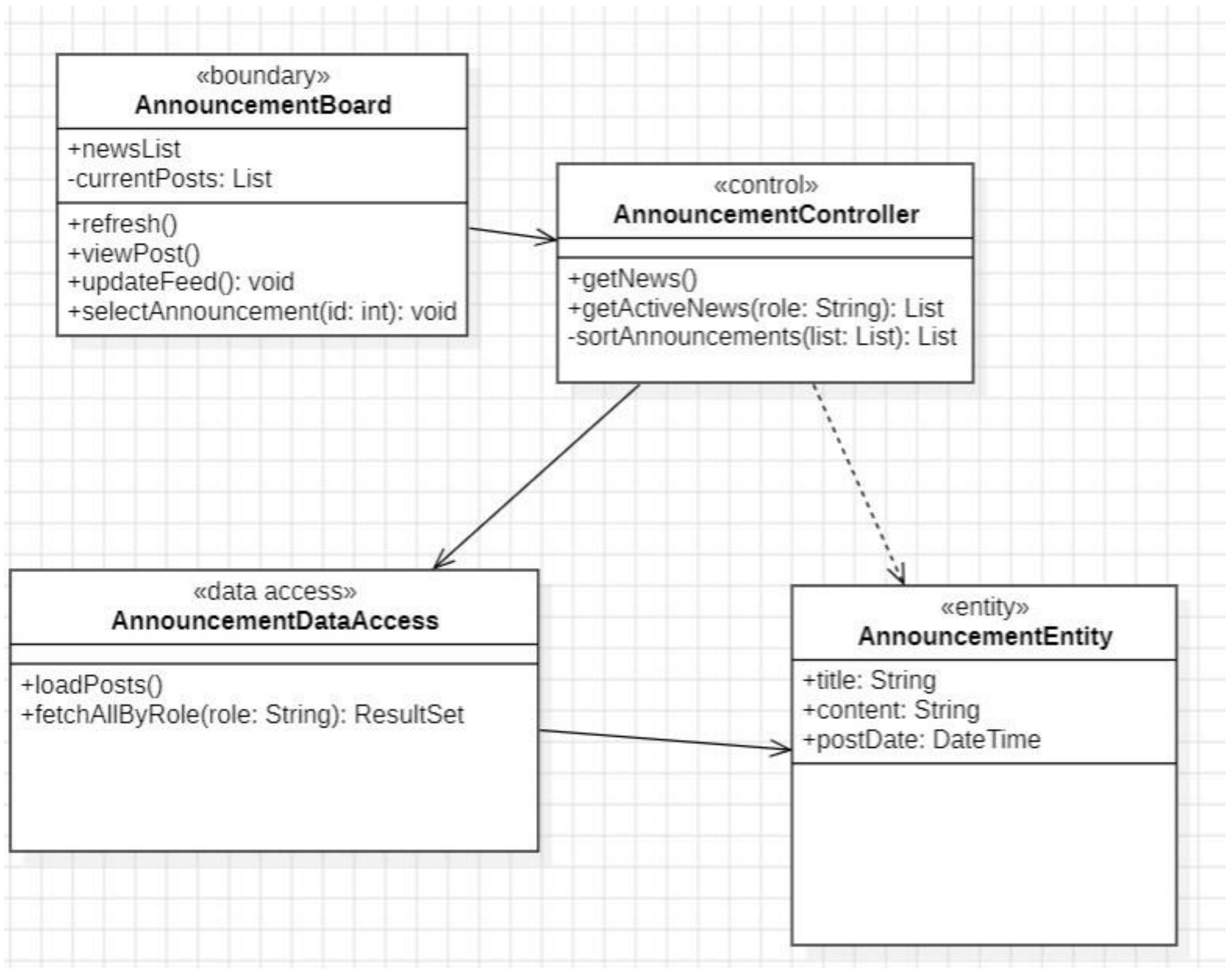


AnnouncementDataAccess	
Responsabilities	**Collaborators**
• 1. Retrieve all active post records from the persistent store. • 2. Perform keyword searches within the announcement database. • 3. Cache frequent announcements to improve system performance.	• AnnouncementEntity



AnnouncementEntity	
Responsabilities	**Collaborators**
• 1. Maintain the title, body text, and publication timestamp. • 2. Store metadata such as target audience (Parent, Student, All). • 3. Handle storage of any associated file paths for attachments.	• AnnouncementData Access

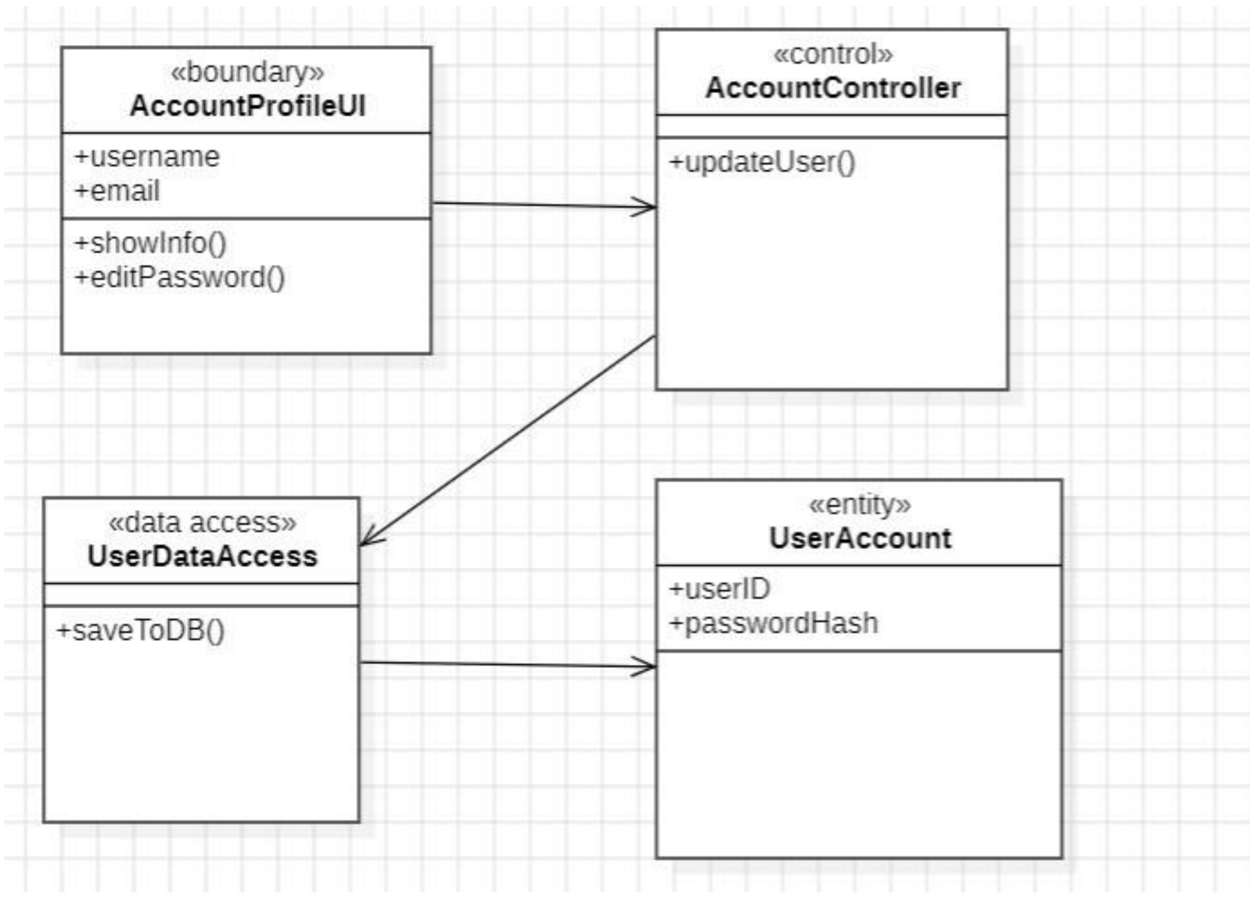
Updated Diagram :



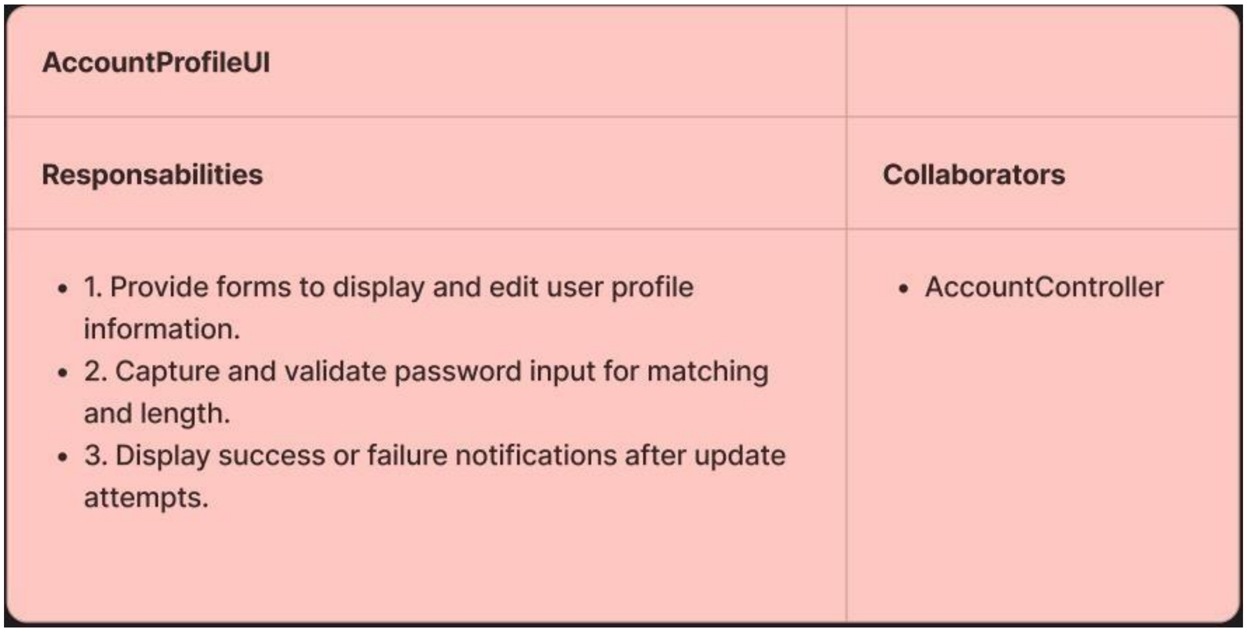
4. Use Case Name: Manage User Accounts (Profile Update)(Simple)

- Scenario: User updates their personal profile information or password.
- Triggering event: User clicks on “Profile Settings.”
- Brief description: Allows users to view their account details and update their password or contact info securely.
- Actors: Primary: Any registered User (Admin, Teacher, Student, Parent).
- Stakeholders: System Admin; All Users; Security/Audit Team.
- Related use cases: Manage User Accounts (Admin-side management).
- Preconditions: * User is authenticated and currently active in the session.
- Postconditions: * UserAccount entity is updated in the database; password is re-hashed.
    - Security log of the change is recorded.
- Flow of activities:
    1. Actor requests “Profile Settings” and enters a new password or email.
    2. System validates password complexity and matching fields.
    3. System (AccountController) hashes the new password for security.
    4. System (UserDataAccess) commits the update to the UserAccount record.
    5. System confirms the success and sends a confirmation notification.
- Exception conditions:
    - Current password incorrect → reject change and notify actor.
    - New password does not meet complexity rules → show requirements error.

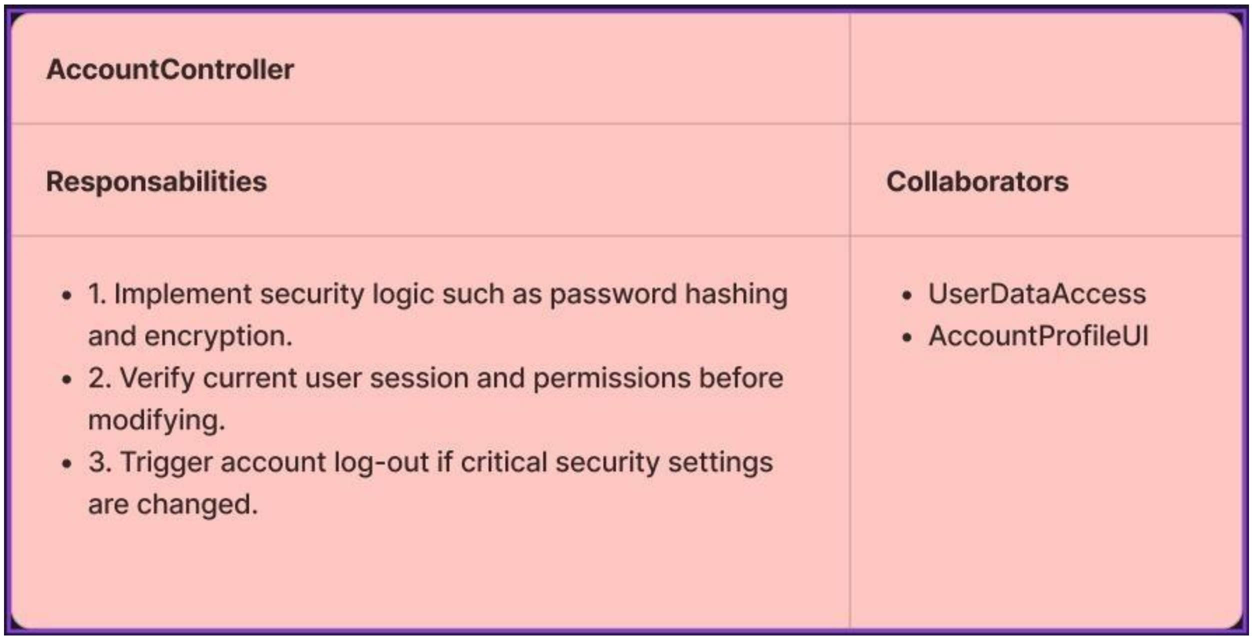
First Cut Design Class Diagram :



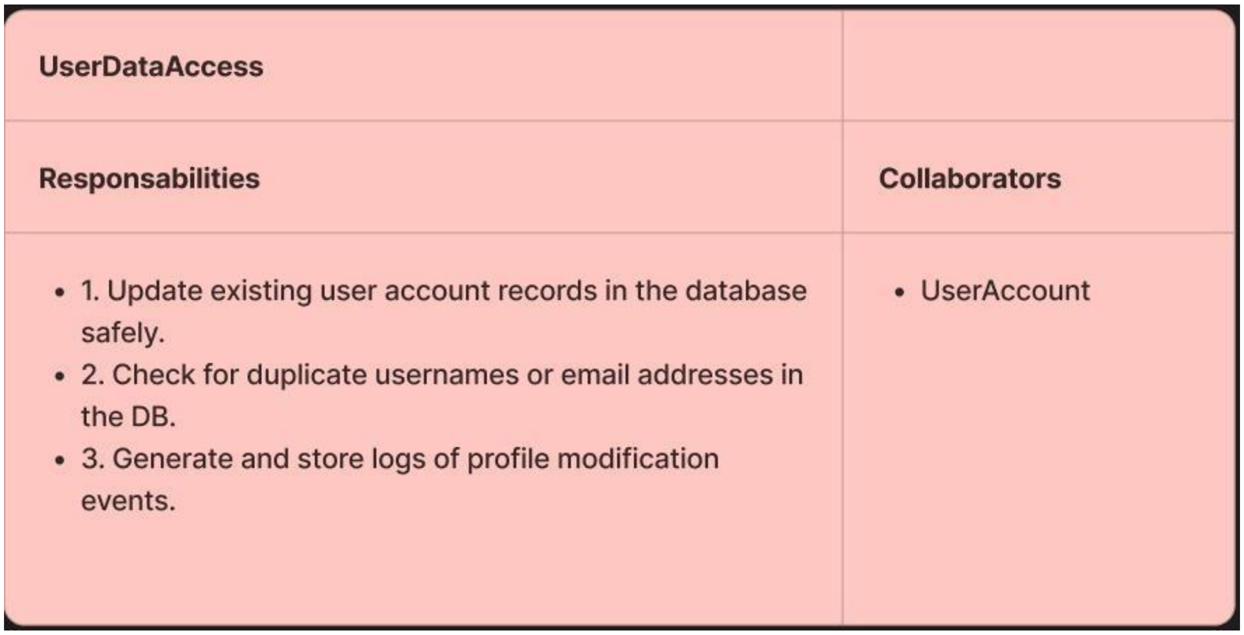
CRC Cards :



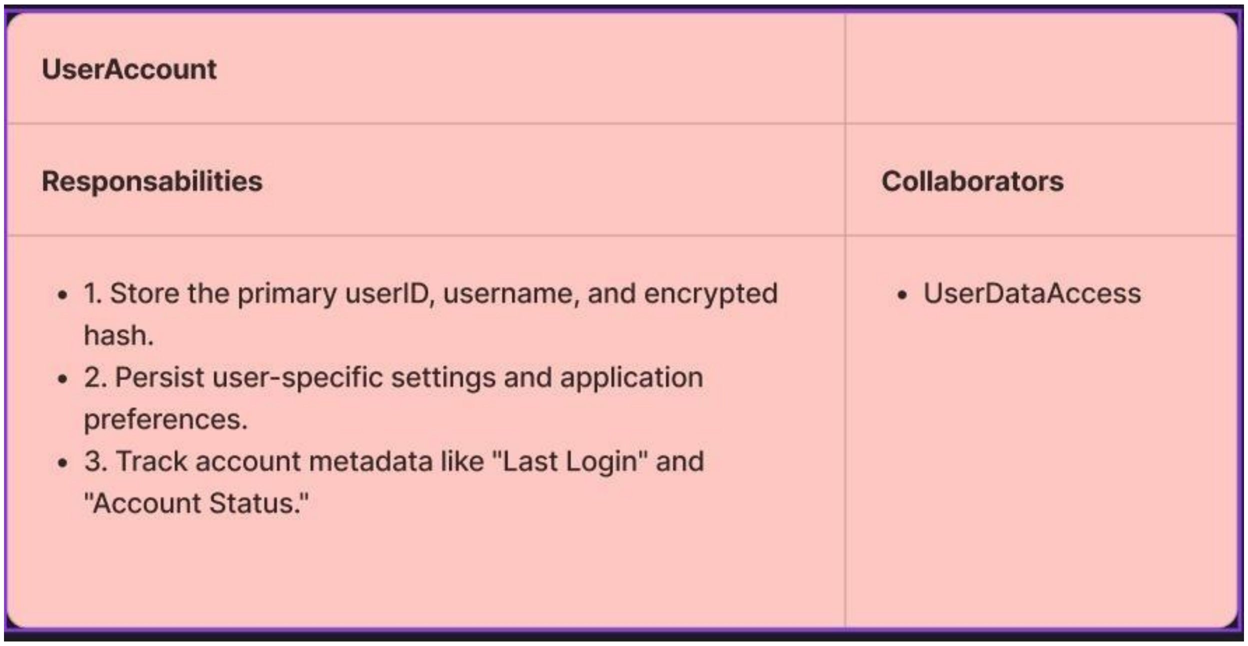
AccountProfileUI	
Responsabilities	**Collaborators**
• 1. Provide forms to display and edit user profile information. • 2. Capture and validate password input for matching and length. • 3. Display success or failure notifications after update attempts.	• AccountController



AccountController	
Responsabilities	**Collaborators**
• 1. Implement security logic such as password hashing and encryption. • 2. Verify current user session and permissions before modifying. • 3. Trigger account log-out if critical security settings are changed.	• UserDataAccess • AccountProfileUI

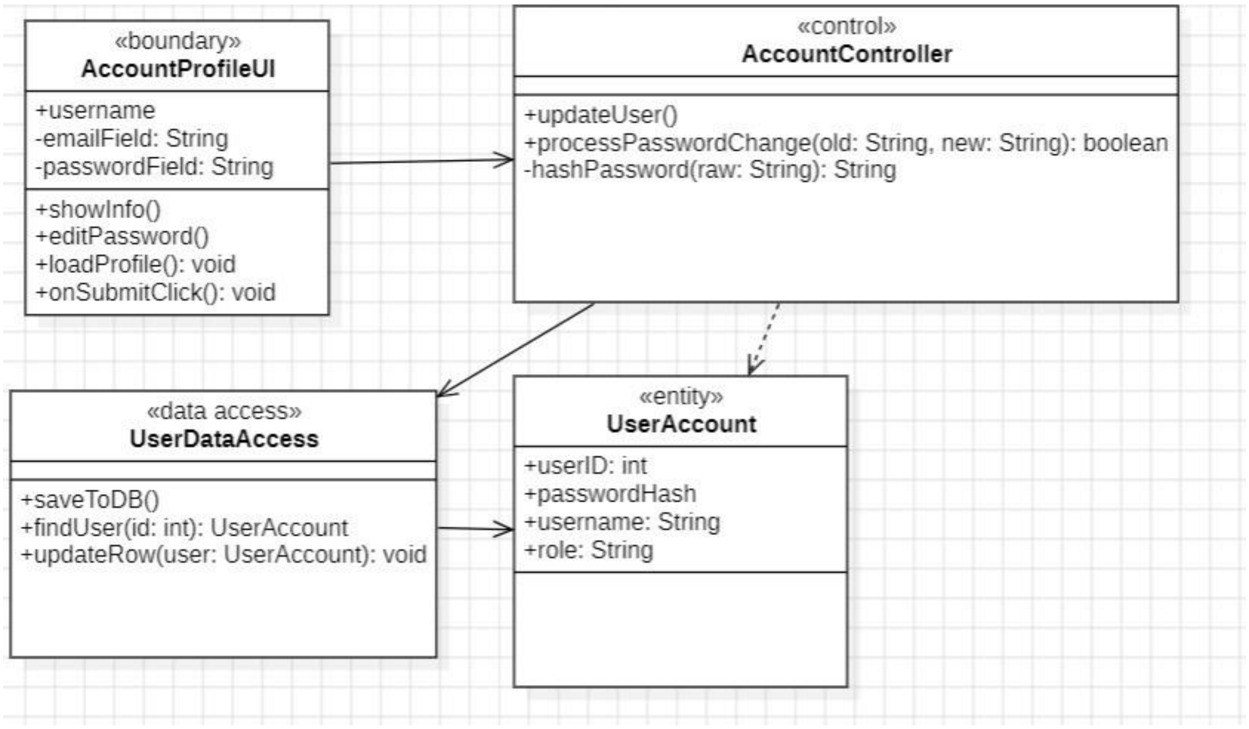


UserDataAccess	
Responsabilities	**Collaborators**
• 1. Update existing user account records in the database safely. • 2. Check for duplicate usernames or email addresses in the DB. • 3. Generate and store logs of profile modification events.	• UserAccount



UserAccount	
Responsabilities	**Collaborators**
• 1. Store the primary userID, username, and encrypted hash. • 2. Persist user-specific settings and application preferences. • 3. Track account metadata like "Last Login" and "Account Status."	• UserDataAccess

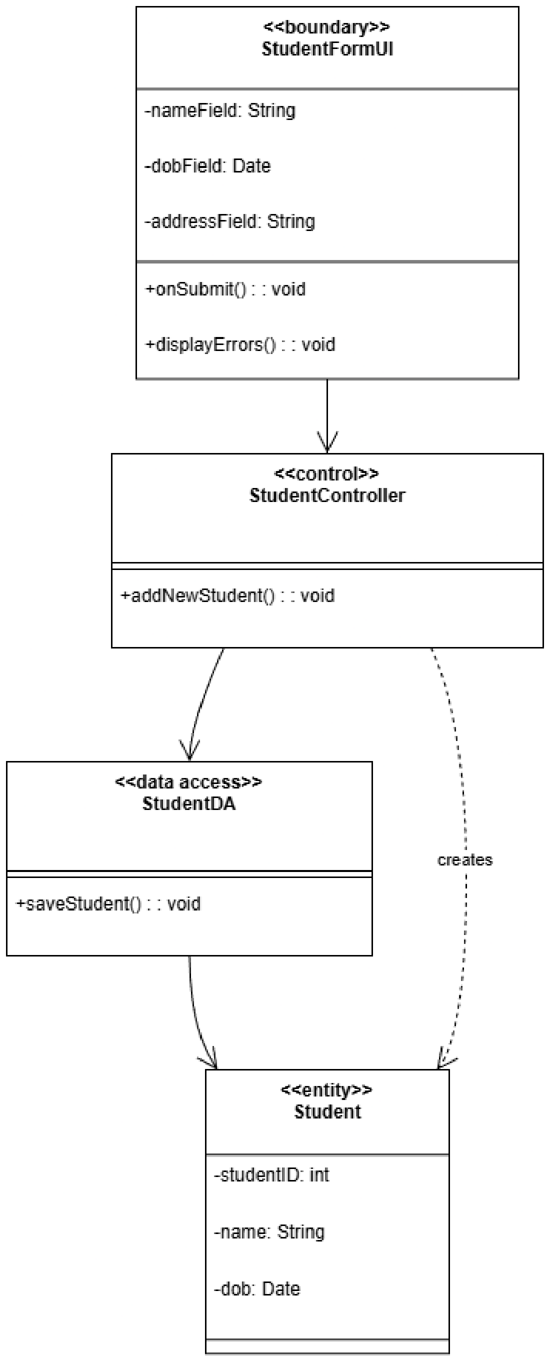
Updated Diagram :



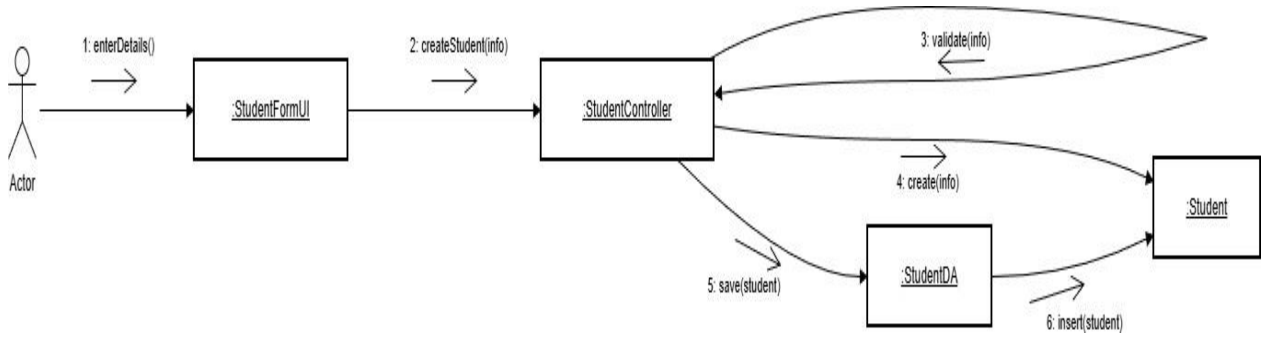
1. Use Case Name: Add Student Info (Moderate)

- **Scenario:** Admin manually enters details to create a new student profile in the system.
- **Triggering event:** Admin submits the "New Student" form.
- **Brief description:** Validate input data (Name, Email, etc.) and create a new Student entity in the database.
- **Actors:** Primary: Admin. Secondary: System.
- **Stakeholders:** Admin; Registrar; Students.
- **Related use cases:** Manage User Accounts.
- **Preconditions:**
    - Admin is authenticated and logged in.
    - The student does not already exist (checked by National ID or Email).
- **Postconditions:**
    - A new Student record is created in the database.
    - The system generates a unique studentID.
- **Flow of activities:**
    - Admin selects "Add Student" and enters student details (Name, Email, Enrollment Date).
    - System (StudentForm) captures the input.
    - System (StudentController) validates that all required fields are filled and email format is correct.
    - System (StudentController) requests creation of a new Student entity.
    - System (Student) creates a new instance and saves data.
    - The system confirms the success and sends a notification to the Admin.
- **Exception conditions:**
    - Validation fails (empty fields) → Show error message.
    - Duplicate email found → Reject and notify Admin.

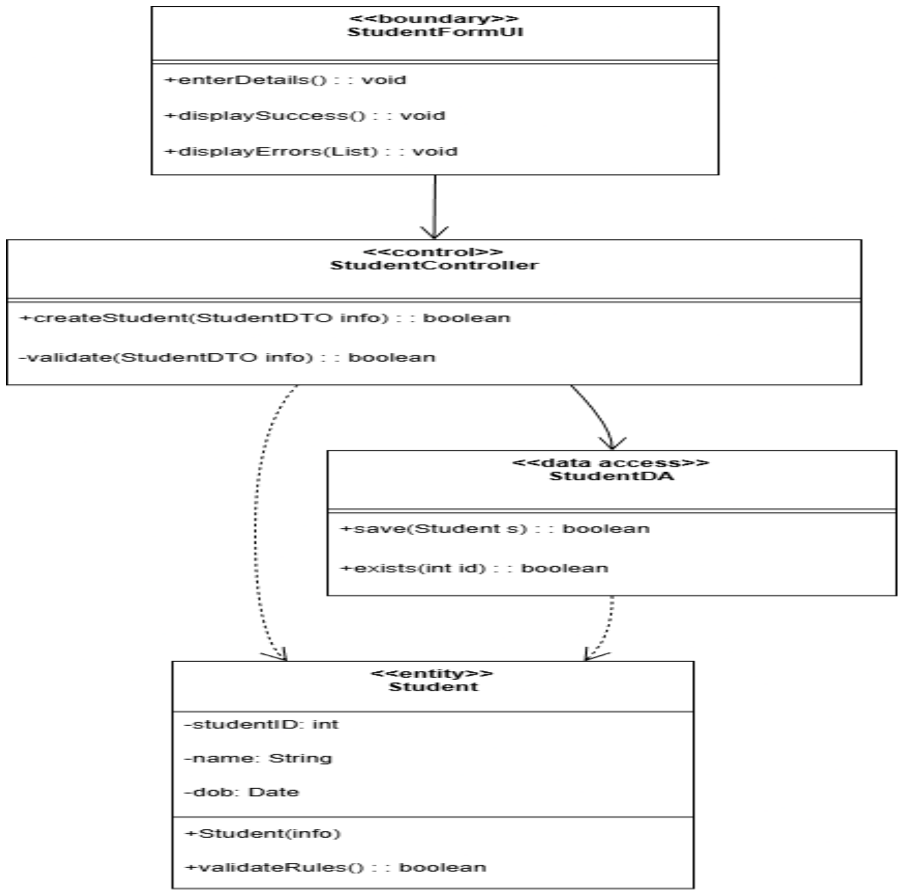
First Cut Design Class Diagram :



Communication Diagram :



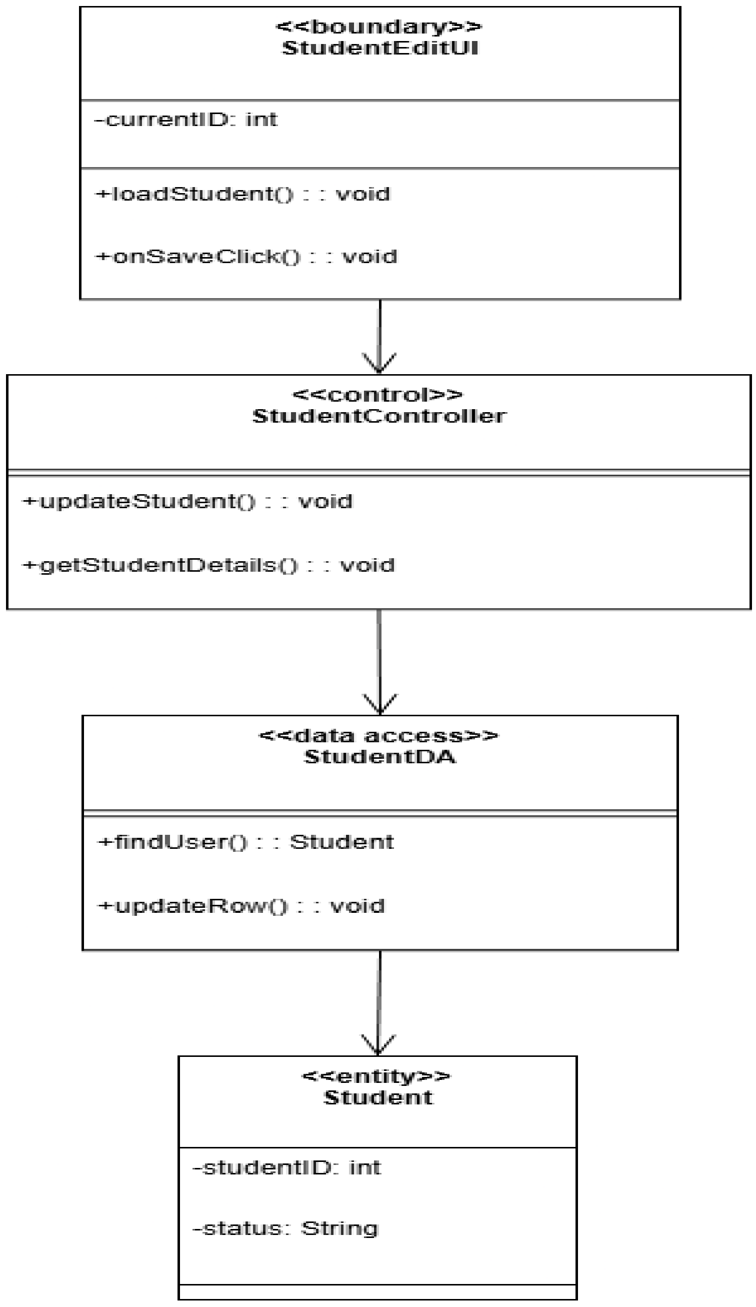
Updated Diagram:


<<boundary>>
StudentFormUI
+enterDetails() : : void
+displaySuccess() : : void
+displayErrors(List) : : void
<<control>>
StudentController
+createStudent(StudentDTO info) : : boolean
-validate(StudentDTO info) : : boolean
<<data access>>
StudentDA
+save(Student s) : : boolean
+exists(int id) : : boolean
<<entity>>
Student
-studentID: int
-name: String
-dob: Date
+Student(info)
+validateRules() : : boolean

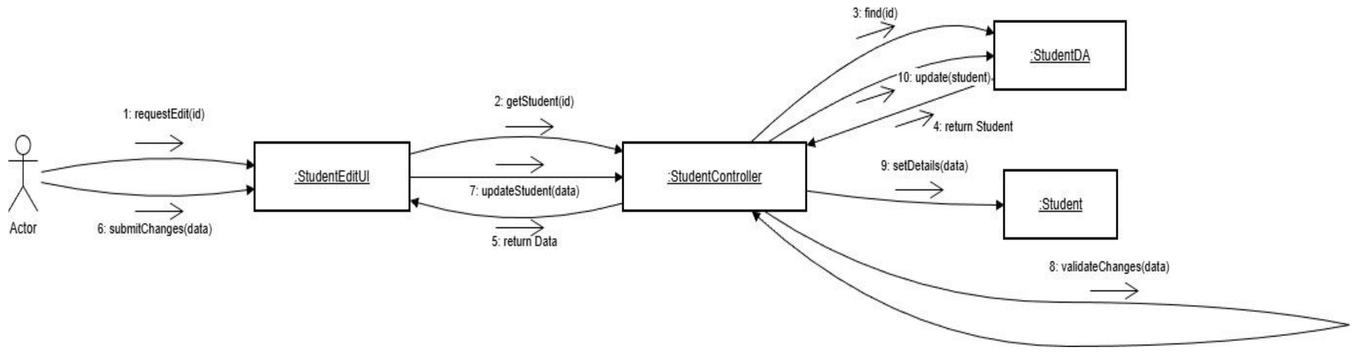
2. Use Case Name: Edit Student Info (Moderate)

- **Scenario**: Admin updates the email or name of an existing student.
- **Triggering event**: Admin saves changes to an existing student profile.
- **Brief description**: Find specific student records, validate changes, and update the entity data.
- **Actors**: Primary: Admin.
- **Stakeholders**: Admin; Student.
- **Related use cases**: Add Student Info; View Grades.
- **Preconditions**:
  - Admin is authenticated.
  - Student records exist in the system.
- **Postconditions**:
  - Student attributes are updated permanently in the database.
- **Flow of activities:**
  - Admin searches for a student, selects "Edit," and modifies fields (e.g., changes Email).
  - System (EditStudentView) captures the updated data.
  - System (StudentController) validates the new data format.
  - System (StudentController) finds the specific Student object by ID.
  - System (Student) updates its details with the new values.
  - The system confirms the update to the Admin.
- **Exception conditions:**
  - Invalid data format → Reject update.
  - Student ID not found in database → Show "Record not found" error.

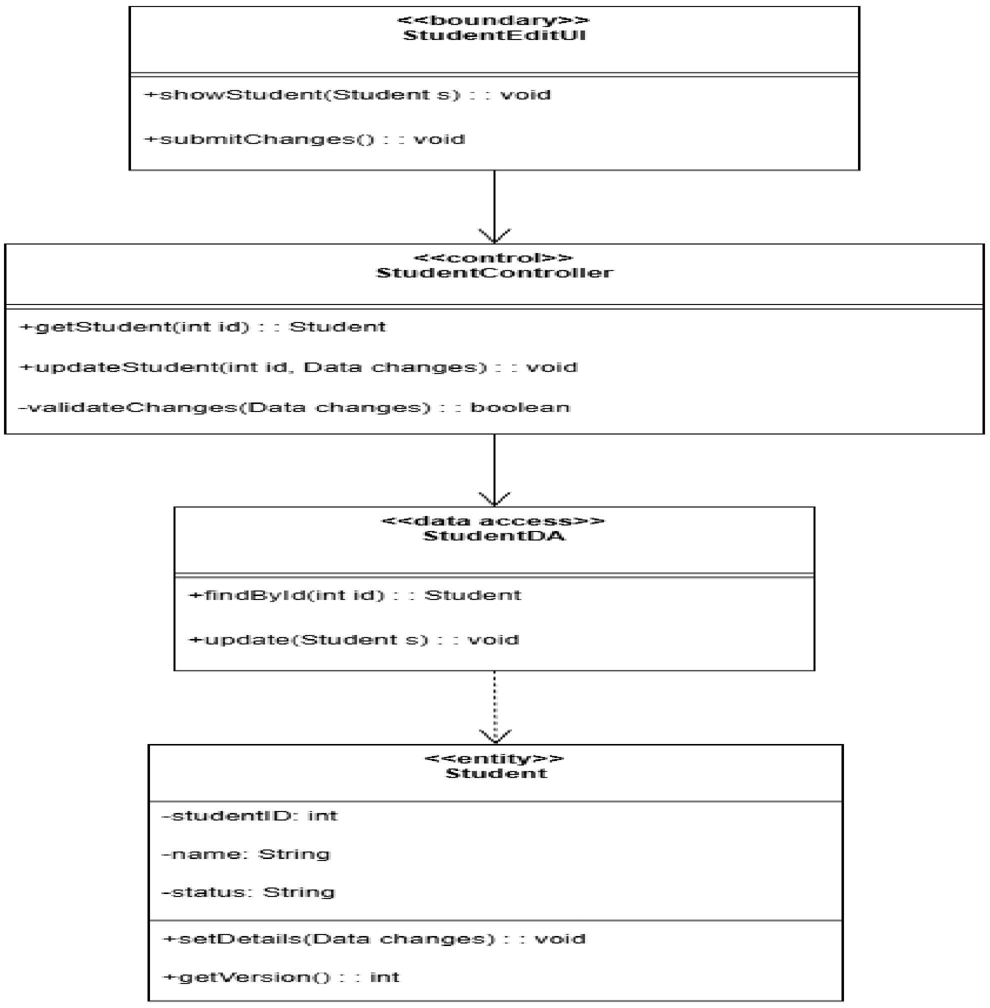
First Cut Design Class Diagram :



Communication Diagram :



Updated Diagram :

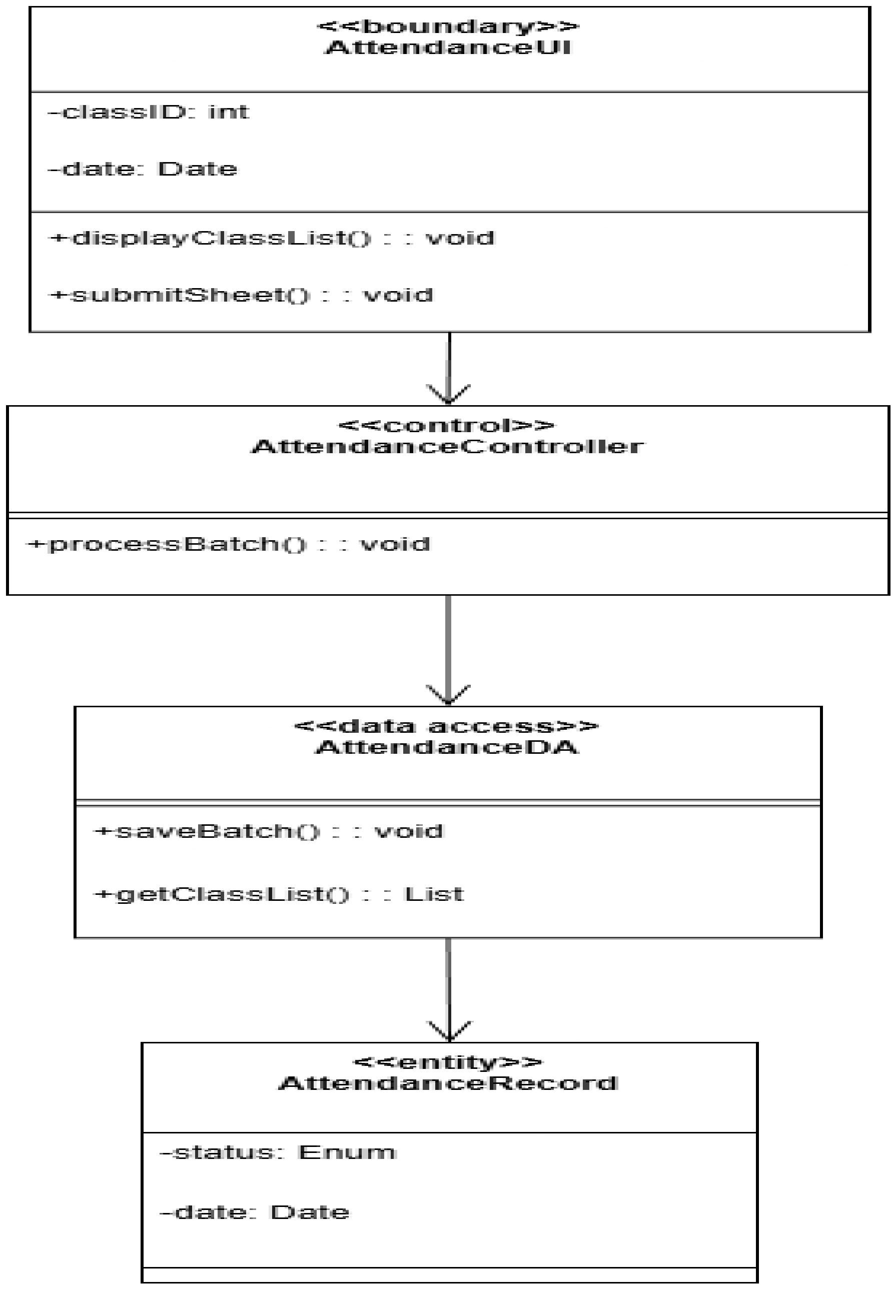


<<boundary>>
StudentEditUI
+showStudent(Student s) : : void
+submitChanges() : : void
<<control>>
StudentController
+getStudent(int id) : : Student
+updateStudent(int id, Data changes) : : void
-validateChanges(Data changes) : : boolean
<<data access>>
StudentDA
+findById(int id) : : Student
+update(Student s) : : void
<<entity>>
Student
-studentID: int
-name: String
-status: String
+setDetails(Data changes) : : void
+getVersion() : : int

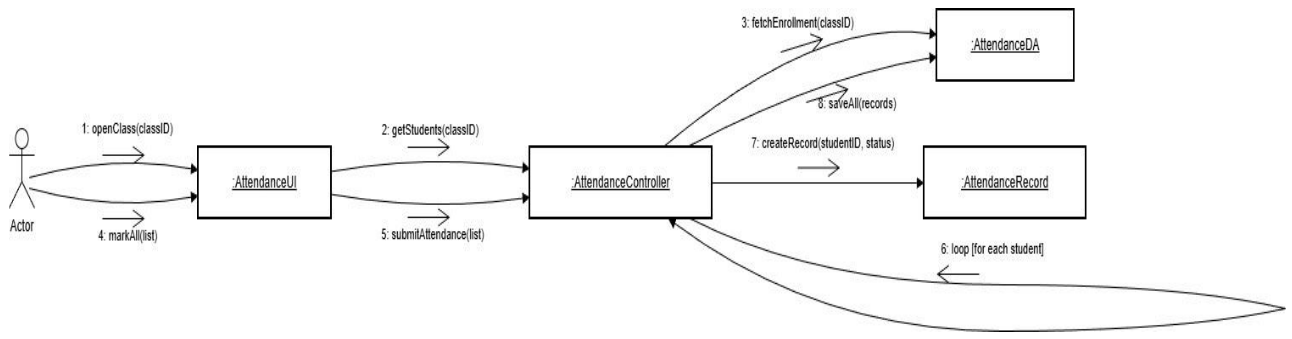
3. Use Case Name: Record Attendance (Moderate)

- **Scenario:** Teacher marks a student as "Present" or "Absent" for a specific class date.
- **Triggering event:** Teacher submits the attendance sheet.
- **Brief description:** Create a new attendance log entry linking a student to a status and date.
- **Actors:** Primary: Teacher. Secondary: Parent (notified).
- **Stakeholders:** Teacher; Student; Parent; Admin.
- **Related use cases:** View Attendance.
- **Preconditions:**
  - The teacher is assigned to the class.
  - The class session is valid and active.
- **Postconditions:**
  - A new AttendanceLog object is created.
  - Log is linked to the Student.
- **Flow of activities:**
  - The teacher selects a class/date and marks status for a student (Present/Absent).
  - System (AttendanceView) submits the data.
  - System (AttendanceController) receives the request and validates the session.
  - System (AttendanceController) initiates creation of a log entry.
  - System (AttendanceLog) creates itself with date and status.
  - System (Student) links the new log to its record.
- **Exception conditions:**
  - Database connection failure → Notify Teacher to try again.
  - Attendance already recorded for this date → Prompt to overwrite or cancel.

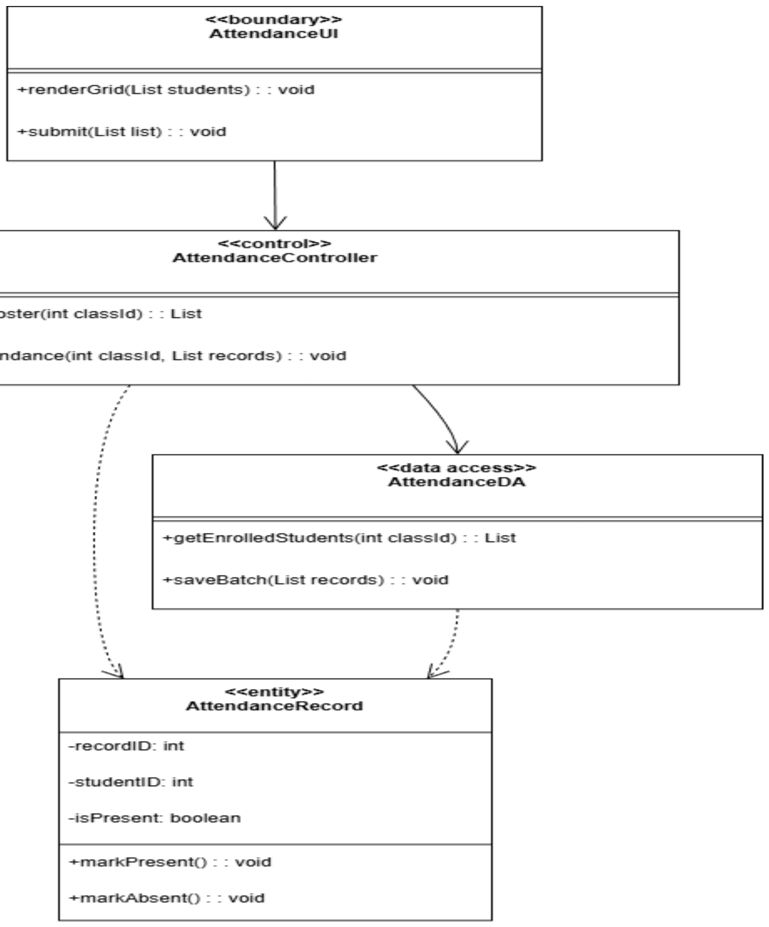
First Cut Design Class Diagram :



Communication Diagram :



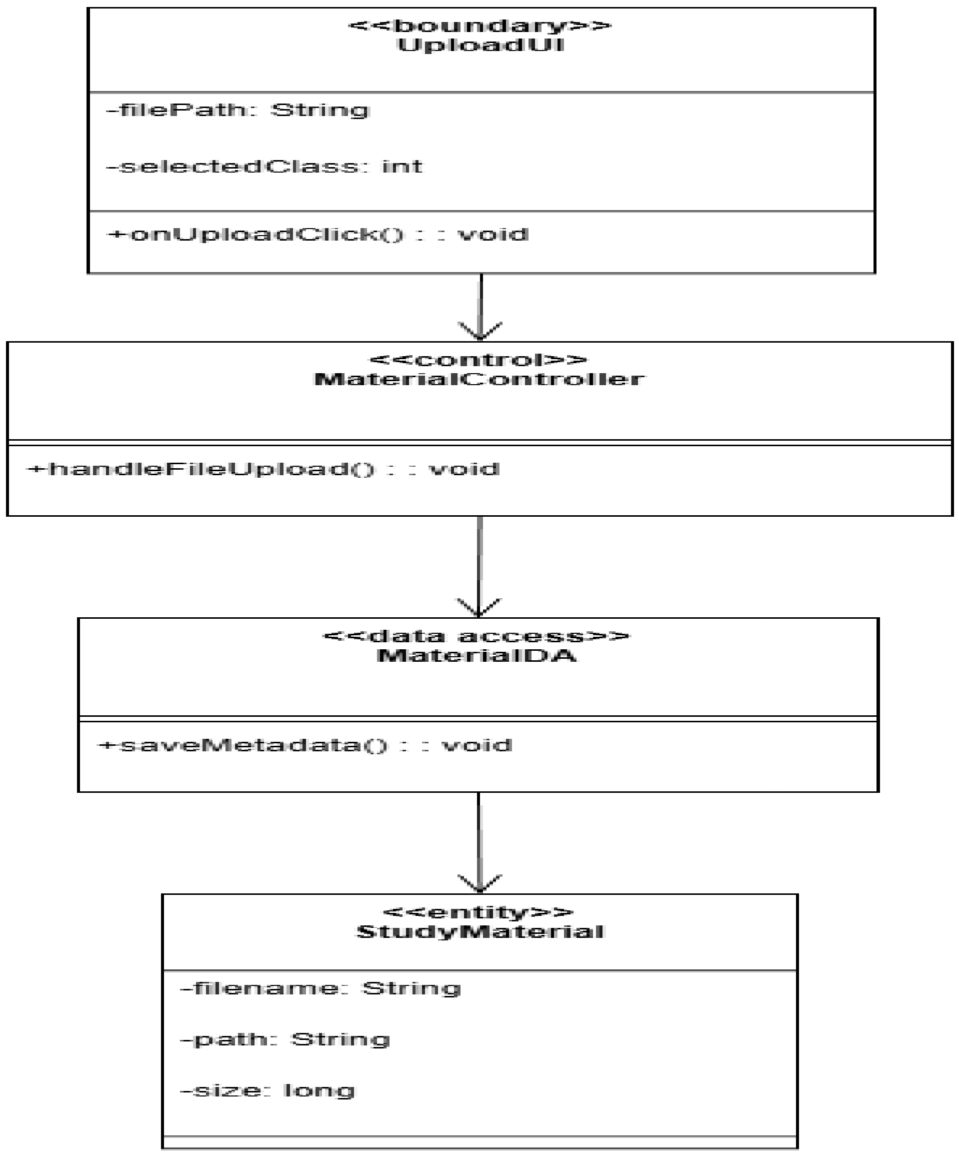
Updated Diagram :


<<boundary>>
AttendanceUI
+renderGrid(List students) : : void
+submit(List list) : : void
<<control>>
AttendanceController
+loadClassRoster(int classId) : : List
+processAttendance(int classId, List records) : : void
<<data access>>
AttendanceDA
+getEnrolledStudents(int classId) : : List
+saveBatch(List records) : : void
<<entity>>
AttendanceRecord
-recordID: int
-studentID: int
-isPresent: boolean
+markPresent() : : void
+markAbsent() : : void

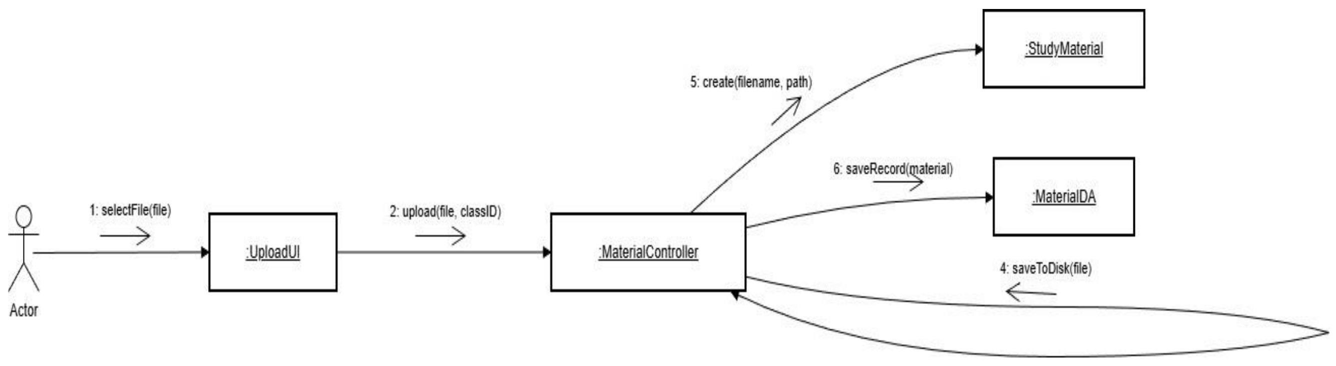
4. Use Case Name: Upload Study Materials (Moderate)

- **Scenario:** Teacher uploads a PDF or resource file for students to access.
- **Triggering event:** Teacher selects a file and clicks upload.
- **Brief description:** Handle binary file upload, save path to database, and create Material entity.
- **Actors:** Primary: Teacher. Secondary: Student (viewer).
- **Stakeholders:** Teacher; Student.
- **Related use cases:** Access Study Materials.
- **Preconditions:**
  - The teacher is logged in.
  - File format is supported (PDF, DOCX, JPG).
- **Postconditions:**
  - Study Material object created.
  - File stored in server/cloud storage.
- **Flow of activities:**
  - The teacher selects a file from their device, enters a title, and clicks "Confirm".
  - System (UploadView) sends file data to the controller.
  - System (MaterialController) validates file type and size.
  - System (MaterialController) initiates creation of the material record.
  - System (StudyMaterial) creates the object and sets the upload date to the current timestamp.
  - System confirms upload completion.
- **Exception conditions:**
  - File too large or wrong format → Show error message.

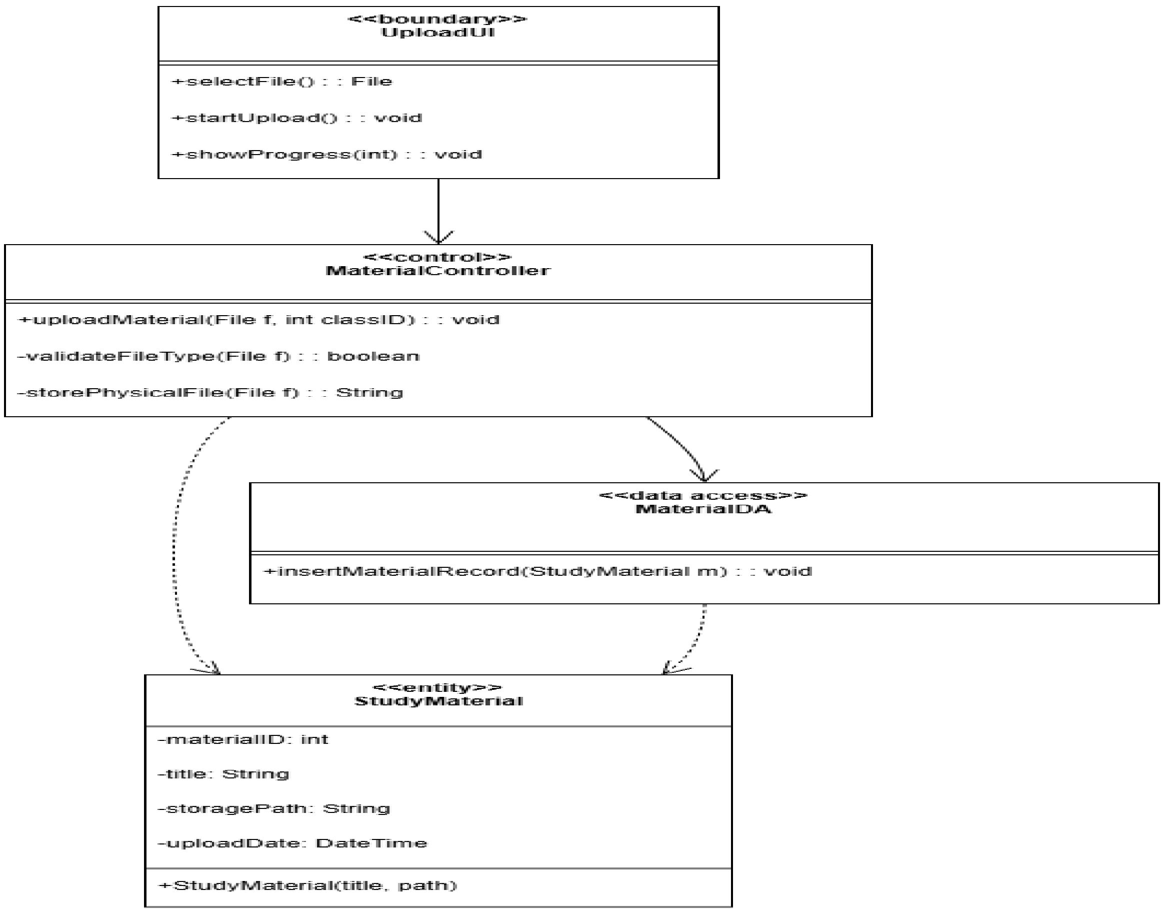
First Cut Design Class Diagram :



Communication Diagram :



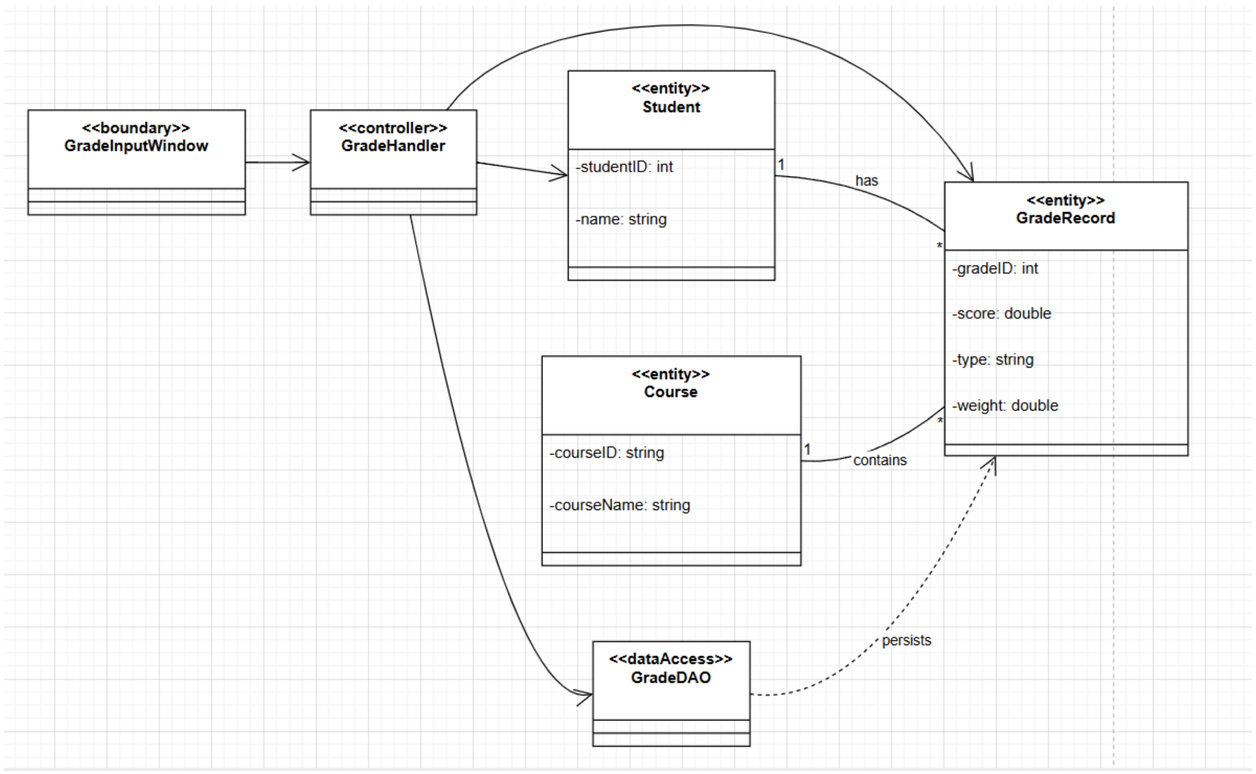
Updated Diagram :


<<boundary>>
UploadUI
+selectFile() : : File
+startUpload() : : void
+showProgress(int) : : void
<<control>>
MaterialController
+uploadMaterial(File f, int classID) : : void
-validateFileType(File f) : : boolean
-storePhysicalFile(File f) : : String
<<data access>>
MaterialDA
+insertMaterialRecord(StudyMaterial m) : : void
<<entity>>
StudyMaterial
-materialID: int
-title: String
-storagePath: String
-uploadDate: DateTime
+StudyMaterial(title, path)

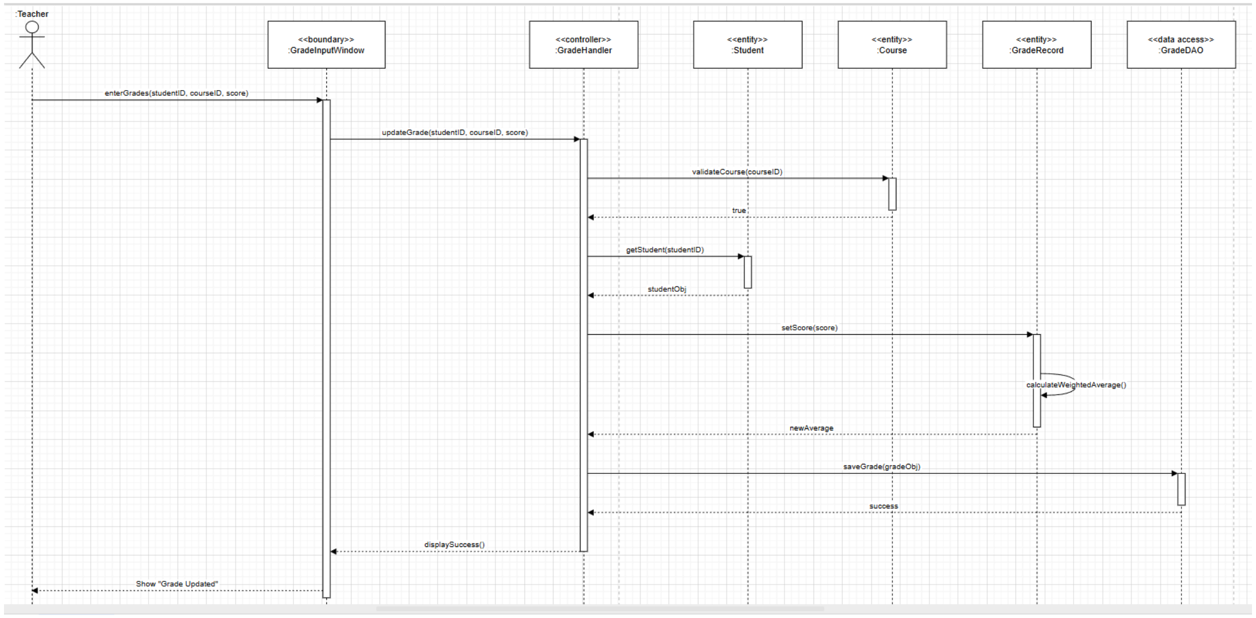
1. **Use Case Name: Enter or Update Grades (Complex)**

- **Scenario:** A teacher modifies a student's grade, triggering a recalculation of their weighted average.
- **Triggering event:** Teacher clicks "Submit Grade" on the grading interface.
- **Brief description:** Validates the course and student, updates the score in the grade record, triggers an internal weighted average recalculation, and persists the change.
- **Actors:** Primary: Teacher.
- **Preconditions:** Teacher is logged in; Course and Student exist.
- **Postconditions:** GradeRecord is updated; Weighted average is recalculated.
- **Flow of activities:**
    1. Teacher enters grade details in the **GradeInputWindow**.
    2. System validates the course and retrieves the student.
    3. System updates the **GradeRecord** and calculates the new average.
    4. System persists the data via **GradeDAO**.
    5. System displays a success message.
- **Exception conditions:** Invalid Course ID → Show error.

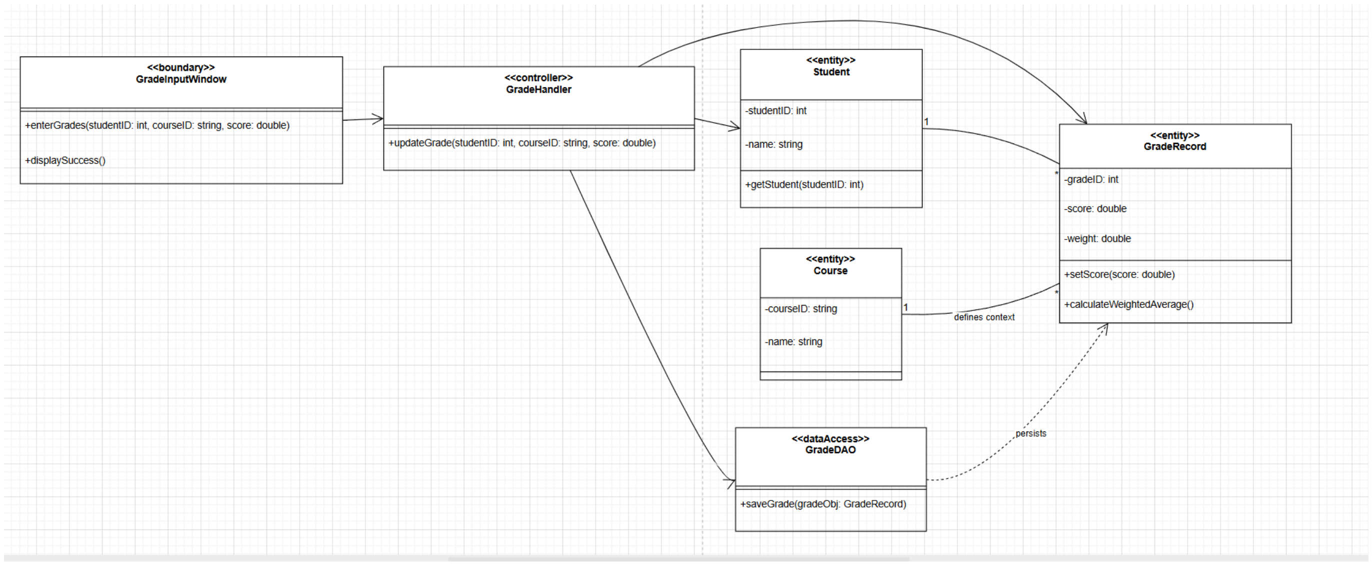
First Cut Design Class Diagram:



Sequence Diagram:



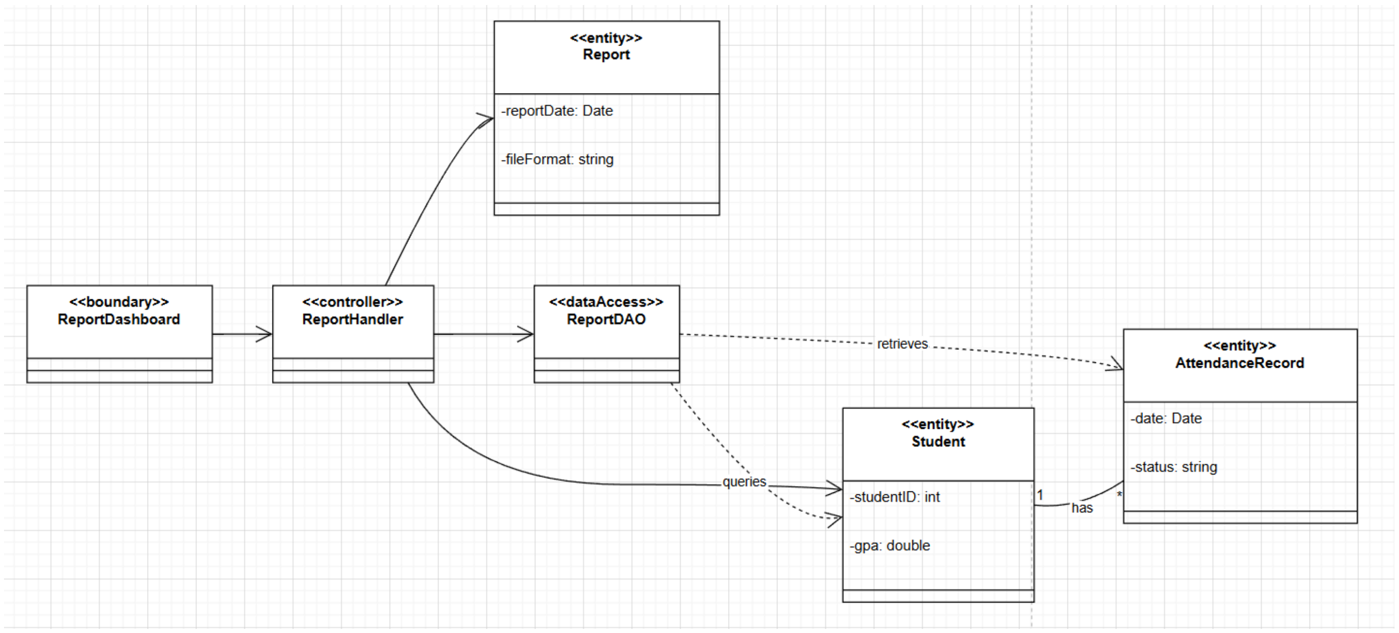
Updated Class Diagram:


<<boundary>>
GradeInputWindow
+enterGrades(studentID: int, courseID: string, score: double)
+displaySuccess()
<<controller>>
GradeHandler
+updateGrade(studentID: int, courseID: string, score: double)
<<entity>>
Student
-studentID: int
-name: string
+getStudent(studentID: int)
<<entity>>
Course
-courseID: string
-name: string
<<dataAccess>>
GradeDAO
+saveGrade(gradeObj: GradeRecord)
<<entity>>
GradeRecord
-gradeID: int
-score: double
-weight: double
+setScore(score: double)
+calculateWeightedAverage()
1
*
1
*
defines context
persists

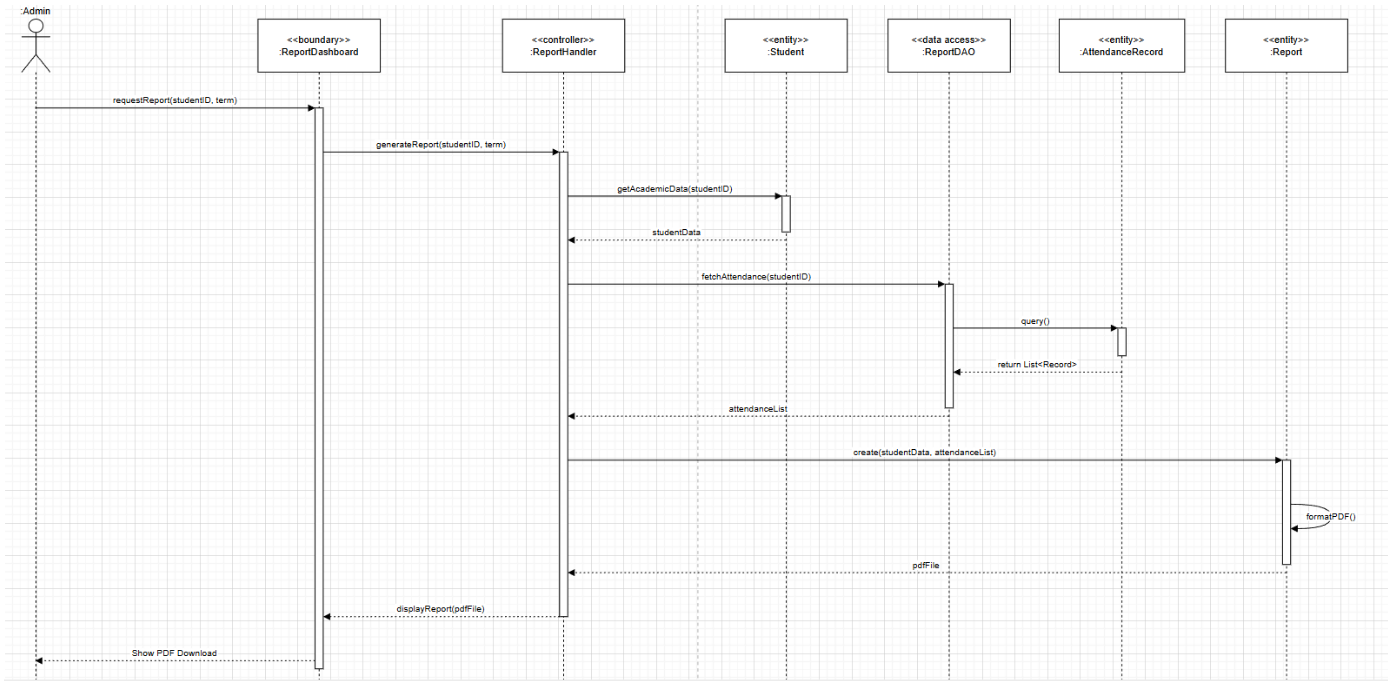
2. Use Case Name: Generate Reports (Complex)

- **Scenario:** Admin generates a student performance report (PDF) combining grades and attendance.
- **Triggering event:** Admin clicks "Generate Report".
- **Brief description:** Aggregates Student GPA and Attendance history, creates a Report object, and formats it as a PDF.
- **Actors:** Primary: Admin.
- **Preconditions:** Student and records exist.
- **Postconditions:** PDF Report generated and displayed.
- **Flow of activities:**
    1. Admin requests report in **ReportDashboard**.
    2. System retrieves **Student** academic data.
    3. System retrieves attendance list via **ReportDAO**.
    4. System instantiates **Report** and formats as PDF.
    5. System displays the PDF for download.

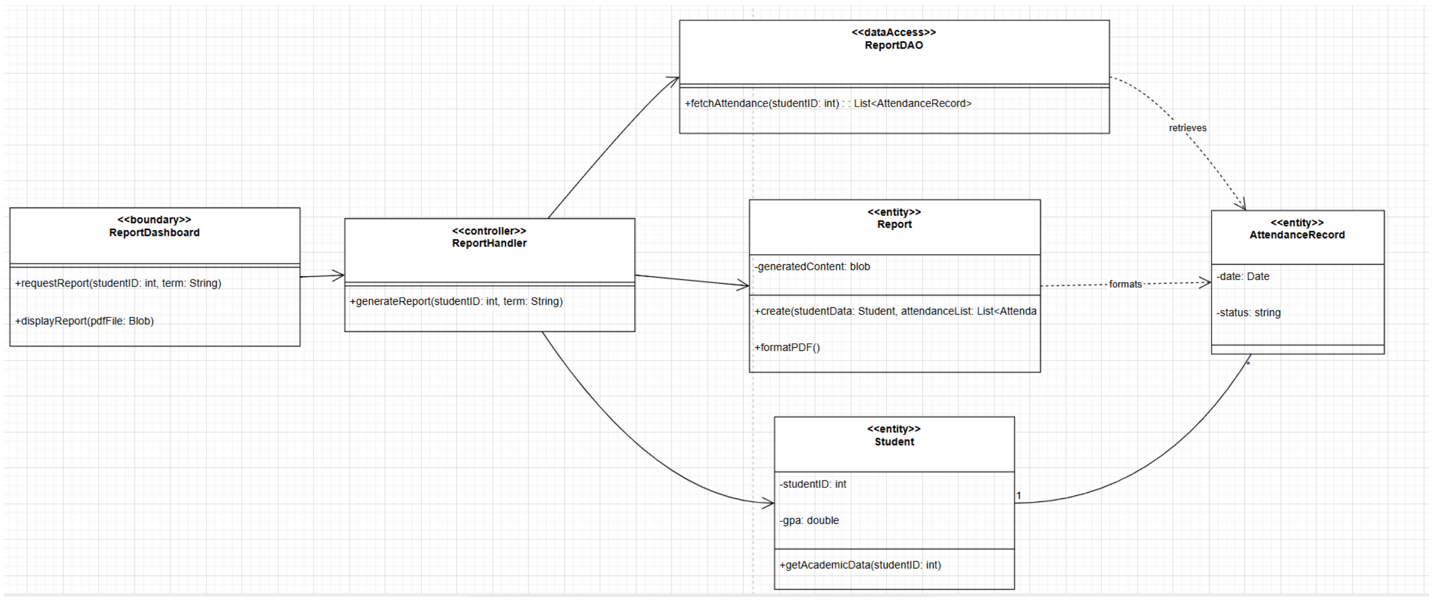
First Cut Design Class Diagram:



Sequence Diagram:



Updated Class Diagram:

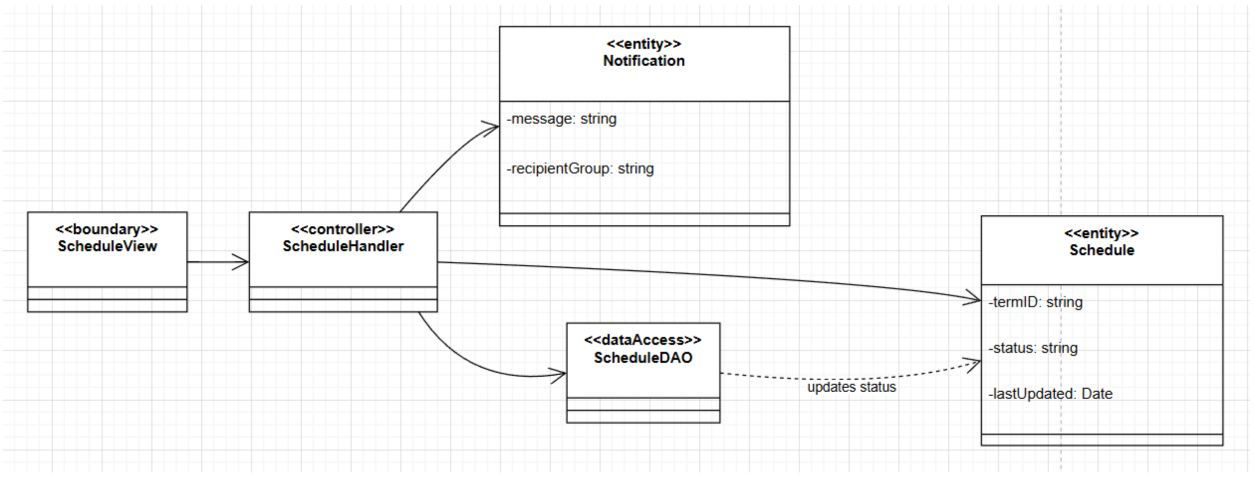


<<dataAccess>>
ReportDAO
+fetchAttendance(studentID: int) : : List<AttendanceRecord>
retrieves
<<boundary>>
ReportDashboard
+requestReport(studentID: int, term: String)
+displayReport(pdfFile: Blob)
<<controller>>
ReportHandler
+generateReport(studentID: int, term: String)
<<entity>>
Report
-generatedContent: blob
+create(studentData: Student, attendanceList: List<Attenda
+formatPDF()
formats
<<entity>>
AttendanceRecord
-date: Date
-status: string
*
<<entity>>
Student
-studentID: int
-gpa: double
+getAcademicData(studentID: int)
1

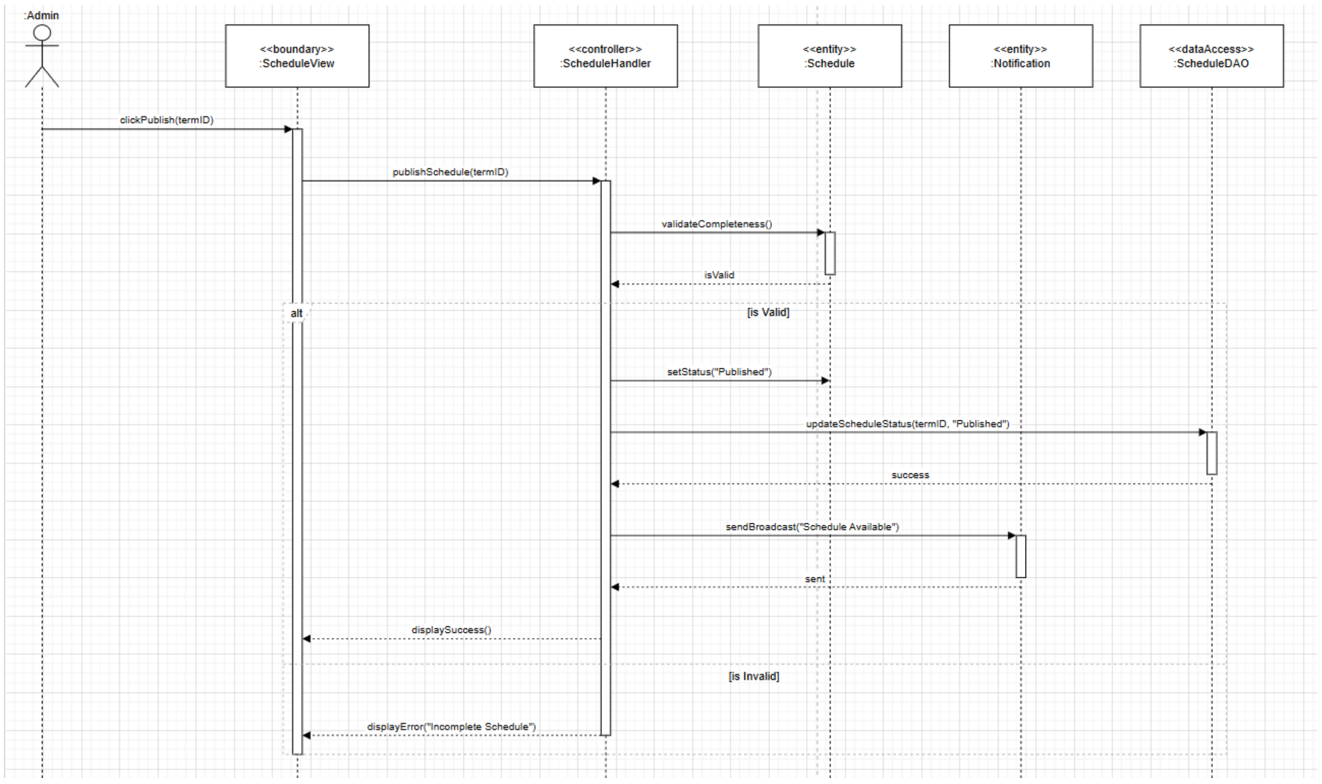
3. Use Case Name: Publish Schedule (Complex)

- **Scenario:** An administrator finalizes and broadcasts the term schedule.
- **Triggering event:** Admin selects "Publish Term".
- **Brief description:** Validates schedule completeness, updates status to "Published", and sends broadcast notifications.
- **Actors:** Primary: Admin.
- **Preconditions:** Draft schedule exists.
- **Postconditions:** Schedule status is "Published"; Notifications sent.
- **Flow of activities:**
    1. Admin requests publish in **ScheduleView**.
    2. System validates the **Schedule** completeness.
    3. System updates status and persists via **ScheduleDAO**.
    4. System sends broadcast via **Notification** entity.
    5. System displays success message.
- **Exception conditions:** Schedule incomplete → Show "Incomplete Schedule" error.

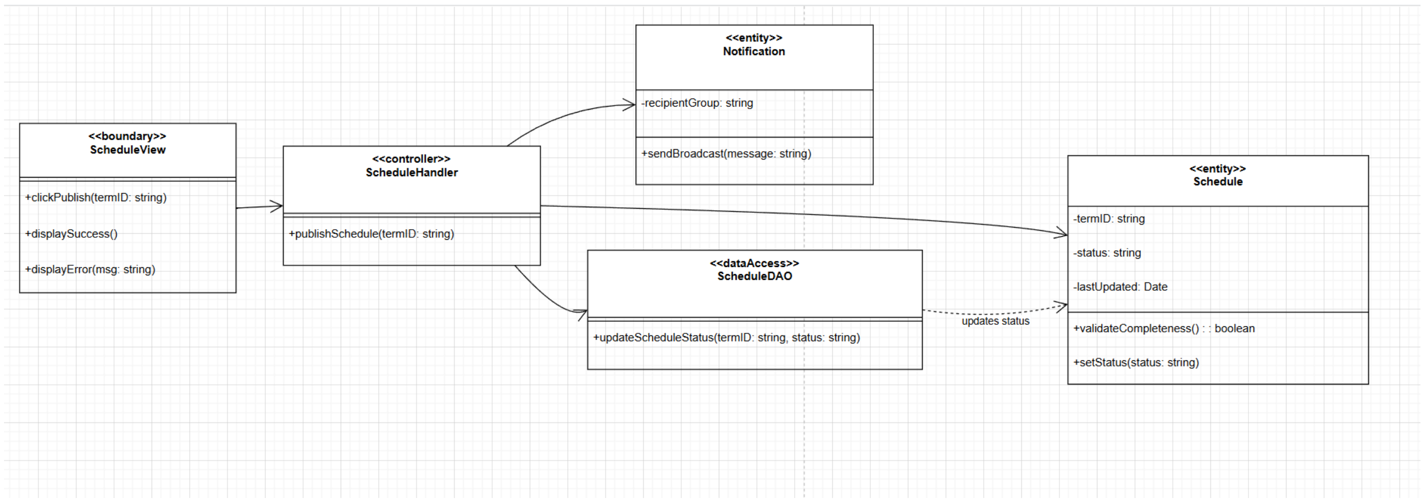
First Cut Design Class Diagram:



Sequence Diagram:



Updated Class Diagram:

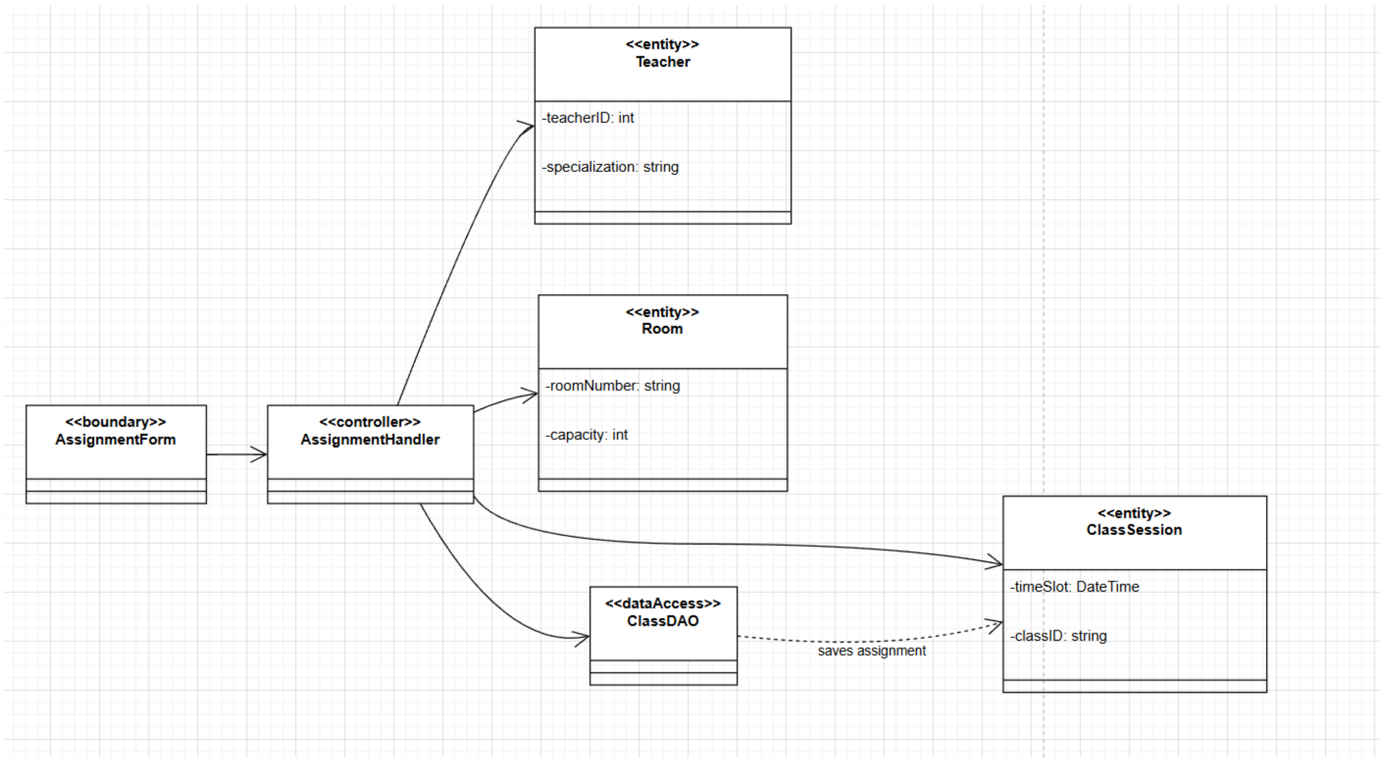


<<boundary>> ScheduleView
+clickPublish(termID: string)
+displaySuccess()
+displayError(msg: string)
<<controller>> ScheduleHandler
+publishSchedule(termID: string)
<<entity>> Notification
-recipientGroup: string
+sendBroadcast(message: string)
<<dataAccess>> ScheduleDAO
+updateScheduleStatus(termID: string, status: string)
<<entity>> Schedule
-termID: string
-status: string
-lastUpdated: Date
+validateCompleteness() : : boolean
+setStatus(status: string)
updates status

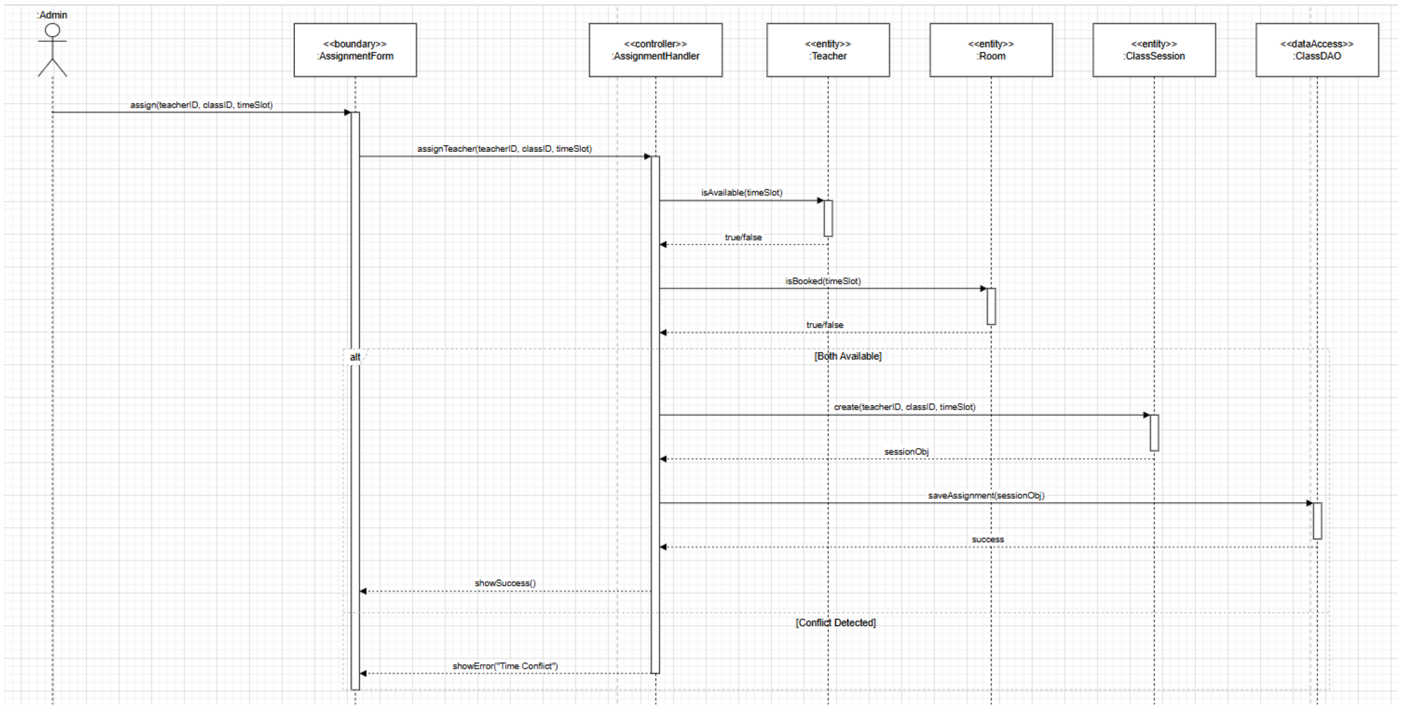
4. Use Case Name: Assign Teachers to Classes (Complex)

- **Scenario:** Admin assigns a teacher to a class session, handling conflicts.
- **Triggering event:** Admin submits assignment form.
- **Brief description:** Checks teacher availability and room booking status. If clear, creates a ClassSession and saves it.
- **Actors:** Primary: Admin.
- **Preconditions:** Teacher and Room exist.
- **Postconditions:** ClassSession created and saved.
- **Flow of activities:**
    1. Admin inputs details in **AssignmentForm**.
    2. System checks **Teacher** availability.
    3. System checks **Room** booking status.
    4. System creates **ClassSession** and saves via **ClassDAO**.
    5. System confirms assignment.
- **Exception conditions:** Conflict detected → Show "Time Conflict" error.

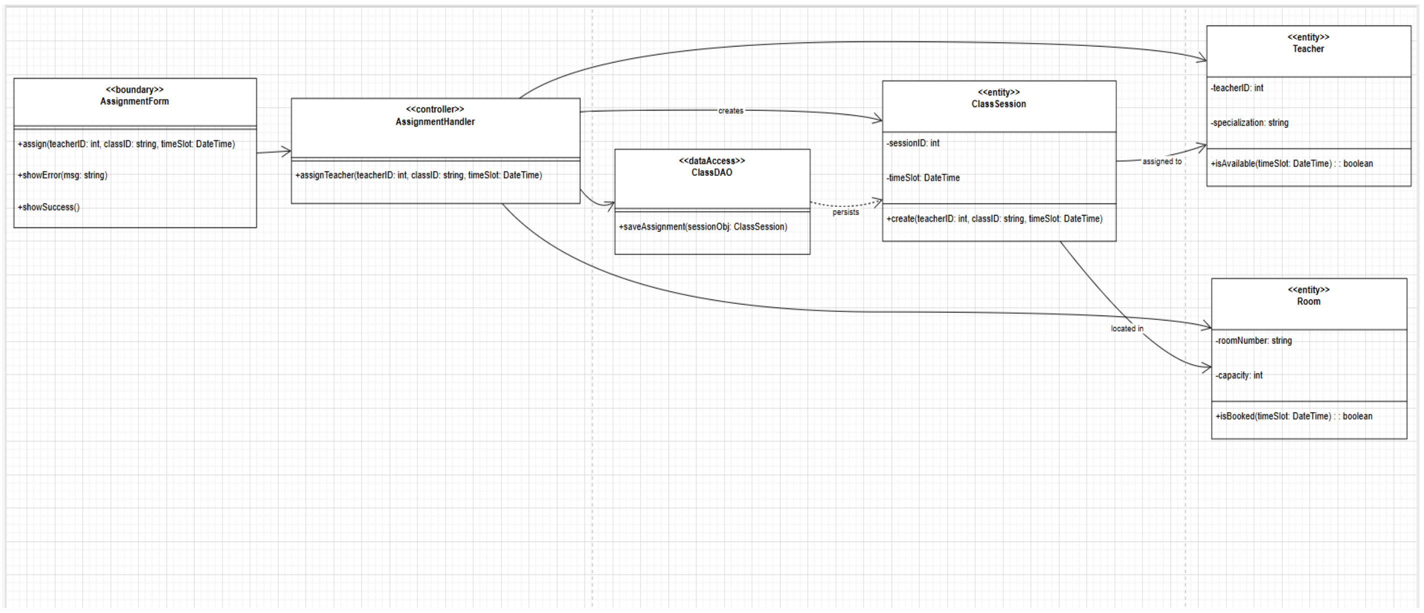
First Cut Design Class Diagram:



Sequence Diagram:



Updated Class Diagram:



<<boundary>>
AssignmentForm
+assign(teacherID: int, classID: string, timeSlot: DateTime)
+showError(msg: string)
+showSuccess()
<<controller>>
AssignmentHandler
+assignTeacher(teacherID: int, classID: string, timeSlot: DateTime)
creates
<<dataAccess>>
ClassDAO
+saveAssignment(sessionObj: ClassSession)
persists
<<entity>>
ClassSession
-sessionID: int
-timeSlot: DateTime
+create(teacherID: int, classID: string, timeSlot: DateTime)
assigned to
located in
<<entity>>
Teacher
-teacherID: int
-specialization: string
+isAvailable(timeSlot: DateTime) : : boolean
<<entity>>
Room
-roomNumber: string
-capacity: int
+isBooked(timeSlot: DateTime) : : boolean