Student Management System

Final Report

Development & Enterprise Application

May 2025

Group Members

Pasan Akalanka
R.S.D.Senarathna-32837
N.A.O.N .Perera-32098
K.L.I.Lakshani-32589
B.A.H.T.Kumari-33061
D.P.S.Senarethna-32814
K.E.I.Kovilawaththa-32500
K.L.Y.K.Lekamge-32476
H.M.L.P.Herath-32648
A.P.R.Jayananda-32927

Table Of Content

1. Problem Statement

▪ Describe the context: Why is an SMS needed? What problems does it solve (e.g., inefficient manual processes, lack of centralized data, difficulty tracking progress)?

▪ Clearly state the specific problem your unique feature addresses.

▪ Explain how your proposed system is a valuable solution.

▪ (Appendix: Maybe detailed scenarios of existing problems)

2. Requirements

▪ Target Audience: Define Admin, Teacher, Student roles and their needs.

▪ Functional Requirements: List the key features implemented (Student CRUD, Course Mgmt, Enrollment, Grading, Attendance, Unique Feature, etc.). Use bullet points or numbered lists.

▪ Non-Functional Requirements: Briefly mention usability, reliability (error handling), security considerations, performance (if relevant).

▪ Justification: Briefly explain why certain key features were included (linking back to the problem statement).

▪ (Appendix: Full detailed list of requirements, User Stories if used)

3. Design

▪ System Architecture: Explain your choice of MVC/3-Tier. Include a high-level architectural diagram.

▪ Component Interaction: Describe how Servlets, JSPs, DAOs, Services work together.

Include 1-2 key Sequence Diagrams embedded here.

▪ Data Structure: Include a snapshot/diagram of your Database Schema (ERD). Explain key tables and relationships.

▪ Code Structure: Describe your package structure (e.g., com.yourgroup.sms...). Include a key Class Diagram embedded here.

▪ Justification: Explain why this design (MVC, DAO pattern, DB schema) is appropriate for this enterprise application (maintainability, scalability, separation of concerns).

▪ (Appendix: Additional diagrams - more sequence, class, full ERD, use case)

4. Personal Reflection

▪ Individual Contribution: Briefly state your main contributions (ensure this aligns with Git history!).

▪ Teamwork & Process: How did the group collaborate? What worked well/poorly in the process (meetings, task allocation, code reviews)?

▪ Technologies Used: Reflect on using Servlets, JSP, JDBC, Tomcat, Git, etc. What were the challenges? What was learned? How does it compare to frameworks (if you have prior experience)?

▪ Lessons Learned: Key takeaways for future projects (technical or soft skills). This section is crucial for the Individual Contribution mark. Be honest and insightful.

5. Future Work

▪ Suggest concrete improvements and extensions (e.g., adding more modules like Library or Finance, API development for mobile app integration, advanced reporting/analytics, improved security measures, deployment to cloud).

▪ Discuss potential changes for better scalability (e.g., using JPA/Hibernate, optimizing

database queries, caching) and sustainability (e.g., better test coverage, CI/CD pipeline).

• Diagrams: Embed key diagrams directly in the report (resize appropriately). Reference appendices for

larger/additional diagrams. Ensure they are clean, readable, and use standard notation.

• Version Control Evidence: Mention the use of Git. Include the repository link (ensure it's accessible

to the marker if required). Briefly describe the branching strategy. Mention how commit logs show

individual contributions (can include a snapshot of representative logs in appendix if needed).

- Referencing: Choose a style (APA, Harvard) and apply it consistently for any external

libraries,

frameworks, major architectural patterns, or significant algorithms/concepts you researched and used.

Cite your sources in-text and provide a reference list. Also cite tools used for diagrams etc.

- Report Quality: Proofread carefully for grammar and spelling. Ensure clear language, logical flow,

and professional presentation. Use headings, subheadings, and potentially screenshots

(sparingly) to

enhance readability.

1. Problem Statement

The Student Management System (SMS) for NSBM is needed to provide a centralized, web based platform for efficiently managing student academic information, faculty data, course details, and related administrative processes. The current problems it solves include inefficient manual processes, lack of centralized data, and difficulty tracking progress, by streamlining operations and improving access to information for students, faculty members, and administrative staff.

The specific problems addressed by the unique features are:

- Personalized Student Dashboard: Lack of an immediate, relevant overview of a student's academic status upon login. The dashboard aggregates data such as current courses, attendance snippets, and relevant announcements for the logged-in student.
- System-Wide Announcements: Difficulty in disseminating information to targeted user groups. The system allows Admins to create, view, update, and delete announcements targeted to specific roles (All, Students, Faculty, Admin).

The proposed SMS is a valuable solution because it aims to develop a robust and user-friendly web application for managing academic records, providing distinct interfaces tailored to different user roles, implementing secure authentication and authorization, facilitating efficient recording and retrieval of grades and attendance, and introducing unique features like the personalized student dashboard and system-wide announcements to enhance user engagement and communication.

2. Requirements

Target Audience

The SMS is designed to serve three main user roles with distinct needs and functionalities:

Administrators: Staff responsible for overall system management, data entry, user account creation, and academic structure setup. They need comprehensive access to manage users, academic structures (Departments, Academic Terms, Locations, Programs, Courses), faculty and student profiles, course and faculty assignments, enrollments, manual creation of lecture sessions, academic records oversight (grades and attendance), and announcement management. They also operate the NFC Attendance Simulation interface.

- Faculty Members: Academic staff responsible for teaching courses, recording student grades, and managing attendance for their assigned lecture sessions. They need to view their own profile, view assigned courses and enrolled student lists, input and update grades, mark attendance for lecture sessions, and view relevant announcements.
- Students: Enrolled individuals who need to access their academic information, course details, grades, attendance, and system announcements. They need to view their personalized dashboard, which summarizes current courses, attendance snippets, and relevant announcements. They also need to view their profile, enrolled courses, grades, and detailed attendance records, and view relevant announcements.

Functional Requirements

The key features implemented in the SMS include:

- Authentication: Secure login and logout for all user roles.

- Authorization: Role-based access control for all functionalities.
- User Management (Admin): Create, view, update, and activate/deactivate user accounts for Students, Faculty, and other Administrators. Assign appropriate roles.
- Academic Structure Management (Admin): Define and manage Departments, Academic Terms, Locations, Programs, and Courses.

  Faculty & Student Profile Management (Admin): Create and manage comprehensive profiles.
- Course & Faculty Assignment (Admin): Assign Faculty members to teach specific Courses.
- Enrollment Management (Admin): Enroll Students into Courses for specific Academic Terms.
- Lecture Session Creation (Admin): Manually create Lecture Session records.
- Academic Records Oversight (Admin): View and modify grades and attendance records.
- Announcement Management (Admin): Create, view, update, and delete system-wide Announcements, with role-based targeting.
- NFC Attendance Simulation (Admin/Operator): Access and operate the simulation interface to mark student attendance.
- Faculty Assigned Courses & Students (Faculty): View assigned courses and enrolled student lists.
- Grade Management (Faculty): Input, view, and update grades for students in assigned courses.
- Attendance Management (Faculty): View scheduled Lecture Sessions and mark student attendance.

- Personalized Student Dashboard (Student Unique Feature): View a dashboard summarizing current courses, attendance snippets, and relevant announcements upon login.
- Student Profile & Academic Records Viewing (Student): View profile, enrolled courses, grades, and detailed attendance records.

  Announcement Viewing (All Users): View announcements targeted to their role or 'ALL'.

Non-Functional Requirements

The SMS also addresses several non-functional requirements:

- Usability: The user interface shall be intuitive and easy to navigate, with clear and helpful error messages.
- Reliability (Error Handling): The system shall handle common errors gracefully without crashing, displaying appropriate error pages.
- Security: Passwords shall be securely hashed and salted, protected against SQL injection (using PreparedStatements), and include input validation on all usersubmitted data. Proper authorization checks shall be implemented.
- Performance: Web pages should load within an acceptable time frame under normal load for a student project.
- Maintainability: Code shall be well-commented, organized, and follow Java coding conventions.
- Compatibility: The system should be accessible via modern web browsers.

3. Design

Based on the project description, the SMS follows a layered architectural design, specifically aligning with the Model-View-Controller (MVC) pattern, which can be seen as a variation of a 3-Tier architecture.

System Architecture: MVC/3-Tier

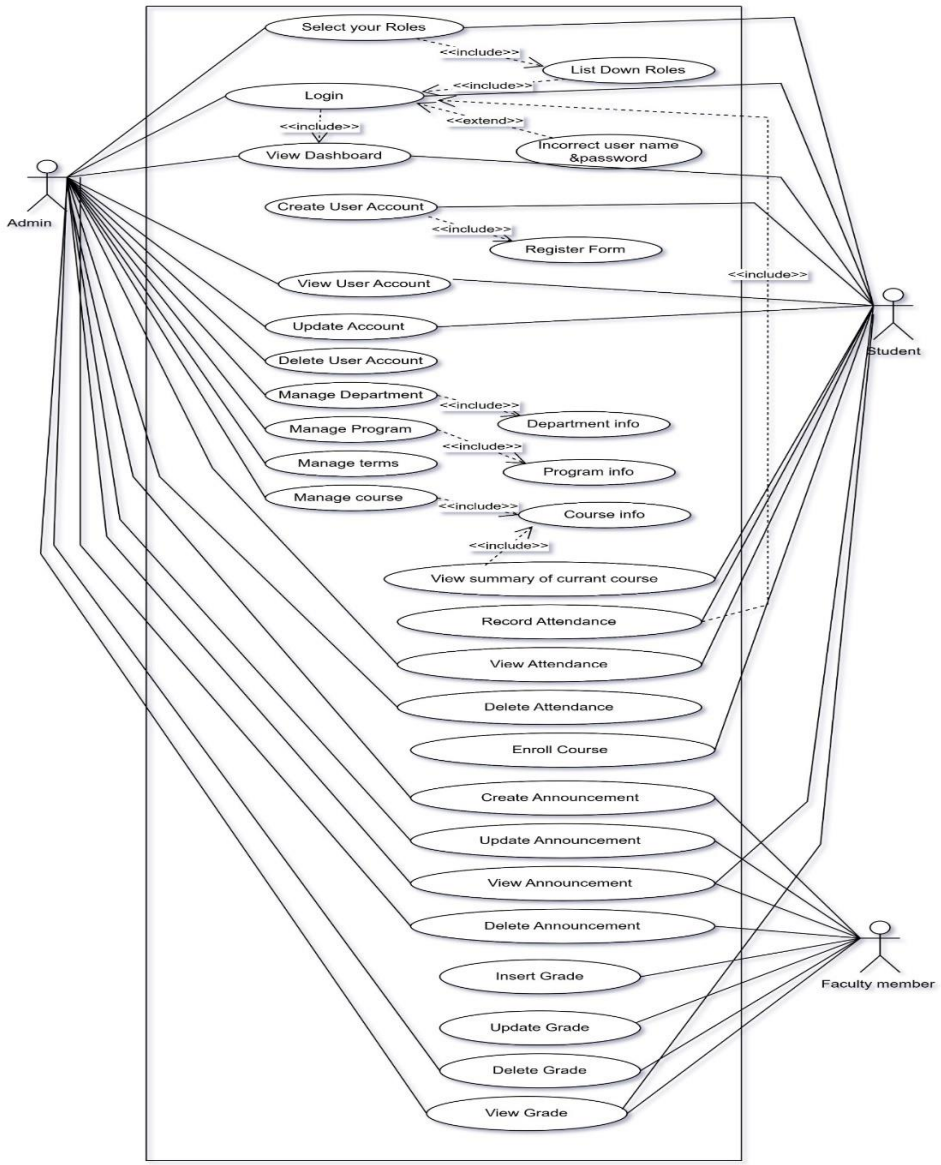
The system is designed following the Model-View-Controller (MVC) pattern. This separates the application into three interconnected parts:

- Model: Represents the data and business logic. In this system, the Model consists of the POJOs (Plain Old Java Objects) representing the data entities (like Student, Course, Attendance) and the DAOs (Data Access Objects) which handle all database interactions . Optional Service classes are included to encapsulate more complex business logic .
- View: Responsible for presenting the data to the user and handling user input. This is primarily handled by JSP (JavaServer Pages) files, which render the user interface .
- Controller: Acts as an intermediary between the Model and the View. Servlets serve as the controllers, receiving user requests, performing initial input validation, interacting with the Model (DAOs or Services) to process the request, and selecting the appropriate View (JSP) to render the response .

This architecture promotes:

- Separation of Concerns: Each component has a specific responsibility, making the code easier to understand, develop, and maintain.
- Modularity: Components can be developed and tested independently.

- Flexibility: Changes to the view do not necessarily affect the model or controller, and vice versa.
- Testability: The separation of business logic in the Model makes it easier to write unit tests.



Component Interaction

The interaction between the components follows a typical request-response flow within the MVC pattern:

1. User Interaction: The user interacts with the application through a web page (a JSP).
   This interaction could be clicking a link or submitting a form.

2. Request to Controller: The user's action sends an HTTP request to a designated Servlet.

The web.xml file or annotations map specific URL patterns to the appropriate Servlets .

3. Servlet Processing: The Servlet receives the request, extracts any necessary data (like form parameters) . It performs initial input validation .

4. Business Logic (Optional Service Layer): For complex operations or business rules, the Servlet might call a method in a Service class . The Service class orchestrates the necessary actions, potentially involving multiple DAOs.

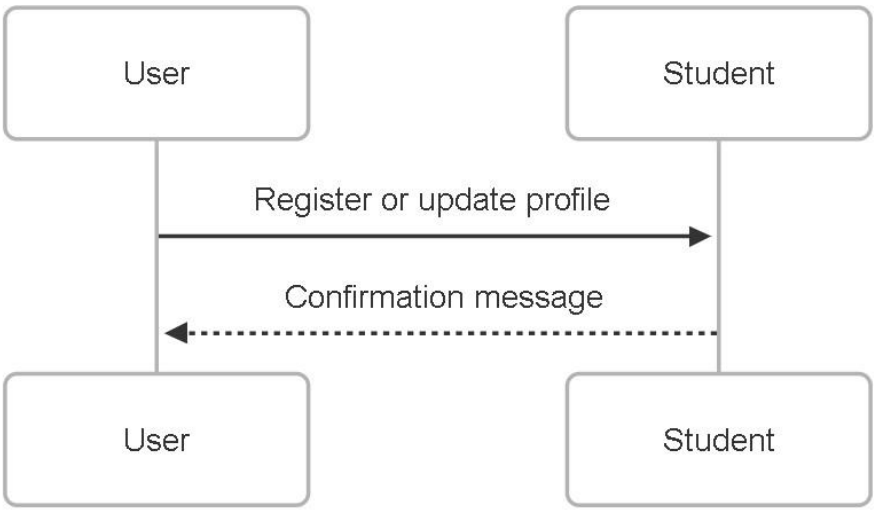
5. Data Access: The Service class or the Servlet (for simpler operations) interacts with the DAOs. The DAOs contain the logic to connect to the Database and execute SQL queries using JDBC to retrieve, insert, update, or delete data . DAOs return data as POJOs or lists of POJOs.

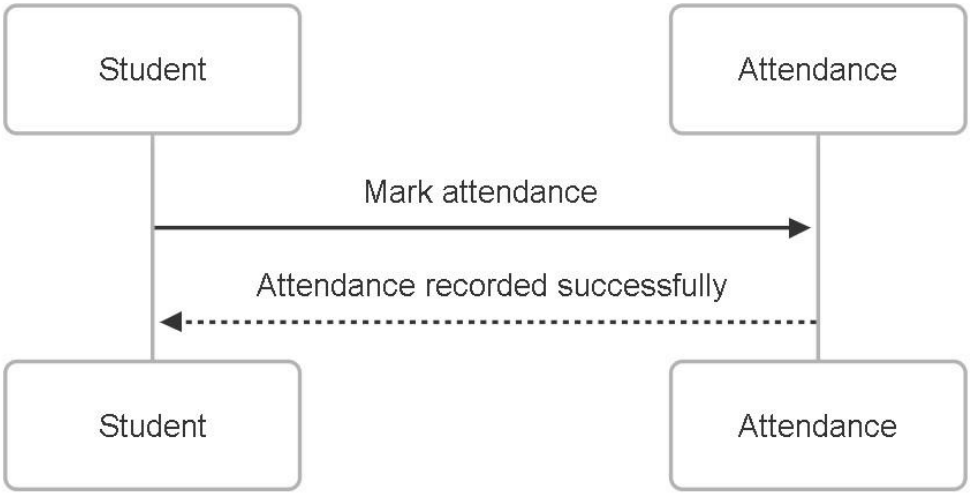
6. Data Preparation for View: The Service or Servlet receives data from the DAOs (in the form of POJOs). The Servlet then prepares the data to be displayed by the View, typically by setting attributes in the HttpServletRequest object .

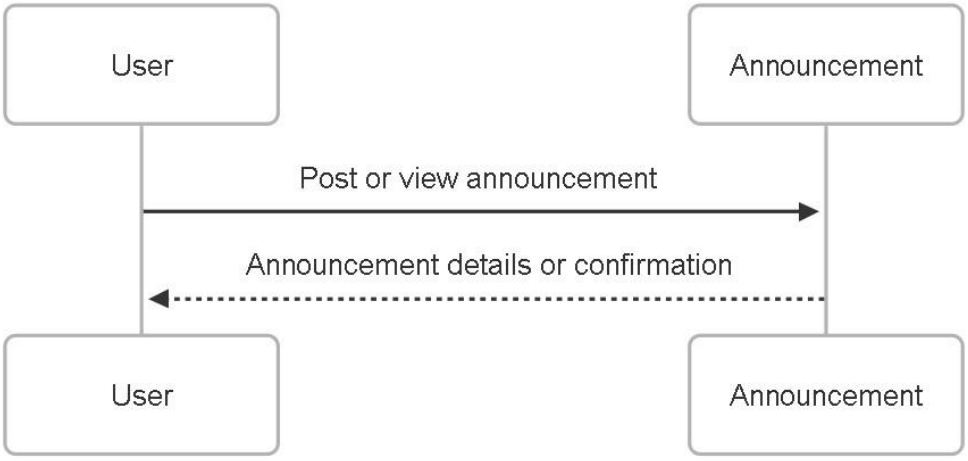
7. Forward to View: The Servlet forwards the request and response objects to the appropriate JSP page .

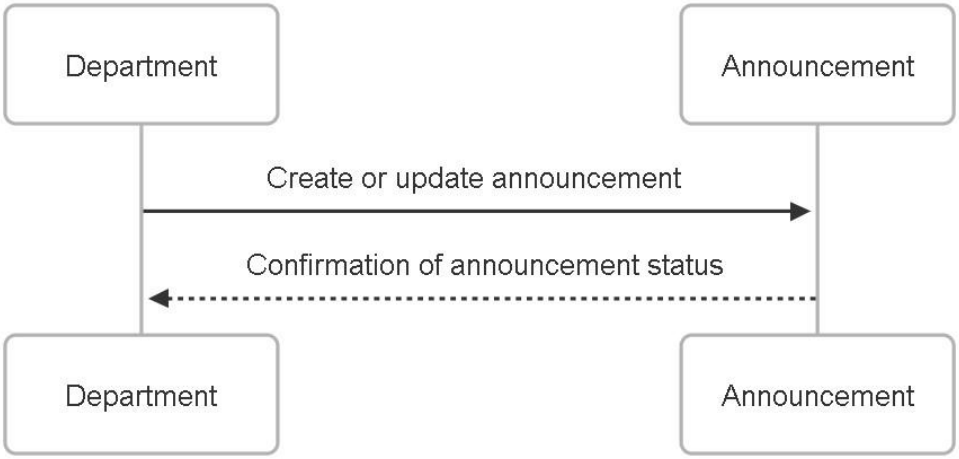
8. Rendering the View: The JSP page uses Expression Language (EL) and JSTL (JSP Standard Tag Library) to access the data set in the request attributes by the Servlet and dynamically generate the HTML content that is sent back to the user's browser .

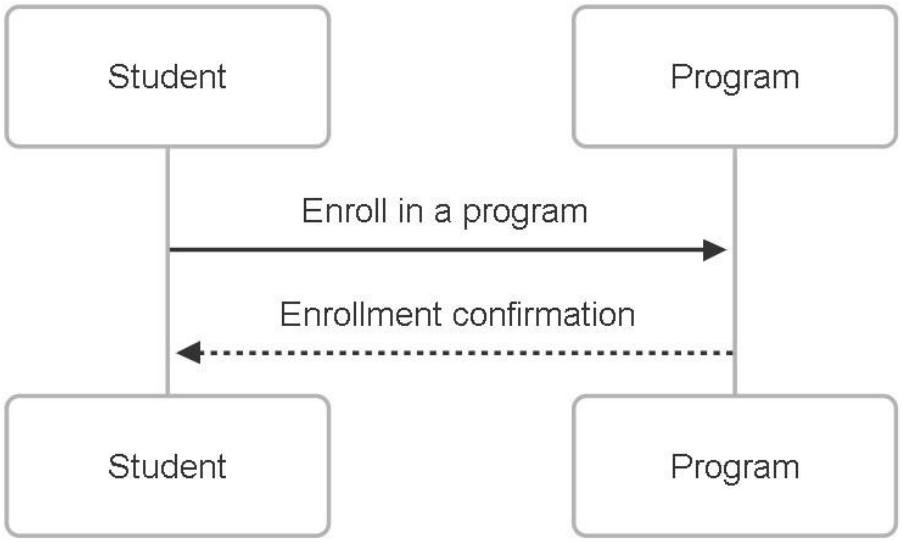
Sequence Diagrams:

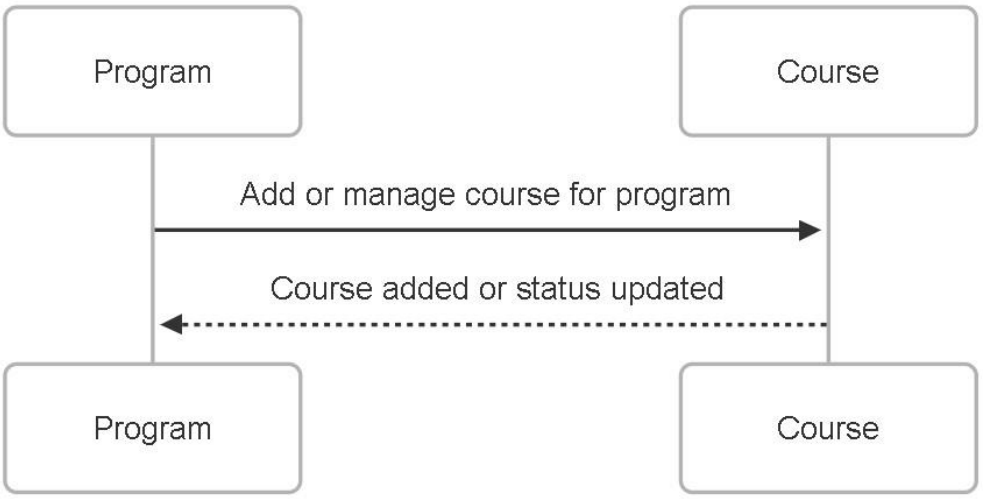


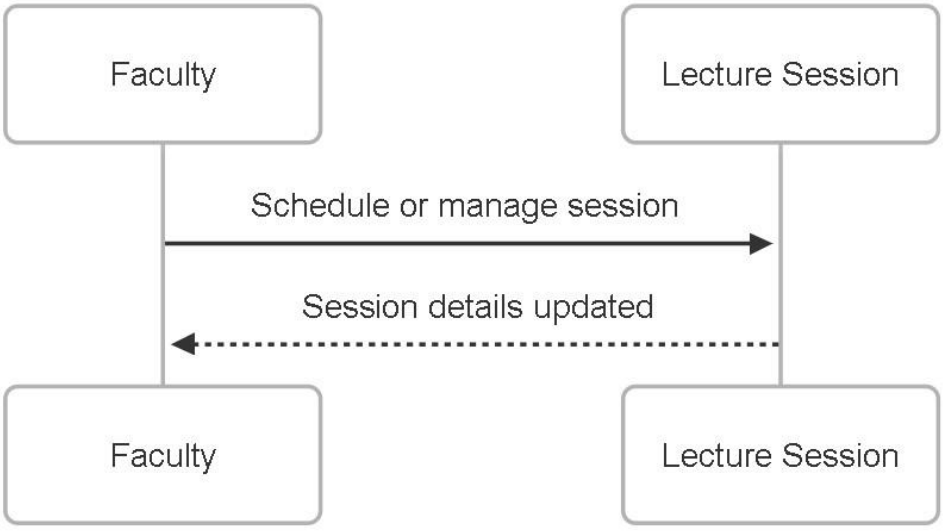


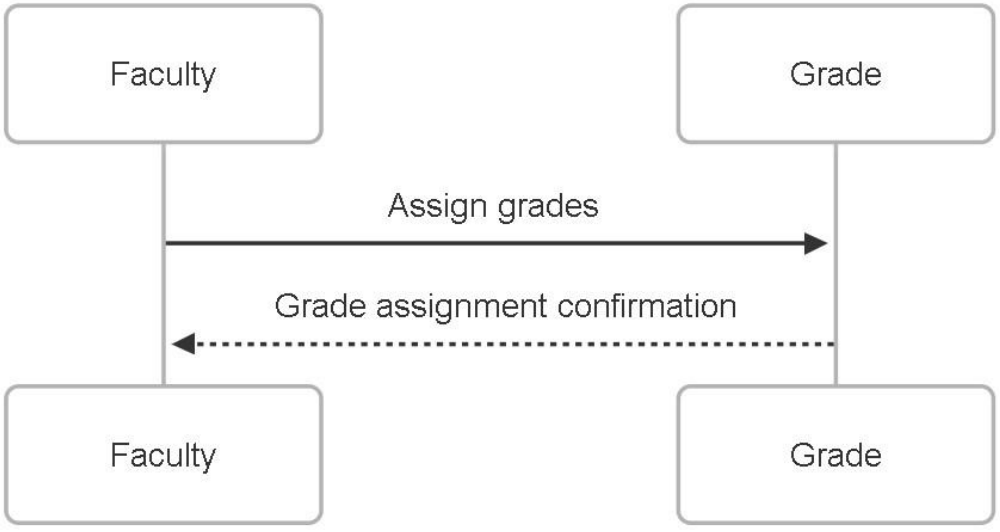

User
Announcement
Post or view announcement
Announcement details or confirmation
User
Announcement

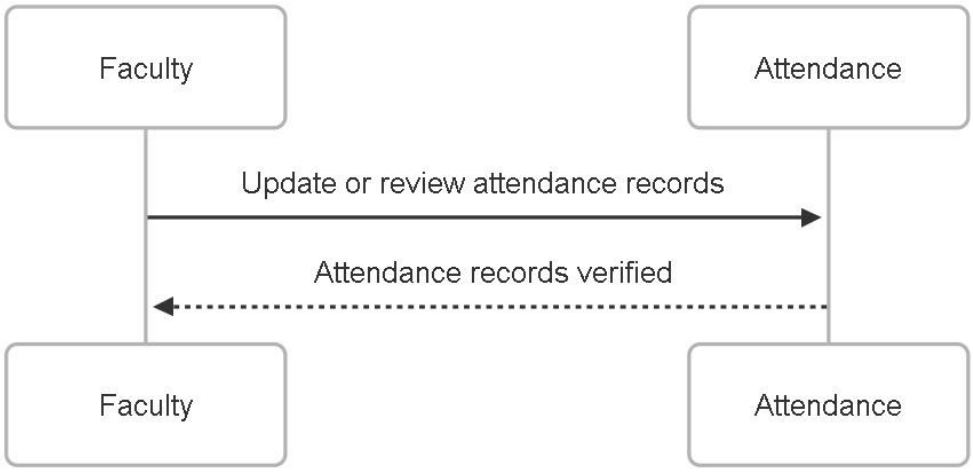

Department
Announcement
Create or update announcement
Confirmation of announcement status
Department
Announcement

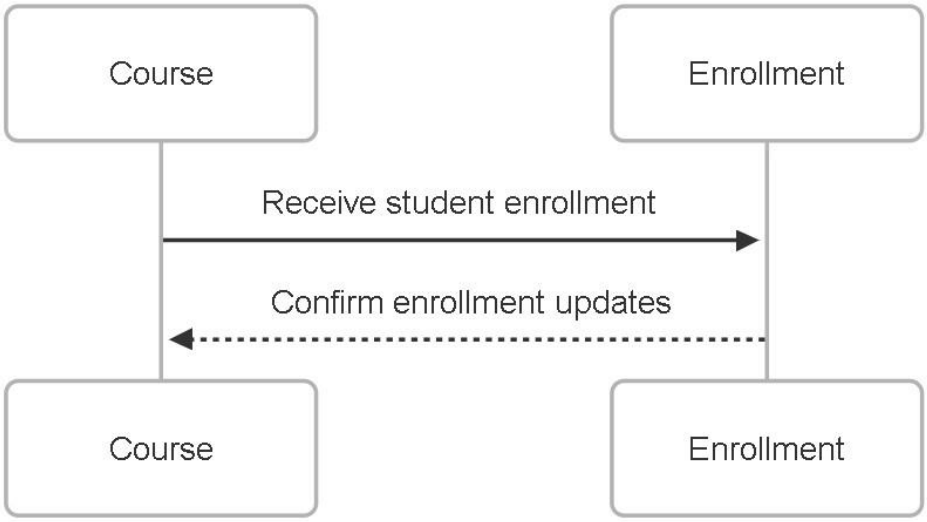

Student
Program
Enroll in a program
Enrollment confirmation
Student
Program

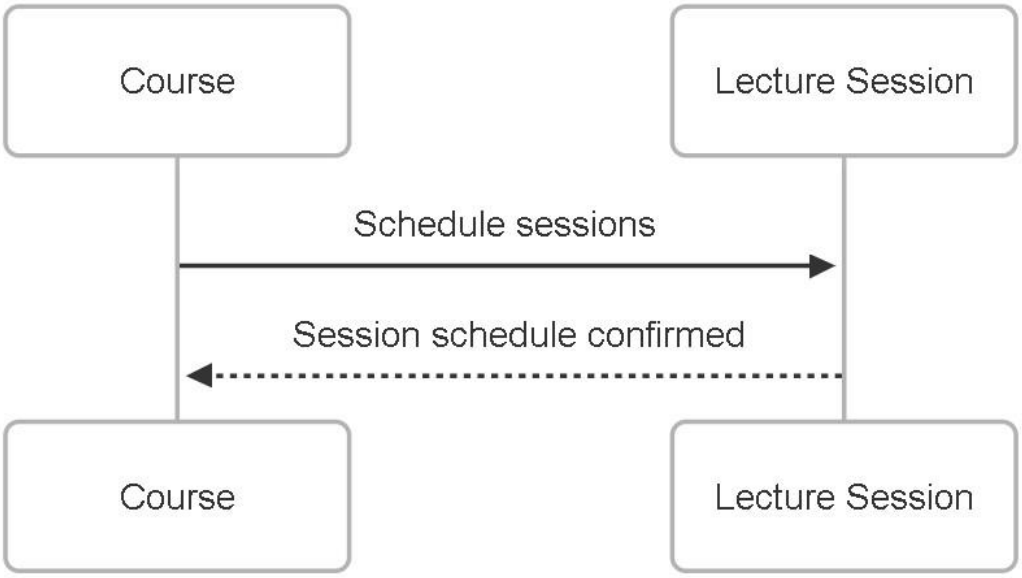

Program
Course
Add or manage course for program
Course added or status updated
Program
Course

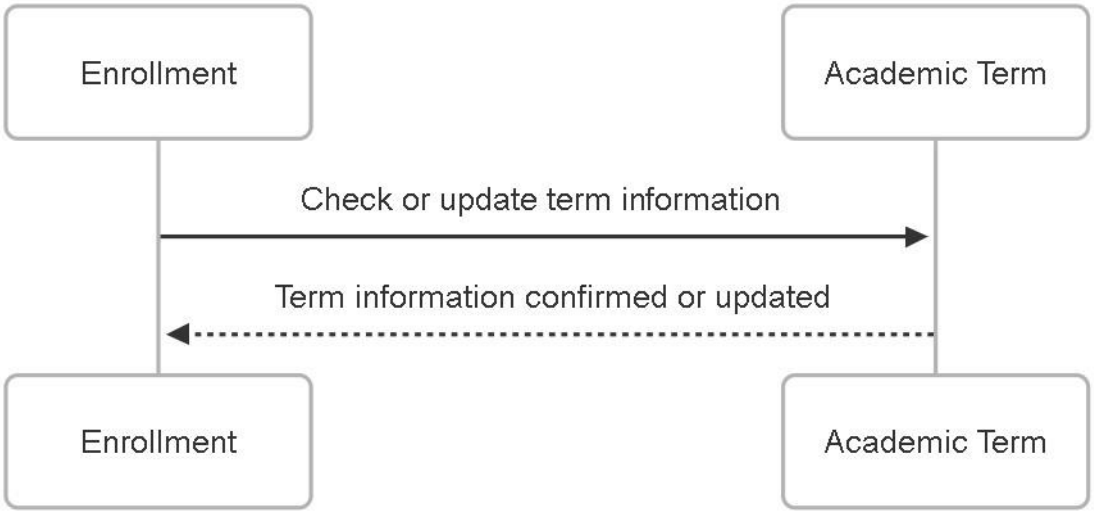

Faculty
Lecture Session
Schedule or manage session
Session details updated
Faculty
Lecture Session

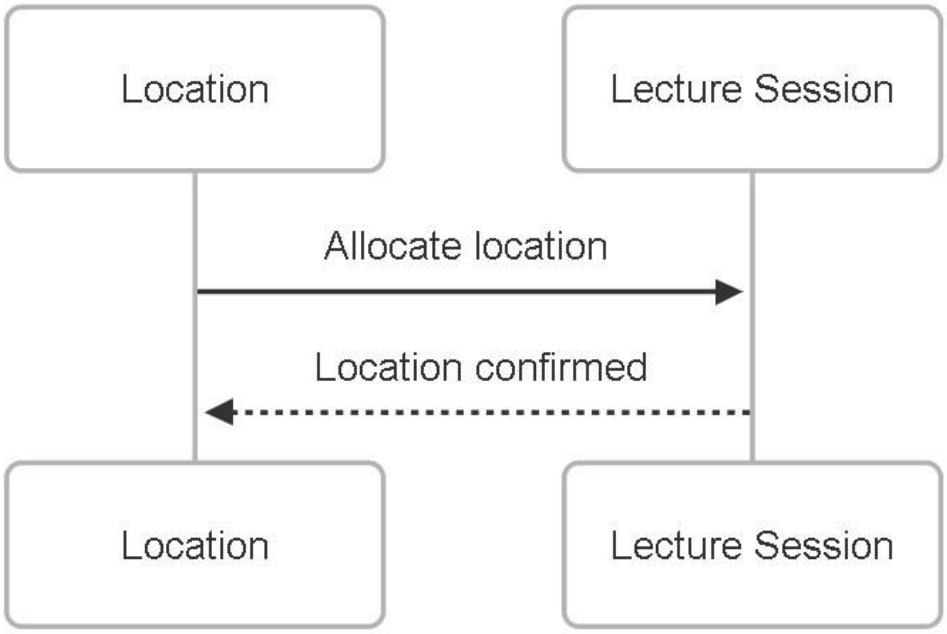

Faculty
Grade
Assign grades
Grade assignment confirmation
Faculty
Grade

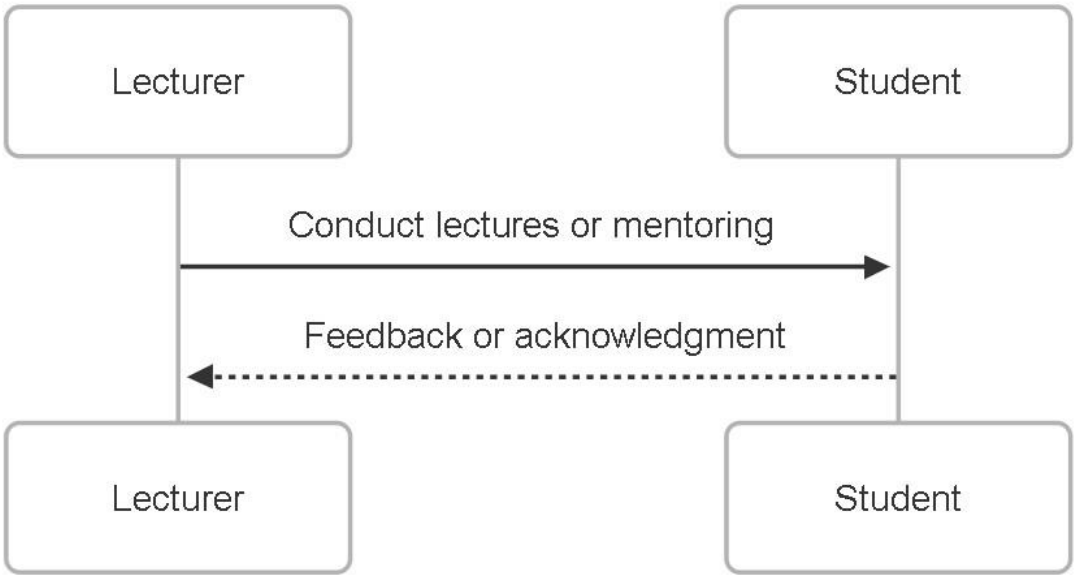

Faculty
Attendance
Update or review attendance records
Attendance records verified
Faculty
Attendance


Course
Enrollment
Receive student enrollment
Confirm enrollment updates
Course
Enrollment

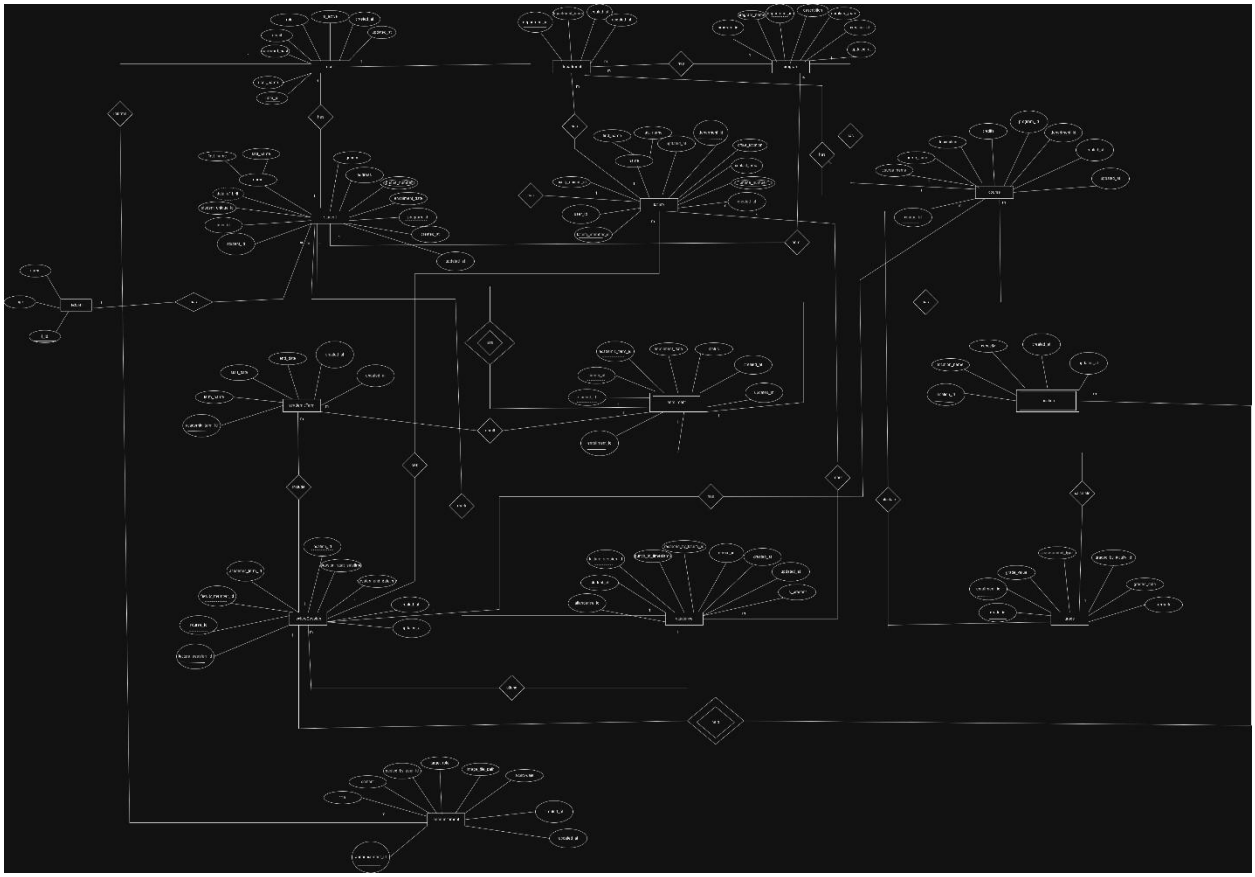

Course
Lecture Session
Schedule sessions
Session schedule confirmed
Course
Lecture Session


Enrollment
Academic Term
Check or update term information
Term information confirmed or updated
Enrollment
Academic Term


Location
Lecture Session
Allocate location
Location confirmed
Location
Lecture Session


Lecturer
Student
Conduct lectures or mentoring
Feedback or acknowledgment
Lecturer
Student

ER Diagram



A breakdown of the key entities and their relationships:

- Users Table: Stores user authentication and role information.
    - user_id (PK)
    - username, password_hash, email, role, is_active, created_at, updated_at
        - Relationships:
            - One-to-One with Students table on user_id: A user can be a student.
            - One-to-One with Faculty table on user_id: A user can be a faculty member.

- One-to-Many with Announcements table on user_id: A user (Admin) can post multiple announcements.

- Departments Table: Represents the faculties or departments within the institution.
  - department_id (PK) ○ department_name, created_at, updated_at ○ Relationships:
    - One-to-Many with Programs table on department_id: A department can offer multiple programs.
    - One-to-Many with Courses table on department_id: A department can offer multiple courses.
    - One-to-Many with Faculty table on department_id: A department can have multiple faculty members.

- Programs Table: Stores information about degree or diploma programs.
  - program_id (PK) ○ program_name, department_id (FK), description, duration_years, created_at, updated_at ○ Relationships:
    - Many-to-One with Departments table on department_id: Multiple programs belong to one department.
    - One-to-Many with Students table on program_id: A program can have multiple students enrolled.
    - One-to-Many with Courses table on program_id: A program can include multiple courses.

- Students Table: Contains details about the students.
  - student_id (PK) ○ user_id (FK), student_unique_id, first_name, last_name, date_of_birth, gender, address, phone_number, enrollment_date, program_id (FK), created_at, updated_at

- Relationships:
  - One-to-One with Users table on user_id: Links a student record to a user account.
  - Many-to-One with Programs table on program_id: Multiple students can be in the same program.
  - One-to-Many with Enrollments table on student_id: A student can have multiple enrollments in courses.
  - One-to-Many with Attendance table on student_id: A student can have multiple attendance records.

- Faculty Table: Stores details about the faculty members. ○ faculty_member_id (PK) ○ user_id (FK), faculty_unique_id, first_name, last_name, department_id (FK), office_location, contact_email, phone_number, created_at, updated_at ○ Relationships:
  - One-to-One with Users table on user_id: Links a faculty record to a user account.
  - Many-to-One with Departments table on department_id: Multiple faculty members belong to one department.
  - One-to-Many with LectureSessions table on faculty_member_id: A faculty member can teach multiple lecture sessions.
  - One-to-Many with Attendance table on recorded_by_faculty_id: A faculty member can record attendance for multiple sessions.
  - One-to-Many with Grades table on graded_by_faculty_id: A faculty member can grade multiple assessments.

- Courses Table: Contains information about the modules or courses offered.

- course_id (PK) ○ course_code, course_name, description, credits, program_id (FK), department_id (FK), created_at, updated_at ○ Relationships:
  - Many-to-One with Programs table on program_id: Multiple courses can belong to a program.
  - Many-to-One with Departments table on department_id: Multiple courses are offered by a department.
  - One-to-Many with Enrollments table on course_id: A course can have multiple student enrollments.
  - One-to-Many with LectureSessions table on course_id: A course can have multiple lecture sessions.

- AcademicTerms Table: Defines academic periods like semesters. ○ academic_term_id (PK) ○ term_name, start_date, end_date, created_at, updated_at ○ Relationships:
  - One-to-Many with Enrollments table on academic_term_id: An academic term can have multiple enrollments.
  - One-to-Many with LectureSessions table on academic_term_id: An academic term can have multiple lecture sessions.

10. Enrollments Table: Records which student is enrolled in which course for a specific academic term.
    - enrollment_id (PK) ○ student_id (FK), course_id (FK), academic_term_id (FK), enrollment_date, status, created_at, updated_at

Relationships:

- Many-to-One with Students table on student_id: Multiple enrollments belong to one student.
- Many-to-One with Courses table on course_id: Multiple enrollments are for one course.
- Many-to-One with AcademicTerms table on academic_term_id: Multiple enrollments are within one academic term.
- One-to-Many with Grades table on enrollment_id: An enrollment can have multiple grades (e.g., for different assessments).

10. Locations Table: Lists the physical locations for lectures or sessions.
    - location_id (PK) ○ location_name, capacity, created_at, updated_at ○

      Relationships:

      - One-to-Many with LectureSessions table on location_id: A location can host multiple lecture sessions.

10. LectureSessions Table: Represents specific instances of lectures.
    - lecture_session_id (PK)

      course_id (FK), faculty_member_id (FK), academic_term_id (FK), location_id (FK), session_start_datetime, session_end_datetime, created_at, updated_at ○

      Relationships:

      - Many-to-One with Courses table on course_id: Multiple lecture sessions are for one course.
      - Many-to-One with Faculty table on faculty_member_id: Multiple lecture sessions are taught by one faculty member.

- Many-to-One with AcademicTerms table on academic_term_id: Multiple lecture sessions are within one academic term.
- Many-to-One with Locations table on location_id: Multiple lecture sessions take place at one location.
- One-to-Many with Attendance table on lecture_session_id: A lecture session can have multiple attendance records (one for each student present).

- Attendance Table: Records student attendance for specific lecture sessions.
  - attendance_id (PK)

    student_id (FK), lecture_session_id (FK), punch_in_timestamp, is_present, recorded_by_faculty_id (FK), device_id, created_at, updated_at ○ Relationships:

    - Many-to-One with Students table on student_id: Multiple attendance records are for one student.
    - Many-to-One with LectureSessions table on lecture_session_id: Multiple attendance records are for one lecture session.
    - Many-to-One with Faculty table on recorded_by_faculty_id: Attendance can be recorded by a specific faculty member.

- Grades Table: Stores student grades for assessments. ○ grade_id (PK) ○ enrollment_id (FK), grade_value, assessment_type, graded_by_faculty_id (FK), graded_date, remarks ○ Relationships:
  - Many-to-One with Enrollments table on enrollment_id: Multiple grades can be associated with a single enrollment (e.g., different assessments).

- Many-to-One with Faculty table on graded_by_faculty_id: Multiple grades can be entered by one faculty member.

- Announcements Table: Stores system-wide announcements. ○ announcement_id (PK) ○ title, content, posted_by_user_id (FK), target_role, image_file_path, expiry_date, created_at, updated_at ○ Relationships:
  - Many-to-One with Users table on posted_by_user_id: Multiple announcements can be posted by one user (Admin).

Project File Structure (com/grpAC_SMS as base package)

```
StudentManagementSystem_GrpAC/
├── pom.xml
├── .gitignore
├── README.md
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── grpAC_SMS/ (Base Package)
│   │   │           ├── controller/
│   │   │           │   ├── admin/
│   │   │           │   │   ├── AdminDashboardServlet.java
│   │   │           │   │   ├── ManageUsersServlet.java
│   │   │           │   │   ├── ManageDepartmentsServlet.java
│   │   │           │   │   ├── ManageProgramsServlet.java
│   │   │           │   │   ├── ManageCoursesServlet.java
│   │   │           │   │   ├── ManageFacultyServlet.java (For specific faculty
profile ops beyond user)
│   │   │           │   │   ├── ManageStudentsServlet.java (For specific student
profile ops beyond user)
│   │   │           │   │   ├── ManageAcademicTermsServlet.java
│   │   │           │   │   ├── ManageLocationsServlet.java
│   │   │           │   │   ├── ManageLectureSessionsServlet.java
│   │   │           │   │   ├── ManageEnrollmentsServlet.java
│   │   │           │   │   ├── AdminManageGradesServlet.java (For oversight)
│   │   │           │   │   ├── AdminManageAttendanceServlet.java (For oversight/manual
entry)
```

```
| | | | | ├── ManageAnnouncementsServlet.java
| | | | | └── SimulatedAttendancePunchServlet.java
| | | | ├── auth/
| | | | | ├── LoginServlet.java
| | | | | └── LogoutServlet.java
| | | | ├── faculty/
| | | | | ├── FacultyDashboardServlet.java
| | | | | ├── FacultyCourseManagementServlet.java (View students,
select session)
| | | | | ├── FacultyRecordGradesServlet.java
| | | | | └── FacultyRecordAttendanceServlet.java
| | | | └── student/
| | | | ├── StudentDashboardServlet.java
| | | | ├── StudentViewCoursesServlet.java
| | | | ├── StudentViewGradesServlet.java
| | | | └── StudentViewAttendanceServlet.java
| | | ├── model/
| | | | ├── User.java
| | | | ├── Department.java
| | | | ├── Program.java
| | | | ├── Student.java
| | | | ├── Faculty.java
| | | | ├── Course.java
| | | | ├── AcademicTerm.java
| | | | ├── Location.java
| | | | ├── LectureSession.java
| | | | ├── Enrollment.java
| | | | ├── Grade.java
| | | | ├── Attendance.java
| | | | ├── Announcement.java| | | | └── Role.java (Enum: ADMIN, STUDENT, FACULTY)
| | | ├── dao/
| | | | ├── UserDao.java
| | | | ├── DepartmentDao.java
| | | | ├── ProgramDao.java
| | | | ├── StudentDao.java
| | | | ├── FacultyDao.java
| | | | ├── CourseDao.java
| | | | ├── AcademicTermDao.java
| | | | ├── LocationDao.java
| | | | ├── LectureSessionDao.java
| | | | ├── EnrollmentDao.java
```

```
| | | | ├── GradeDao.java
| | | | ├── AttendanceDao.java
| | | | ├── AnnouncementDao.java
| | | | └── impl/
| | | | ├── UserDaoImpl.java
| | | | ├── DepartmentDaoImpl.java
| | | | ├── ProgramDaoImpl.java
| | | | ├── StudentDaoImpl.java
| | | | ├── FacultyDaoImpl.java
| | | | ├── CourseDaoImpl.java
| | | | ├── AcademicTermDaoImpl.java
| | | | ├── LocationDaoImpl.java
| | | | ├── LectureSessionDaoImpl.java
| | | | ├── EnrollmentDaoImpl.java
| | | | ├── GradeDaoImpl.java
| | | | ├── AttendanceDaoImpl.java
| | | | └── AnnouncementDaoImpl.java
| | | ├── service/ (Optional, but good for complex logic)
| | | | ├── UserService.java
| | | | ├── AttendanceService.java
| | | | └── impl/
| | | | ├── UserServiceImpl.java
| | | | └── AttendanceServiceImpl.java
| | | ├── util/
| | | | ├── DatabaseConnector.java
| | | | ├── PasswordHasher.java
| | | | ├── InputValidator.java
| | | | ├── ApplicationConstants.java
| | | | └── DateFormatter.java
| | | ├── filter/
| | | | ├── AuthenticationFilter.java
| | | | └── AuthorizationFilter.java
| | | ├── listener/
| | | | └── ApplicationContextListener.java
| | | └── exception/
| | | ├── DataAccessException.java
| | | └── BusinessLogicException.java
| | ├── resources/
| | | ├── db/
| | | | └── database.properties
| | | └── logback.xml (If SLF4J/Logback is used)
```

```
| | └── webapp/
| |  ├── WEB-INF/
| | |  ├── web.xml
| | |  └── lib/
| |  ├── assets/
| | |  ├── css/| | | | └── styles.css (Main custom stylesheet)
| | |  ├── js/
| | | | └── scripts.js (For general JS, AJAX for simulator)
| | |  └── img/
| | |  └── logo.png
| |  ├── common/ (Reusable JSP parts)
| | |  ├── header.jsp
| | |  ├── footer.jsp
| | |  ├── admin_nav.jsp
| | |  ├── faculty_nav.jsp
| | |  ├── student_nav.jsp
| | |  └── unauthorized.jsp
| |  ├── auth/
| | |  └── login.jsp
| |  ├── admin/
| | |  ├── dashboard.jsp
| | |  ├── users_list.jsp
| | |  ├── user_form.jsp (For add/edit user)
| | |  ├── departments_list.jsp
| | |  ├── department_form.jsp
| | |  ├── programs_list.jsp
| | |  ├── program_form.jsp
| | |  ├── courses_list.jsp
| | |  ├── course_form.jsp
| | |  ├── faculty_list.jsp (List of faculty profiles)
| | |  ├── faculty_form.jsp (For faculty-specific profile details)
| | |  ├── students_list.jsp (List of student profiles)
| | |  ├── student_form.jsp (For student-specific profile details)
| | |  ├── terms_list.jsp
| | |  ├── term_form.jsp
| | |  ├── locations_list.jsp
| | |  ├── location_form.jsp
| | |  ├── lecture_sessions_list.jsp
| | |  ├── lecture_session_form.jsp
| | |  ├── enrollments_list.jsp
| | |  ├── enrollment_form.jsp
```

```
| | |  ├── announcements_list.jsp
| | |  ├── announcement_form.jsp
| | |  └── nfc_attendance_simulator.jsp
| |  ├── faculty/
| | |  ├── dashboard.jsp
| | |  ├── my_courses.jsp (List of courses faculty teaches)
| | |  ├── course_details_faculty.jsp (Shows enrolled students, links to
grade/attendance)
| | |  ├── record_grades_form.jsp (For a specific course/assessment)
| | |  ├── select_lecture_session_for_attendance.jsp (Faculty selects session)
| | |  └── record_session_attendance_form.jsp (Faculty marks for selected
session)
| |  ├── student/
| | |  ├── dashboard.jsp (Personalized dashboard)
| | |  ├── my_profile.jsp
| | |  ├── my_enrolled_courses.jsp
| | |  ├── my_grades.jsp
| | |  └── my_attendance.jsp
| |  ├── errorpages/
| | |  ├── error_404.jsp
| | |  └── error_500.jsp
| |  └── index.jsp
|  └── test/|  ├── java/
| |  └── com/grpAC_SMS/
| |  ├── dao/impl/ (Tests for DAO implementations)
| |  ├── service/impl/ (Tests for Service implementations)
| |  └── util/ (Tests for Utility classes)
|  └── resources/
|  └── test-database.properties
```

4. Personal Reflection

contribution primarily focused on the [mention specific module or feature, e.g., implementation of the student academic records viewing functionalities]. This involved designing and implementing the necessary [e.g., DAOs like StudentDaoImpl.java] to interact with the database and retrieve student-specific data, developing the corresponding [e.g., Servlets like StudentViewGradesServlet.java] to handle user requests and data processing, and creating/modifying the [e.g., JSPs like my_grades.jsp] to present the information clearly to the student user. A key technical challenge in this area was [describe a specific challenge, e.g.,

accurately calculating the overall GPA based on various grading schemes, or ensuring efficient data retrieval for large student records], which I addressed by [explain how you solved it, e.g., implementing a specific calculation logic in a service class or optimizing database queries]. I also contributed to [mention other contributions, e.g., writing unit tests for certain DAO methods, or assisting with frontend styling]. Ensuring my code integrated seamlessly with the work of other team members required close coordination.

In terms of teamwork and process, our group adopted a collaborative approach using [e.g., GitHub] for version control, which facilitated parallel development but occasionally led to [e.g., merge conflicts] that required careful resolution. We conducted [e.g., weekly video calls] to synchronize progress, discuss roadblocks, and plan upcoming tasks. Task allocation was generally based on [e.g., individual strengths or preferences], and we tried to maintain open communication through [e.g., a group chat]. What worked well was our team's willingness to
[e.g., help each other when stuck on technical issues]. However, an area for improvement was [e.g., establishing a more formal code review process to catch potential bugs earlier], and occasionally [e.g., inconsistent availability for meetings] posed challenges to timely decisionmaking.

Reflecting on the technologies used, working with raw Servlets, JSP, and JDBC provided a foundational understanding of Java web application development. A significant challenge was the amount of boilerplate code required for tasks like [e.g., managing

database connections or handling form data]. Compared to using a framework like [e.g., Spring MVC] which I have prior experience with, the development felt more manual and required a deeper understanding of the underlying protocols and APIs. However, this also offered greater insight into the request-response lifecycle and how components interact at a lower level. Learning to configure and deploy on Apache Tomcat was also a valuable experience. Git proved indispensable for managing our codebase and collaborating effectively, though mastering conflict resolution was a learning curve.

The lessons learned from this project are numerous. Technically, I now have a much stronger appreciation for [e.g., the importance of robust input validation on the server-side] to prevent security vulnerabilities, and the benefits of abstracting database interactions through the DAO pattern. From a teamwork perspective, I learned that [e.g., proactive communication about progress and impediments] is crucial for keeping the project on track. I also recognized the value of [e.g., breaking down large tasks into smaller, manageable units] and the importance of [e.g., clearly defining responsibilities] to avoid confusion. Moving forward, I would prioritize [e.g., implementing automated testing earlier in the development cycle] and dedicate more time to [e.g., refining the database design before coding begins] in future projects. This project reinforced that successful software development requires not just technical skill, but also effective collaboration and a willingness to continuously learn and adapt.

5. Future Work

This section outlines potential avenues for enhancing and extending the Student Management System, focusing on new features, scalability, sustainability, and overall report quality.

Improvements and Extensions:

The SMS can be significantly enhanced by incorporating additional modules and features. One key extension is the development of a Library Management Module, which

could include functionalities for cataloging books, managing student loans, tracking due dates, and sending automated reminders. A Finance Module could be integrated to handle fee payments, generate invoices, track outstanding balances, and provide financial reports for students and administration.

To improve accessibility and user experience, developing APIs for mobile app integration would be beneficial. This would allow students and faculty to access core system functionalities, such as viewing grades, attendance, and announcements, through dedicated mobile applications on iOS and Android platforms.

Furthermore, the system could be enhanced with advanced reporting and analytics capabilities. This would involve implementing features to generate custom reports on student performance, course enrollment trends, and other relevant metrics, providing valuable insights for academic planning and decision-making. This could involve integrating a charting library.

Finally, improved security measures are always a priority. Future work could include implementing more robust authentication mechanisms, such as multi-factor authentication, and conducting regular security audits to identify and address potential vulnerabilities. Deployment to a cloud platform (like AWS, Azure, or Google Cloud) could improve accessibility and scalability.

Scalability and Sustainability:

To ensure the long-term viability and performance of the SMS, several changes can be considered. Transitioning from raw JDBC to an Object-Relational Mapping (ORM) framework like JPA/Hibernate would simplify database interactions, improve code maintainability, and potentially enhance performance through features like caching. Optimizing database queries is crucial for handling large volumes of data efficiently. This could involve using database indexing, query profiling tools, and implementing query optimization techniques. Implementing a caching mechanism (e.g., using Redis or Memcached) can reduce database load and improve response times for frequently accessed data.

For sustainability, improving test coverage is essential. This involves writing comprehensive unit and integration tests for all modules and functionalities to ensure code reliability and prevent regressions. Implementing a Continuous Integration/Continuous Deployment (CI/CD) pipeline would automate the build, test, and deployment processes, enabling faster and more reliable software releases. This would involve tools like Jenkins, GitLab CI, or GitHub Actions.

Diagrams:

Key diagrams, such as the [e.g., updated ER diagram, or a deployment diagram showing the cloud architecture], should be embedded directly within the report. Ensure these diagrams are appropriately sized and clearly readable. Use a consistent and standard notation (e.g., UML) for all diagrams. Larger or supplementary diagrams, like detailed sequence diagrams or class diagrams, can be included in the appendices and referenced within the main body of the report.

Version Control Evidence:

The project's development process was managed using Git. The project repository is located at [https://github.com/Killbane12/StudentManagementSystem_GrpAC]. Our branching strategy involved using [e.g., feature branches for individual modules and a main branch for integration]. The commit logs provide a detailed history of the project's evolution and demonstrate individual contributions. For example, the commit log entries for [e.g., the user management module] clearly show the implementation of [e.g., user authentication and authorization functionalities]. A snapshot of representative commit logs can be found in [e.g., Appendix A].

Referencing:

This report adheres to the [e.g., APA 7th Edition] referencing style. All external libraries, frameworks, architectural patterns, and significant algorithms or concepts used in the project are cited both in-text and in the reference list. For instance, the [e.g., JDBC API] was used for database connectivity, and its usage is referenced as [e.g., (Oracle, 2023)]. The tools used for creating diagrams, such as [e.g., draw.io], are also cited.