Assignment 2

WIF3006 COMPONENT-BASED SOFTWARE ENGINEERING

GROUP 2

Lecturer: ASSOCIATE PROF. TS. DR. MUMTAZ BEGUM BINTI PEER MUSTAFA

Team Members:

No	Name	Matric Number
1	Muhammad Haziq Halifi bin Ab Rahman	U2101028
2	Muhammad Irham Syahmi bin Arawi	U2102067
3	Ariff Irfan Bin Che Hamzah	22000822
4	Amir Mustaqim bin Samsudin	22001730
5	Mao Shengwu	22117349
6	Ahmad Danial Adli Bin Mohd Zaki	22000628

Table of Contents

1.0 Introduction

The digital transformation of higher education has necessitated the development of comprehensive University Management Systems to streamline academic and administrative operations. A robust UMS is responsible for managing a vast array of interconnected functions, including student record keeping, course enrollment, financial billing, grading, and timetable scheduling. However, as these systems evolve to accommodate growing student populations and complex academic requirements, the underlying architectural framework becomes a critical factor in determining the system's long-term viability and performance.

The current University Management System analyzed in this study is built upon a monolithic architecture. In this design, all functional modules reside within a single, unified codebase. While this approach may initially simplify development, it introduces significant structural challenges that hinder the institution's ability to innovate and maintain the system effectively.

Recognizing these limitations, the primary goal of this project is to transition the University Management System from its current monolithic state toward a more modular and resilient architecture. The project is guided by three core objectives:

1. Identify and design the use case and the class diagrams for the monolithic University Management System
2. Evaluate the dependency depth of the system’s design to pinpoint high-risk areas of tight coupling and identify specific modules that require decoupling to improve system stability.
3. Implement Component-Based Software Engineering (CBSE) concepts to redesign the application. By converting the monolithic structure into independent, reusable components, the system will achieve better fault isolation, independent scalability, and the flexibility to incorporate diverse technological stacks for specialized tasks.

By moving toward a component-based design, the University Management System will be better equipped to serve the evolving needs of students, faculty, and administrators while ensuring high availability and ease of continuous deployment.

2.0 Monolithic System Identification and Problem Analysis

2.1 Selected Monolithic System

- **System Name:** University Management System

2.2 Functional Scope of the Monolithic System

Module Name	Functionalities
Module 1: Student Management Module [Syahmi]	1. Manage student records 2. Manage student enrollment into semesters 3. Generate academic profiles 4. Track disciplinary records
Module 2: Course & Curriculum Management Module [Haziq]	1. Manage Course Catalog 2. Validate Prerequisite Course 3. Assign course to Lecturers 4. Retrieve Course Schedule/Rooms
Module 3: Lecturer/Staff Management Module [Adli]	1. Manage lecturer information 2. Teaching load assignment 3. Performance evaluation tracking 4. Staff timetable generation
Module 4: Examination & Grading Module [Ariff]	1. Manage exam schedule 2. Grade submission and approval 3. GPA/CGPA calculation 4. Generate transcripts
Module 5: Timetable & Scheduling Module [Mao]	1. Manage class timetable 2. Conflict detection (room/lecturer clashes) 3. Auto-generate proposed timetable 4. Publish updated timetable to students/lecturers
Module 6: Fee & Billing Management Module [Amir]	1. Manage student bills 2. Track payments and outstanding fees 3. Generate invoices 4. Apply scholarships/discounts

2.3 Problems with the Existing Traditional System

The existing traditional system that our group has chosen - University Management System, presents some usual problems when dealing with monolithic applications.

1. Tight Coupling (Inter-module Dependencies)

In a monolithic architecture, different functional areas often share the same database tables and internal classes, creating a complex code structure where business logic is entangled. When a module tries to access data owned by another, it often bypasses the API and reads the database directly. Thus, modifying the code or database schema in one module can eventually cause errors in a completely different part of the system. For example, if the developer modifies the Student database table structure under Manage Student Records functionalities in Module 1, it will immediately break the Generate Invoice feature in Module 6 as the billing code is hard-wired to the old Student table definition.

2. Inefficient Scalability

Monolithic applications must be scaled as a single unit. This leads to inefficient resource utilisation because scaling the application larger means replicating the whole system including idle modules just to handle traffic for a single busy function. For example, Module 2 experiences massive traffic during the student registration week as thousands of students attempt to Assign Course To Section simultaneously. The IT team then adds more servers to run the entire application including the Module 4 which is completely idle, causing the waste of memory and CPU power.

3. Low Reliability (Lack of Fault Isolation)

In usual monolithic applications, all modules run within the same process or application server. This leads to a fatal risk where one error in a module can bring down the entire system. If one module consumes all available memory or triggers an unhandled exception, the operating system kills the entire process, causing the breakdown of the entire system for all users. For example, when there is a memory leak due to coding error in Auto-generate Proposed Timetable feature in Module 5, the entire server will crash. This crash would also prevent the other users from accessing the other functionalities in all modules of the system.

4. Difficulty in Continuous Deployment

A monolithic application requires re-compiling, re-testing, and re-deploying the entire system even when the changes made are minor. This results in long development cycles where updates are delayed because the risk of breaking something unrelated is too high. For example, fixing a bug in the Manage Lecturer Information feature in Module 3 would require re-build and re-deploy the entire 6-modules system. This introduces developers to a risk that the new fix

might eventually create errors in the Grade Submission and Approval feature in Module 4. Because of the risk that persist in the small fix plan, developers might delay the maintenance task until the next major scheduled update window.

5. Technology Lock-in

A single technology stack is typically enforced to a monolithic application. This prevents developers from using the right tools for specific problems, and instead they are forced to compromise on performance of the system because they cannot easily mix different programming languages or frameworks within the same application. For example, complex constraint problems in the Conflict Detection feature in Module 5 are best solved using Python or Prolog that have strong AI libraries. However, Module 6 requires strict transactional integrity best handled by Java (initial language used by the application). This monolithic system will force developers to use Java to write the Conflict Detection feature in Module 5 which might be less efficient for AI tasks.

3.0 Monolithic System Design using UML

3.1 Use Case Diagram & Description

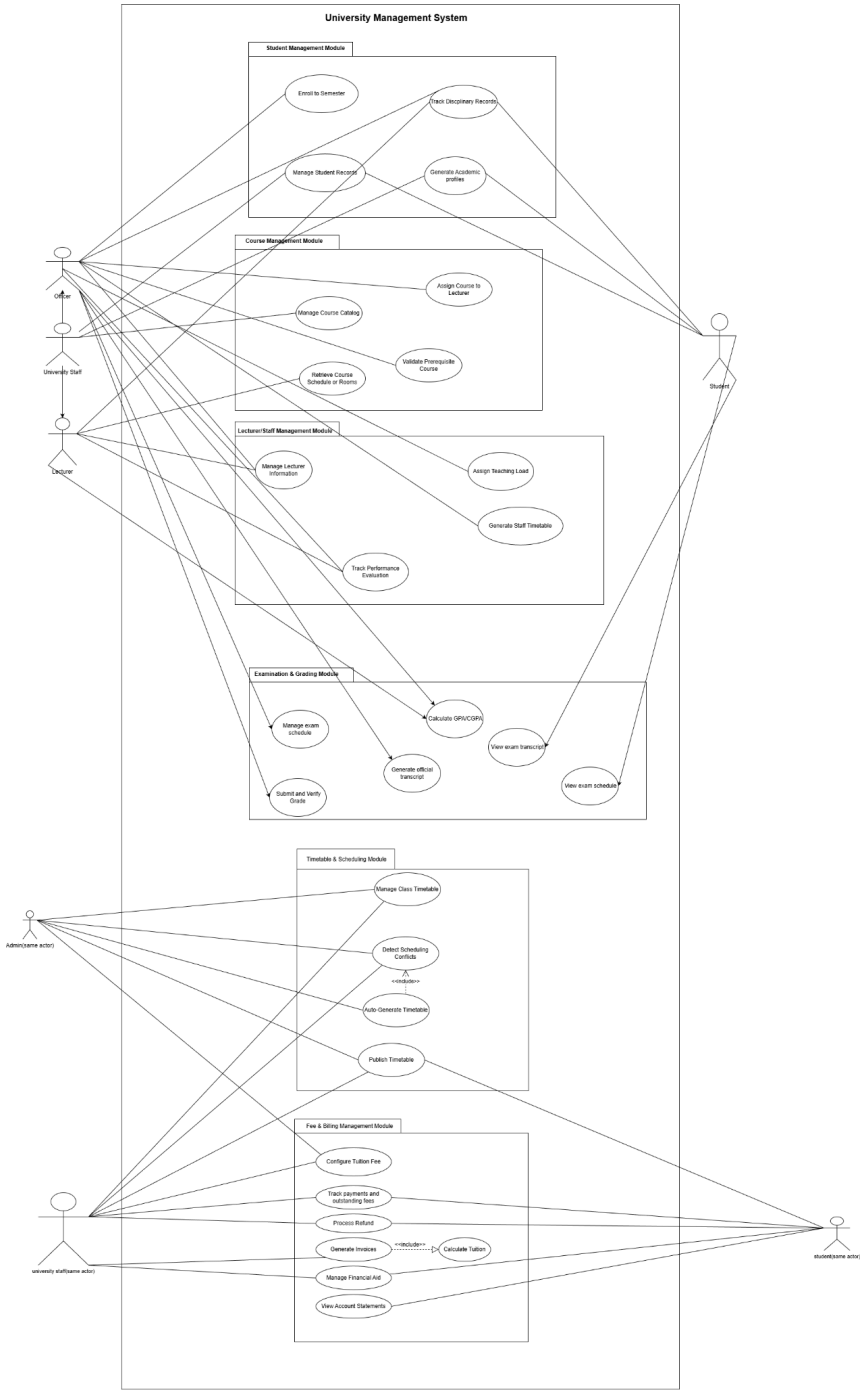

Figure 3.1: Use Case Diagram

3.1.1 Module 1: Student Management Module

3.1.1.1 UC-01: Manage Student Records

Element	Detail
Use Case Name	Manage Student Records
Use Case ID	UC-01
Priority	HIGH
Actor(s)	Officer , Lecturer , Student
Description	Allow officer and authorized users to create , read , update and delete student records including information , contact details and documents
Pre-condition(s)	User is logged in with appropriate privileges
Post-condition(s)	Student records are created/updated/deleted in the database
Flow of events	1. User logs into the system 2. User selects Manage Student Records option 3. System displays student management interface 4. User selects desired operation (Add/View/Update/Delete) 5. For Add: User enters new student information and uploads documents 6. For View: User searches and views student information based on access level 7. For Update: User modifies existing student information 8. For Delete: User removes student record with confirmation 9. System validates all input data 10. System performs the requested operation 11. System logs the transaction with timestamp and user details 12. System displays confirmation message 13. System sends notifications to relevant parties if applicable

Alternative Flows and Exceptions	Exception Flow : E1 : Validation fails E1.1 System display specific error message and prompt user to correct E2 : Student already exist E2.1 System alerts user E3 : Student not found E3.1 System displays no record founds

Table 3.1.1.1: UC-01 Manage Student Records

3.1.1.2. UC-02: Enroll into Semester

Element	Detail
Use Case Name	Enroll to semesters
Use Case ID	UC-02
Priority	HIGH
Actor(s)	Officer , student
Description	Enroll student into specific semesters with course selection , prerequisite validation and withdraw the semester
Pre-condition(s)	Student record exist and active , semester is available for enrollment
Post-condition(s)	Student is enrolled in or withdrawn from semester
Flow of events	1. User selects Student Enrollment option 2. System displays available semesters 3. User selects student (if staff) or system loads current student 4. User selects target semester 5. For Enrollment: System displays eligible courses with prerequisites 6. User selects courses to enroll 7. System validates prerequisites for each selected course 8. System checks credit hour limits (minimum and maximum) 9. System calculates total fees based on selected courses

	10. System displays enrollment summary with course list and total fees 11. User confirms enrollment 12. System processes enrollment and generates course schedule 13. System creates fee invoice 14. For Withdrawal: System checks drop eligibility period 15. System updates enrollment status 16. System sends confirmation to student and relevant staff
Alternative Flows and Exceptions	Exception Flow : E1 : No eligible course available E1.1 System displays reasons(message) and end process E2 : Prerequisite not met E2.1 System displays missing prerequisite and block enrollment E3 : Credit hours exceed/below limit E3.1 System display error and suggest adjustment

Table 3.1.1.2: UC-02 Student Enrollment into Semesters

3.1.1.3. UC-03: Generate Academic Profile

Element	Detail
Use Case Name	Generate academic profile
Use Case ID	UC-03
Priority	HIGH
Actor(s)	Officer , Lecturer , Student
Description	Generate comprehensive academic documents including transcript , performance reports
Pre-condition(s)	User is logged in to the system and student has academic records in the system
Post-condition(s)	Academic document are generated

Flow of events	1. User selects Generate Academic Profile option 2. System displays profile generation menu 3. User selects document type (Transcript/Performance Report) 4. User selects student (if staff) or system loads current student 5. For Transcript: User selects type (official/unofficial) 6. For Performance Report: User selects reporting period and components 7. System retrieves all academic records from database 8. System calculates CGPA and semester GPAs 9. System compiles comprehensive document with all required information 10. System applies digital signature for official transcripts 11. System displays document preview 12. User confirms and downloads the document 13. System logs the generation with user details and timestamp
Alternative Flows and Exceptions	Exception Flow : E1 : No grade records found E1.1 System displays no academic records available

Table 3.1.1.3: UC-03 Generate Academic Profile

3.1.1.4. UC-4: Track Disciplinary Records

Element	Detail
Use Case Name	Track Disciplinary Records
Use Case ID	UC-04
Priority	HIGH
Actor(s)	Officer , Student
Description	Manage disciplinary cases including recording new incidents , update case status , track case history , generate report and maintaining disciplinary records with documentations
Pre-condition(s)	Student records exist in the system for case recording

Post-condition(s)	Disciplinary case are recorded.updated/viewed in the system
Flow of events	1. User logs into the system 2. User selects Disciplinary Records Management option 3. System displays disciplinary management interface 4. User selects operation (Record Case/Update Status/View History/Generate Report) 5. For Record Case: Officer enters student identification and incident details 6. System retrieves and displays student information 7. Officer enters date, time, location, violation type, and description 8. Officer uploads supporting evidence (photos, documents, witness statements) 9. Officer assigns case severity level and investigating officer 10. System generates unique case reference number 11. For Update Status: Officer searches and selects case 12. System displays current case details and history timeline 13. Officer updates status (Under Investigation/Hearing Scheduled/Resolved/Dismissed) 14. Officer enters update notes and specifies next actions 15. System validates status transitions 16. For View History: User enters student identification 17. System verifies user permissions 18. System retrieves and displays all cases chronologically 19. For Generate Report: User selects report type and criteria 20. System performs analysis and generates report with statistics 21. System saves all changes and updates case timeline
Alternative Flows and Exceptions	Exception Flow : E1 : Student not found E1.1 System displays error and allow retry E2 : Violation type not in list E2.1 System prompt officer to selects Others and provide description E3 : No disciplinary records exist

	E3.1 System display no records found

Table 3.1.1.4: UC-04 Track Disciplinary Records

3.1.2 Module 2: Course & Curriculum Management Module

3.1.2.1 UC-05: Manage Course Catalog

Element	Detail
Use Case Name	Manage Course Catalog
Use Case ID	UC-05
Priority	High
Actor(s)	Officer
Description	Allows the Officer to create, update, or retire course records, including Course Code, Course Name, Credit Value, and Course Description.
Pre-condition(s)	The Officer is authenticated and has catalog management privileges.
Post-condition(s)	Course records are validated and stored in the course repository.
Flow of events	1. Officer selects Manage Course Catalog from the system menu. 2. System displays the list of existing courses. 3. Officer selects an action: Create, Update, or Retire a course. 4. Officer enters or modifies course details. 5. Officer confirms the changes. 6. System validates the input and updates the course repository. 7. System displays a success confirmation.
Alternative Flows and Exceptions	**Alternative Flows:** **A1: Search and Filter Courses** A1.1 Officer enters search criteria or applies filters. A1.2 System displays the filtered course list. **Exceptions:** **E1: Delete Course Linked to Active Sections**

	E1.1 Officer attempts to retire or delete a course that is linked to one or more active sections. E1.2 System rejects the request. E1.3 System displays the message:

Table 3.1.2.1: UC-05 Manage Course Catalog

3.1.2.2 UC-06: Validate Prerequisite Course

Element	Detail
Use Case Name	Validate Prerequisite Course
Use Case ID	UC-06
Priority	High
Actor(s)	Officer
Description	Validates whether a student has satisfied the prerequisite requirements for a selected course based on the student's academic records.
Pre-condition(s)	The Officer is authenticated. The target course exists. The student has academic records stored in the system.
Post-condition(s)	The system returns a prerequisite validation result (MET or NOT MET) to the requesting module.
Flow of events	1. Officer selects a student and a target course for prerequisite validation. 2. System retrieves the prerequisite rules for the selected course. 3. System requests the student's completed course and grade records from the Examination Module. 4. System evaluates whether all prerequisite requirements are satisfied. 5. System returns the validation result (MET or NOT MET) to the calling module.
Alternative Flows and Exceptions	**Alternative Flows:** **A1: Course Has No Prerequisites** A1.1 System detects that the selected course has no prerequisite requirements. A1.2 System returns the validation result as **MET**. **Exceptions:**

	E1: Unable to Retrieve Student Grade Records E1.1 System fails to retrieve grade data from the Examination Module. E1.2 System logs the error for audit purposes. E1.3 System returns the validation result as **NOT MET**.

Table 3.1.2.2: UC-06 Validate Prerequisite Course

3.1.2.3 UC-07: Assign Course to Lecturer

Element	Detail
Use Case Name	Assign Course to Lecturers
Use Case ID	UC-07
Priority	High
Actor(s)	Officer
Description	Allows the Officer to assign one or more lecturers to a course based on expertise and teaching load constraints.
Pre-condition(s)	The Officer is authenticated. The course exists. At least one lecturer exists in the system.
Post-condition(s)	Lecturer–course assignments are recorded, and each assigned lecturer's teaching load is updated accordingly.
Flow of events	1. Officer selects a course to be assigned. 2. System displays eligible lecturers along with their expertise and current teaching load. 3. Officer selects one or more lecturers for the course. 4. System validates the assignment against teaching load limits. 5. System displays the updated load status. 6. Officer confirms the assignment. 7. System saves the assignment and displays a success message.
Alternative Flows and Exceptions	**Alternative Flows:** **A1: Assign Course as "To Be Determined (TBD)"** A1.1 Officer selects the **TBD** option instead of choosing lecturers. A1.2 System records the course as unassigned. A1.3 System marks the course assignment status as Pending.

	Exceptions: **E1: Lecturer Teaching Load Exceeds Maximum Capacity** E1.1 System detects that one or more selected lecturers exceed the maximum teaching load. E1.2 System displays a warning message showing the overload details. E1.3 Officer chooses to either cancel the assignment or proceed with an override. E1.4 If an override is approved, system records the assignment with an override flag.

Table 3.1.2.3: UC-07 Assign Course to Section and Lecturer

3.1.2.4 UC-08: Retrieve Course Schedule/Rooms

Element	Description
Use Case Name	Retrieve Course Timetable Slot
Use Case ID	UC-08
Priority	Medium
Actor(s)	Officer, Student
Description	Allows an Officer or Student to retrieve the scheduled day, time, and room for a specific course section.
Pre-condition(s)	The course section exists. The timetable has been published or marked as tentative by the Timetable Module.
Post-condition(s)	The requested timetable information is displayed to the actor or returned to the requesting module.
Flow of events	1. Actor selects a course section to view its timetable details. 2. System requests the schedule information for the selected section from the Timetable Module. 3. Timetable Module returns the schedule data. 4. System formats the schedule information. 5. System displays the finalized timetable details (day, time, and room) to the actor.

Alternative Flows and Exceptions	**Alternative Flows:** **A1: Final Schedule Not Yet Published** A1.1 System detects that the finalized timetable is unavailable. A1.2 System retrieves the tentative schedule instead. A1.3 System displays the timetable with a **"Tentative"** status indicator. **Exceptions:** **E1: Schedule Conflict Detected** E1.1 Timetable Module reports a schedule conflict for the requested section. E1.2 System logs the conflict details. E1.3 System displays a conflict warning message to the actor.

Table 3.1.2.4: UC08 Retrieve Course Timetable Slot

3.1.3 Module 3: Lecturer/Staff Management Module

3.1.3.1 UC-09: Manage Lecturer information

Element	Detail
Use case name	Manage Lecturer Information
Use case ID	UC-09
Actor(s)	Officer, Lecturer
Brief description	Officer and Lecturer able to Create, Read, Update, and Delete (CRUD) lecturer profiles, including personal details, expertise, and contact info.
Pre-conditions	Officer is logged into the system. The lecturer has been officially hired and their personal/employment details are available.
Post-conditions	Lecturer records are successfully added, updated, or removed from the database.
Main flow	1. Officer selects "Lecturer Management" from the menu. 2. System displays a list of existing lecturers. 3. Officer chooses to "Add New", "Edit", or "Delete". 4. **If Add:** Officer enters name, ID, department, and qualifications. 5. **If Edit:** Officer modifies existing details. **6. If Delete:** The lecturer information will be deleted from database.

	7. System validates the input. 8. System saves the new record to the database. 9. System confirms success.
Alternative Flow	If mandatory fields are missing, the System displays an error and prompts the Admin to retry.
Exception flow	System detects a duplicate Staff ID. System displays error "ID already exists" and prompts Admin to re-enter a unique ID.
Priority	High
Non-functional requirements	The system must validate input data (e.g., email format) in real-time.
Assumptions	Officer has the necessary permissions to write to the staff database.

Table 3.1.3.1: UC-09 Manage Lecturer Information

3.1.3.2 UC-10: Assign Teaching Load

Element	Detail
Use case name	Assign Teaching Load
Use case ID	UC-10
Actor(s)	Officer
Brief description	The Officer assigns specific courses (subjects) and credit hours to a lecturer for the upcoming semester.
Pre-conditions	The list of active courses for the semester is finalized and the lecturer exists in the system.
Post-conditions	The lecturer is linked to specific course codes and their total credit hours are updated.

Main flow	1. Officer selects the target Lecturer from the department list. 2. System displays the lecturer's current teaching load and remaining capacity. 3. Officer clicks "Assign Course". 4. System shows a dropdown of unassigned courses. 5. Officer selects a course (e.g., WIF3006). 6. System calculates the new total credit hours. 7. Officer confirms the assignment.
Alternative Flow	Officer selects an already assigned course and clicks "Unassign". System removes the link and recalculates credit hours.
Exception flow	The assignment causes the lecturer to exceed the maximum allowed credit hours (e.g., >18 hours). System alerts "Limit Exceeded" but allows Officer to proceed by entering a justification note.
Priority	High
Non-functional requirements	The calculation of total credit hours must happen in real-time (without page reload)
Assumptions	One course can be taught by multiple lecturers (split load), but this use case assumes a single lecturer assignment for simplicity.

Table 3.1.3.2: UC-10 Assign Teaching Load

3.1.3.3 UC-11: Track Performance Evaluation

Element	Detail
Use case name	Track Performance Evaluation
Use case ID	UC-11
Actor(s)	Officer
Brief description	Records and calculates the Key Performance Indicators (KPI) scores for a lecturer based on teaching, research, and service.
Pre-conditions	The academic semester or year has concluded, and the evaluation window is open.
Post-conditions	A finalized performance score is stored for the lecturer for that specific year.

Main flow	1. Officer navigates to the "Performance Review" module. 2. Officer retrieves the lecturer's profile. 3. System displays the evaluation form with categories (Teaching, Research, Service). 4. Officer inputs numerical scores (0-100) for each category. 5. System automatically calculates the weighted average/final score. 6. Officer adds qualitative comments/feedback. 7. Officer submits the evaluation.
Alternative Flow	Officer enters partial scores and clicks "Save Draft". System saves data without finalizing. Officer can resume later.
Exception flow	Officer tries to submit without filling all mandatory categories. System highlights missing fields in red and prevents submission.
Priority	Medium
Non-functional requirements	Access to performance data must be strictly restricted to Officer and Admin (Confidentiality)
Assumptions	The weightage logic (e.g., Research = 40%, Teaching = 40%) is pre-configured in the system settings.

Table 3.1.3.3: UC-11 Track Performance Evaluation

3.1.3.4 UC-12: Generate Staff Timetable

Element	Detail
Use case name	Generate Staff Timetable
Use case ID	UC-12
Actor(s)	Officer
Brief description	Generates the weekly class schedule for a staff member by mapping their assigned courses to available room slots
Pre-conditions	Teaching loads (Use Case 2) have been assigned and Room availability data is ready.
Post-conditions	A conflict-free timetable is published and available for the lecturer to view

Main flow	1. Officer selects "Generate Timetable" for a specific department or lecturer. 2. System retrieves all courses assigned to the lecturer. 3. System checks the Master Schedule for available time slots and rooms. 4. System allocates time slots for each course (e.g., Monday 10-12 PM). 5. System displays a draft timetable grid. 6. Officer reviews and clicks "Publish"
Alternative Flow	Officer sees a generated slot is not ideal, clicks on the slot, and manually moves it to a different available time
Exception flow	System detects that the lecturer is double-booked or the room is occupied. System flags the conflict in red and prompts the Officer to resolve it manually (as per "Resolve Timetable Conflict" logic).
Priority	High
Non-functional requirements	The generated timetable should be exportable to PDF format
Assumptions	Classes are scheduled in standard 1-hour or 2-hour blocks.

Table 3.1.3.4: UC-12 Generate Staff Timetable

3.1.4 Module 4: Grading & Examination

3.1.4.1 UC-13: Manage Exam Schedules

Element	Detail
Use Case Name	Manage Exam Schedules
Use Case ID	UC-13
Priority	High
Actor(s)	Officer
Description	The process where the Admin allocates dates, times, and venues for course exams while ensuring no clashes occur with other bookings.
Pre-condition(s)	1. Course list is finalized (from Course Module). 2. Venues are registered (from Timetable Module).

Post-condition(s)	1. Exam date, time, and venue are successfully published to the portal.
Flows of events	1. Officer selects a specific Semester and Course (e.g., WIF3006). 2. Officer inputs the proposed Date, Start Time, and Duration. 3. Officer selects a Venue type (e.g., Main Hall). 4. System validates Venue Availability (checks against Timetable Module). 5. System checks for Student Clashes (ensures enrolled students hav no overlapping exams). 6. System saves the schedule. 7. System publishes the slot to the Student and Lecturer portals.
Alternative Flows and Exceptions	**Alternative Flows:** **A1: Edit Existing Schedule** A1.1 Officer selects an already scheduled exam. A1.2 Officer modifies the time/venue. A1.3 System re-validates and updates the record. **Exceptions:** **E1: Venue Conflict (Clash)** E1.1 System detects the chosen Venue is occupied. E1.2 System displays error: "Venue unavailable at this time." E1.3 Use case resumes at step 2 (Admin enters new time).

Table 3.1.4.1: UC-13 Manage Exam Schedule

3.1.4.2 UC-14: Submit & Verify Grades

Element	Detail
Use Case Name	Submit & Verify Grades
Use Case ID	UC-14
Priority	High
Actor(s)	Lecturer, Officer
Description	The workflow where Lecturers enter student marks and Admin review them for completeness before finalizing.
Pre-condition(s)	Exam period has ended and the system is open for grading.

Post-condition(s)	Grades are permanently locked and ready for GPA/CGPA calculation
Flows of events	1. Lecturer logs in and selects a Course. 2. Lecturer inputs marks for Continuous Assessment and Final Exam. 3. System calculates the total raw score. 4. Lecturer clicks "Submit Final Grades". 5. System changes status to "Pending Verification". 6. Officer receives a notification of submission. 7. Officer opens the submission to check for completion (ensure no missing marks). 8. Officer clicks "Approve & Lock". 9. System updates status to "Finalized".
Alternative Flows and Exceptions	**Alternative Flows:** **A1: Save as Draft** A1.1 Lecturer inputs partial marks. A1.2 Lecturer clicks "Save Draft". A1.3 System saves data without changing status to "Pending". A1.4 Use case ends (Lecturer can return later). **Exceptions:** **E1: Invalid Mark Format** E1.1 Lecturer enters non-numeric data or value > 100. E1.2 System displays error: "Invalid Input." **E2: Incomplete Submission** E2.1 Officer finds missing marks during review. E2.2 Officer rejects submission. E2.3 System reverts status to "Draft".

Table 3.1.4.2: UC-14 Submit & Verify Grades

3.1.4.3 UC-15: Calculate GPA & CGPA

Element	Detail
Use Case Name	Calculate GPA & CGPA
Use Case ID	UC-15
Priority	High
Actor(s)	Admin
Description	The automated calculation logic that processes finalized marks to determine a student's semester GPA, cumulative CGPA, and academ standing.

Pre-condition(s)	All grades for the current semester are "Finalized".
Post-condition(s)	1. Student academic records are updated with new GPA/CGPA and Academic Status.
Flows of events	1. Trigger activates at the "Result Release" deadline. 2. System retrieves the credit hour value for every course taken (from Course Module). 3. System converts raw marks to Grade Points (e.g., 80 → 4.0). 4. System calculates GPA: (Sum of Grade Points * Credit Hours) / Tot Credit Hours. 5. System calculates CGPA: Aggregates current data with previous semesters' history. 6. System determines Academic Status (e.g., Pass, Dean's List, Probation, Fail). 7. System updates the student's Master Record. 8. System generates a notification for the student.
Alternative Flows and Exceptions	**Alternative Flows:** **-** **Exceptions:** **E1: Data Dependency Failure** E1.1 System cannot retrieve Credit Hours (Course Module invalid). E1.2 System halts calculation. E1.3 System logs critical error: "Missing Course Data." **E2: Pending Grades Detected** E2.1 System finds a course not yet "Finalized". E2.2 System skips calculation for affected students. E2.3 System generates "Exception Report" for Admin.

Table 3.1.4.3: UC-15 Calculate GPA & CGPA

3.1.4.4 UC-16: Generate Official Transcript

Element	Detail
Use Case Name	Generate Official Transcript
Use Case ID	UC-16
Priority	Medium

Actor(s)	Officer, Student
Description	Generates a formal, printable PDF document containing a student's complete academic history, contingent on financial clearance.
Pre-condition(s)	Student has no outstanding fees (Dependency on Fee Module).
Post-condition(s)	A PDF transcript is generated and downloaded.
Flows of events	1. User requests an Official Transcript. 2. System checks financial standing (queries Fee & Billing Module). 3. System retrieves full academic history (Sem 1 to current). 4. System calculates final graduating CGPA. 5. System formats the data into the University's official template (with watermark/seal). 6. System generates a downloadable PDF. 7. User downloads the file.
Alternative Flows and Exceptions	**Alternative Flows:** **A1: Preview Transcript** A1.1 User selects "View Web Version" instead of "Download PDF". A1.2 System renders HTML preview on screen. **Exceptions:** **E1: Financial Block (Outstanding Fees)** E1.1 System detects unpaid balance. E1.2 System stops generation. E1.3 System displays: "Transcript blocked due to outstanding fees."

Table 3.1.4.4: UC-16 Generate Official Transcript

3.1.4.5 UC-17: View Exam Transcript

Element	Detail
Use Case Name	View Exam Transcript
Use Case ID	UC-17
Priority	High
Actor(s)	Student
Description	Allows students to securely access and view their finalized grades, GPA, and academic status for the specific semester.
Pre-condition(s)	Results have been released by the Officer.
Post-condition(s)	Student views their grades and status.
Flows of events	1. Student logs into the portal. 2. Student navigates to "Exam Results". 3. System displays the current semester's subject list. 4. System retrieves grades, GPA, and CGPA from the database. 5. Student views the on-screen "Result Slip".
Alternative Flows and Exceptions	**Alternative Flows:** **A1: View Past Semesters** A1.1 Student selects a previous semester from dropdown. A1.2 System reloads page with historical data. **Exceptions:** **E1: Financial Hold** E1.1 System detects unpaid fees. E1.2 System hides results. E1.3 System displays: "Please contact Bursary." **E2: Early Access Attempt** E2.1 Student tries to view results before release date. E2.2 System displays: "Results available on [Date]."

Table 3.1.4.5: UC-17 View Exam Transcript

3.1.4.6 UC-18: View Exam Schedule

Element	Detail
Use Case Name	View Exam Schedule
Use Case ID	UC-18
Priority	Medium
Actor(s)	Student
Description	Provides students with a personalized view of their upcoming examination dates, times, and venue locations.
Pre-condition(s)	1. Exam Schedule has been published by the Admin. 2. Student is enrolled in active courses.
Post-condition(s)	Student views their personalized exam timetable.
Flows of events	1. Student logs into the portal. 2. Student clicks on "My Exam Schedule". 3. System retrieves enrolled courses (Dependency on Student Modul 4. System filters the Master Exam Schedule for these courses. 5. System checks for any venue updates. 6. System displays the Date, Time, Venue, and Seat Number for each exam.
Alternative Flows and Exceptions	**Alternative Flows:** **A1: Print Schedule** A1.1 Student clicks "Print Version". A1.2 System generates printer-friendly page. **Exceptions:** **E1: Schedule Not Published** E1.1 Schedule data is not yet active. E1.2 System displays: "Schedule pending release."

Table 3.1.4.6: UC-18 View Exam Schedule

3.1.5 Module 5: Timetable & Scheduling Module

3.1.5.1 UC-19: Manage Class Timetable

Element	Detail

Use Case Name	Manage Class Timetable
Use Case ID	UC-19
Priority	High
Actor(s)	Officer
Description	This use case allows the officer to create, view, and update the class timetable for a given semester, including assigning time slots, rooms, and linking lecturers to each class session.
Pre-condition(s)	1. Semester has been created in the system. 2. Course data (course list, sections, required contact hours) is available from the Course & Curriculum Management Module. 3. Room resources (rooms, capacities, basic availability) are registered in the system.
Post-condition(s)	1. A valid timetable record exists for each scheduled class session. 2. Any changes made to the timetable are stored in the system as the latest version.
Flow of events	1. The officer opens the timetable for a selected semester. 2. The system displays the existing timetable and available courses/sections. 3. The officer selects a course or section to schedule or edit. 4. The system shows available rooms and time slots. 5. The officer chooses a time slot, room and (optionally) a lecturer. The system checks room and lecturer availability. 6. The system saves the updated class session. 7. The system displays the updated timetable.
Alternative Flows and Exceptions	**Alternative Flows**: A.1. Room not available At Step 6, if the selected room is occupied, the system prompts the officer to choose another room or time slot. A.2. Lecturer not available At Step 6, if the lecturer is unavailable, the system prompts the officer to select a different lecturer or time slot.

	Exceptions: E.1. Missing course data If the system cannot retrieve required course data at Step 2: E.1.1. The system displays an error message. E.1.2. The system returns to the semester selection step.

Table 3.1.5.1: UC-19 Manage Class Timetable

3.1.5.2 UC-20: Detect Scheduling Conflicts

Element	Detail
Use Case Name	Detect Scheduling Conflicts
Use Case ID	UC-20
Priority	High
Actor(s)	Officer
Description	This use case checks the existing timetable for conflicts, such as overlapping bookings of the same room, overlapping classes for the same lecturer, or overlapping compulsory classes for the same program or cohort.
Pre-condition(s)	1. A draft timetable exists for the selected semester (at least partially scheduled). 2. Lecturer availability and teaching load information is available from the Lecturer/Staff Management Module. 3. Course and program structures (including compulsory courses per cohort) are available from the Course & Curriculum Management Module.
Post-condition(s)	1. A list of detected conflicts (if any) is generated and stored for further review. 2. Each conflict is classified by type (room conflict, lecturer conflict, cohort/program conflict).
Flow of events	1. The officer selects “Detect Scheduling Conflicts.” 2. The system loads all scheduled classes for the selected semester. 3. System checks for room conflicts. 4. System checks for lecturer conflicts. 5. System checks for cohort/program conflicts. 6. The system compiles and displays all detected conflicts or shows “No conflicts found.”

Alternative Flows and Exceptions	**Alternative Flows:** A.1. No conflicts found At Step 6, if no conflicts are detected, the system displays a “No conflicts found” message. A.2. Large timetable handling For large timetables, the system may present conflicts in multiple pages or filtered groups. **Exceptions:** E.1. Missing external data If lecturer or course data cannot be retrieved, the conflict-checking process cannot proceed. E.1.1. The system displays an error message. E.1.2. The system stops the process and returns to the data selection step.

Table 3.1.5.2: UC-20 Detect Scheduling Conflicts

3.1.5.3 UC-21: Auto-Generate Timetable

Element	Detail
Use Case Name	Auto-Generate Timetable
Use Case ID	UC-21
Priority	High
Actor(s)	Officer
Description	This use case allows the officer to automatically generate a proposed timetable for a semester based on course requirements, room capacities, and lecturer availability. The use case programmatically schedules classes and then invokes “Detect Scheduling Conflicts” to ensure basic consistency.
Pre-condition(s)	1. Semester has been defined and is active for scheduling. 2. Course offerings, required contact hours and preferred patterns (e.g., 2×2-hour sessions) are available from the Course & Curriculum Management Module. 3. Lecturer availability and maximum teaching load are available from the Lecturer/Staff Management Module. 4. Room resources (capacities, types) exist in the system.

Post-condition(s)	1. A proposed timetable is generated and stored as a draft for the selected semester. 2. Conflicts in the generated timetable (if any) are identified through the “Detect Scheduling Conflicts” use case. 3. The officer can further manually refine the timetable using “Manage Class Timetable”.
Flow of events	1. The officer selects “Auto-Generate Timetable.” 2. The system loads course offerings and contact hour requirements. 3. The system loads lecturer availability and teaching load data. 4. The system loads room information. 5. The system runs the auto-scheduling algorithm. 6. The system checks the generated timetable for conflicts. 7. The system saves the generated timetable as a draft. 8. The system displays the draft timetable and any conflicts or unassigned courses.
Alternative Flows and Exceptions	**Alternative Flows:** A.1. Unassigned courses At Step 5, if some courses cannot be scheduled, the system marks them as “Unassigned” and shows the reasons. A.2. Excessive conflicts After Step 7, if conflicts exceed the threshold, the system completes auto-generation but warns that manual adjustment is recommended. **Exceptions:** E.1. Algorithm failure or timeout If the auto-generation algorithm fails or times out: E.1.1. The system reports the error. E.1.2. No timetable changes are applied. E.2. Missing external data If required course or lecturer data cannot be retrieved:

	E.2.1. The system displays an error message. E.2.2. The system stops the auto-generation process.

Table 3.1.5.3: UC-21 Auto-Generate Timetable

3.1.5.4 UC-22: Publish Timetable

Element	Detail
Use Case Name	Publish Timetable
Use Case ID	UC-22
Priority	Medium–High
Actor(s)	Officer(primary), Lecturer (secondary)
Description	This use case allows the officer to publish the finalized timetable so that lecturers and students can view it through their respective interfaces or portals. It may also trigger notifications to lecturers about their teaching schedules.
Pre-condition(s)	1. A timetable version for the selected semester exists and has been reviewed (preferably with minimal or no conflicts). 2. The institution has decided that the timetable is ready to be communicated to lecturers and students.
Post-condition(s)	1. The selected timetable version is marked as “Published” in the system. 2. Lecturers can view their teaching timetable. 3. Students can view their class timetable (via student portal, if integrated). 4. Optional: notifications are sent to lecturers informing them of the final schedule.
Flow of events	1. The officer opens the timetable for a selected semester. 2. The system displays the timetable and indicates any remaining conflicts. 3. The officer selects “Publish Timetable.” 4. System marks the timetable version as Published. 5. The system updates the lecturer portal with teaching schedules. 6. The system updates the student portal with class schedules.

	7. System optionally sends notifications to lecturers
Alternative Flows and Exceptions	**Alternative Flows:** A.1. Conflicts still present At Step 2, if conflicts still exist, the system warns the officer and requires confirmation before publishing. A.2. Re-publishing If a new version is published, the system replaces the previous version and updates the record. A.3. Notification failure If some lecturer notifications fail, the system logs the failures but still publishes the timetable. **Exceptions:** E.1. Publishing failure If the system encounters an error and cannot publish the timetable: E.1.1. The system displays an error message. E.1.2. No timetable is published.

Table 3.1.5.3: UC-22 Publish Timetable

3.1.6 Module 6: Fee & Billing Management Module

3.1.6.1 UC-23 : Configure Tuition Fee

Elements	Detail
Use Case Name	Configure Tuition Fee
Use Case ID	UC-23
Priority	High
Actor(s)	Officer
Description	The Bursar defines the cost per credit hour, fixed semester fees, and student categories for the upcoming academic year.
Pre-conditions	The Academic Year must be created in the system.
Post-conditions	A fee structure template is saved and ready to be used for invoice calculation.
Flow of events	**Actor Actions**

	1. Logs into the system and navigates to the "Fee Configuration" module. 2. Selects the Department (e.g., Engineering) and Student Type. 3. Inputs the Cost Per Credit amount and adds fixed fee items (e.g., Lab Fee, Library Fee). 4. Clicks "Save Configuration".
Alternative Flows & Exception	**Alternative Flow** 1. If a fee structure already exists, the System asks for confirmation to overwrite. The Actor confirms, and the System updates the record. **Exceptions** 1. If the Actor enters text or negative numbers in the currency fields, the System displays an error: "Invalid Currency Format" and prevents saving.

Table 3.1.6.1: UC-23 Configure Tuition Fee

3.1.6.2 UC-24 : Generate Invoices

Elements	**Description**
Use Case Name	Generate Invoices
Use Case ID	UC-24
Super Use Case	None
Priority	High
Actor(s)	Officer
Description	The system automatically generates invoices for all registered students on a specific scheduled date (e.g., start of the semester)
Pre-conditions	Students have registered for courses; Fee structure (UC-01) is active.
Post-conditions	Invoices are generated, saved with "Unpaid" status, and students are notified.

Flow of events	Actor Actions
	1. Triggers the "Semester Billing" batch job at the scheduled time.
Alternative Flows & Exception	**Alternative Flow** 1. **I**f UC-03 returns a value of $0 (e.g., a fully funded student), the System generates a "Statement of Account" marked as "Settled" instead of an Invoice. **Exceptions** 1. If the database locks or crashes during the batch job, the System logs a "Critical Error," aborts the process, and sends an alert to the IT Administrator.

Table 3.1.6.2: UC-24 Generate Invoices

3.1.6.3 UC-25: Calculate Tuition

Elements	Description
Use Case Name	Calculate Tuition
Use Case ID	UC-25
Priority	High
Actor(s)	Billing Engine
Description	The internal mathematical logic to calculate the total payable amount based on registered credits and fees
Pre-conditions	The internal mathematical logic to calculate the total payable amount based on registered credits and fees. UC-02 has been initiated.
Post-conditions	The total calculation is returned to the parent use case.
Flow of events	**System Responses**
	1. Retrieves the number of credits the student is registered for (e.g., 15 credits). 2. Retrieves the Cost Per Credit and fixed fees from the Fee

	Master (UC-01). 3. Performs calculation: (Credits * Cost Per Credit) + Fixed Fees. 4. Checks for active Housing/Meal Plan assignments and adds those costs. 5. Returns the Total Payable Amount to UC-02.
Alternative Flows & Exception	**Alternative Flow :** If the student registered after the cutoff date, the System applies a prorated logic (e.g., charging only 80% of fixed fees). **Exception :** If the Fee Structure (UC-01) is not found for the student's specific department, the System returns a calculation error and flags the student record for manual review.

Table 3.1.6.3: UC-25 Calculate Tuition

3.1.6.4 UC-26: Manage Financial Aid

Elements	**Description**
Use Case Name	Manage Financial Aid
Use Case ID	UC-26
Super Use Case	None
Priority	Medium
Actor(s)	Officer
Description	The system automatically identifies eligible students and applies scholarships or grants to their accounts
Pre-conditions	Student grades/data are finalized; Scholarship rules are defined.
Post-conditions	The student's outstanding balance is reduced by the aid amount.
Flow of events	**Actor's Actions**

	1. Officer triggers the Financial Aid Allocation process. 2. System retrieves all students with finalized academic and financial records. 3. System evaluates each student against predefined scholarship and grant eligibility rules (e.g., GPA threshold, aid limits). 4. System verifies that the student has not exceeded the maximum allowable financial aid cap. 5. System applies the appropriate financial aid code to the eligible student's account. 6. System recalculates the student's outstanding balance after aid application. 7. System records the allocation and displays a summary of applied financial aid.
Alternative Flows & Exception	**Alternative Flow** **A1: Eligibility Breach Detected** A1.1 System detects that a student no longer meets the eligibility criteria (e.g., GPA below threshold). A1.2 System removes or marks the pending financial aid as revoked. A1.3 System recalculates the student's outstanding balance. **Exceptions** **E1: Aid Amount Exceeds Total Bill** E1.1 System detects that the aid amount exceeds the student's total bill. E1.2 System caps the aid amount at the total bill value unless the aid is flagged as **Refundable**. E1.3 System updates the student ledger accordingly.

Table 3.1.6.4: UC-26 Manage Financial Aid

3.1.6.5 UC-27: Track Payments & Outstanding Fees

Elements	Description
Use Case Name	Track Payments & Outstanding Fees
Use Case ID	UC-27
Super Use Case	None
Priority	High

Actor(s)	Officer
Description	Handles the recording of manual payments (wires/checks) and the monitoring of overdue accounts.
Pre-conditions	Invoices exist in the system.
Post-conditions	Ledger is updated (Paid), or a Defaulter list is generated.
Flow of events	**Actor Actions**
	1. Enters Student ID and Payment Details (Amount, Reference #) received via bank transfer. **2.** Submits the record. **3.** Navigates to reports and requests "Outstanding Dues Report".
Alternative Flows & Exception	**Alternative Flow** 1. If Amount Paid < Invoice Amount, the System marks status as "Partially Paid". **Exceptions** 1. If the Reference # already exists, the System prevents saving and shows "Duplicate Transaction Detected."

Table 3.1.6.5: UC-27 Track Payments & Outstanding Fees

3.1.6.6 UC-28: Process Refund

Elements	**Description**
Use Case Name	Process Refund
Use Case ID	UC-28
Super Use Case	None
Priority	Low
Actor(s)	Officer
Description	The Bursar manually processes a refund for a student who has overpaid or withdrawn from a course.
Pre-conditions	Student has a credit balance (negative outstanding amount).
Post-conditions	Student balance is zeroed out; Refund instruction is sent to the

	bank.	
Flow of events	**Actor Actions**	**System Responses**
	1.Searches for the Student ID and selects "Process Refund". 2. Enters Refund Amount and Reason (e.g., "Course Withdrawal"). 3. Confirms the transaction.	1. Displays the current credit balance and refund eligibility status. 2. Validates that Refund Amount <= Credit Balance. 3. Records the refund in the ledger and updates the balance to zero. 4. Triggers a payment instruction to the external bank gateway.
Alternative Flows & Exception	**Alternative Flow** 1. Instead of a bank refund, the Actor selects "Transfer to Next Semester". The System moves the credit to a holding account. **Exceptions** 1. If the Actor tries to refund more than the available credit balance, the System blocks the action.	

Table 3.1.6.6: UC-28 Process Refund

3.1.6.7 UC-29: View Account Statements

Elements	**Description**
Use Case Name	View Account Statements
Use Case ID	UC-29
Super Use Case	None
Priority	Medium
Actor(s)	Student
Description	The student logs in to view their financial ledger, transaction history, and can generate specific billing statements or tax

	forms.
Pre-conditions	Student is logged into the University Portal.
Post-conditions	Student has viewed or downloaded the statement.
Flow of events	**Actor Actions**
	1. Navigates to the "My Finance" dashboard. 2. Selects a specific Term or clicks "Generate Tax Form". 3. Clicks "Download PDF"
Alternative Flows & Exception	**Alternative Flow** 1. The user clicks "Print". The System formats the view for a printer-friendly version. **Exceptions** 1. If a new student tries to view a tax form for a year they were not enrolled, the System displays "No records found."

Table 3.1.6.7: UC-29 View Account Statements

3.2 Class Diagram

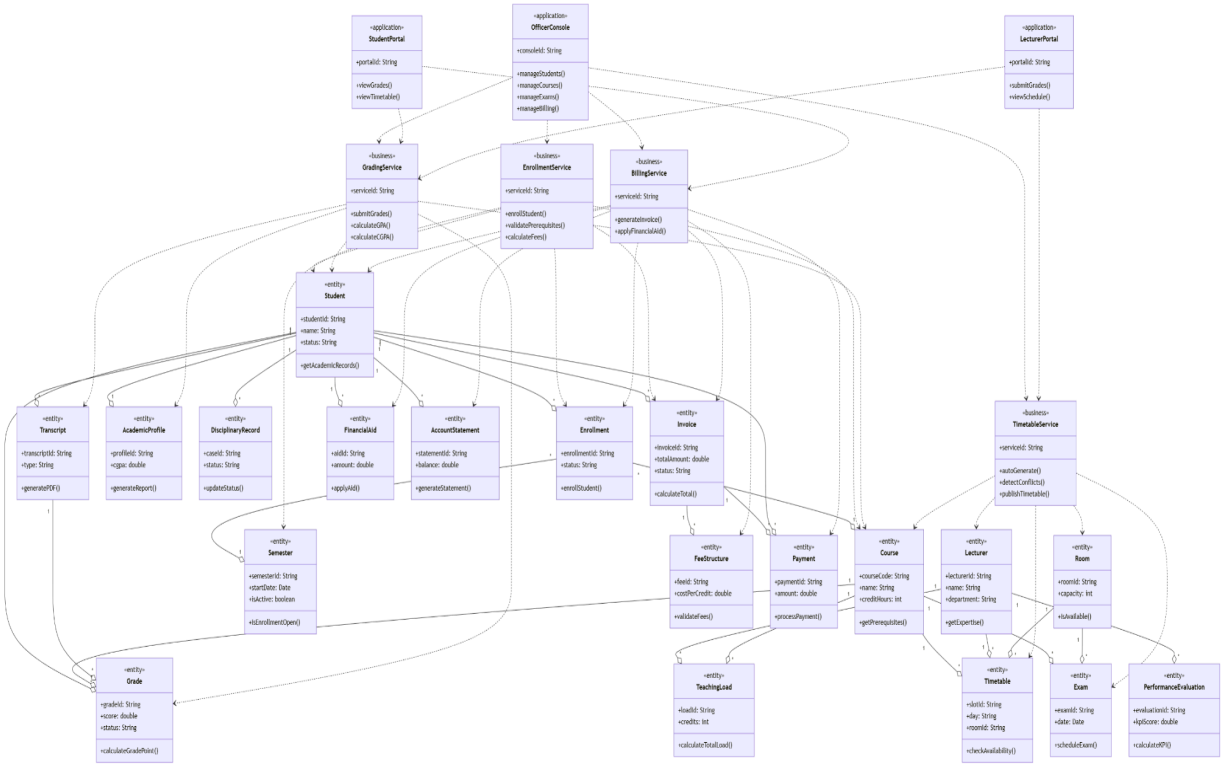

Figure 3.2: Class Diagram

4.0 Dependency Analysis of the Monolithic Application

No	Class Name	Depends On (Dependency Depth)	Dependency Depth
1	**Student**	-	**0**
2	**Course**	-	**0**
3	**Lecturer**	-	**0**
4	**Semester**	-	**0**
5	**Room**	-	**0**
6	**Enrollment**	Student (0), Course (0), Semester (0)	**1**
7	**Grade**	Student (0), Course (0)	**1**
8	**AcademicProfile**	Student (0)	**1**
9	**DisciplinaryRecord**	Student (0)	**1**

10	**TeachingLoad**	Lecturer (0), Course (0)	**1**
11	**PerformanceEvaluation**	Lecturer (0)	**1**
12	**Timetable**	Course (0), Lecturer (0), Room (0)	**1**
13	**Exam**	Course (0), Room (0)	**1**
14	**Transcript**	Student (0), Grade (1)	**2**
15	**Invoice**	Student (0)	**1**
16	**FeeStructure**	Invoice (1)	**2**
17	**Payment**	Invoice (1), Student (0)	**2**
18	**FinancialAid**	Student (0)	**1**
19	**AccountStatement**	Student (0)	**1**
20	**EnrollmentService**	Student (0), Enrollment (1), Course (0), Semester (0), Invoice (1)	**2**

21	**GradingService**	Student (0), Course (0), Grade (1), AcademicProfile (1), Transcript (2)	**3**
22	**BillingService**	Student (0), Enrollment (1), Invoice (1), FeeStructure (2), Payment (2), FinancialAid (1), AccountStatement (1)	**3**
23	**TimetableService**	Course (0), Lecturer (0), Room (0), Timetable (1), Exam (1)	**2**
24	**OfficerConsole**	EnrollmentService (2), GradingService (3), BillingService (3), TimetableService (2)	**4**
25	**StudentPortal**	GradingService (3), BillingService (3)	**4**
26	**LecturerPortal**	GradingService (3), TimetableService (2)	**4**

Table 4.0: Monolithic System Dependency Depth Analysis

Table 4.0 illustrates the dependencies between different classes in the university management system. A class that depends on another implies that a change in the dependency target will likely require modifications to the dependee. The dependency depth indicates the level of coupling, where higher values represent a greater susceptibility to cascading changes throughout the system.

At the foundation of the system, classes such as Student, Course, Lecturer, Semester, and

Room have a dependency depth of 0. These are independent entities that do not rely on any other classes, making them the most robust and stable components of the architecture. Changes made elsewhere in the system will not affect these base entities.

The next layer consists of classes with a dependency depth of 1, such as Enrollment, Grade, AcademicProfile, DisciplinaryRecord, TeachingLoad, PerformanceEvaluation, Timetable, Exam, Invoice, FinancialAid, and AccountStatement. These classes depend directly on the foundational entities. For instance, the Enrollment class depends on Student, Course, and Semester to function. Similarly, TeachingLoad is directly coupled with Lecturer and Course. Because these classes rely on the base entities, they are vulnerable to any structural changes made to those depth 0 classes.

Classes with a dependency depth of 2, including Transcript, FeeStructure, Payment, EnrollmentService, and TimetableService, represent a moderate level of coupling. These classes often bridge multiple layers. For example, the Transcript class depends directly on Grade (Depth 1) and indirectly on Student (Depth 0). The EnrollmentService coordinates multiple entities like Student, Enrollment, Course, Semester, and Invoice, making it a more complex component that is sensitive to changes in both foundation and secondary entities.

The business logic layer, represented by GradingService and BillingService, exhibits a high dependency depth of 3. The GradingService is tightly coupled with a wide array of classes including Student, Course, Grade, AcademicProfile, and Transcript. Any modification to the way grades are recorded or how transcripts are generated will necessitate changes in this service. This layer is highly susceptible to the "ripple effect," where a minor change in a low-level entity can force extensive refactoring in these services.

At the highest level of the hierarchy, the application portals—OfficerConsole, StudentPortal, and LecturerPortal—possess a dependency depth of 4. These are the most fragile components of the system because they act as the primary interface and depend on the complex business services below them. The OfficerConsole, for instance, depends on the EnrollmentService, GradingService, BillingService, and TimetableService. This extreme level of coupling means that almost any significant change in the underlying system architecture will likely require updates to these portal classes.

This analysis highlights that the monolithic nature of the system creates tight coupling at the

service and application levels. While foundational entities are isolated and easy to maintain, the top-level portals and services are highly sensitive to system-wide changes, requiring careful management during development and maintenance.

5.0 Component-Based Software Engineering (CBSE) Application

5.1 Application of Component Development Rules

To componentize the University Management System, five rules have been implemented as discussed in the previous section.

Rule 1: Separate the entity classes in a common library.

Entity classes that carry data and represent domain objects are separated into a Base Library that is shared by all other components. These classes do not contain business logic and only act as data carriers.

In this system, the Base Library consists of the following entity classes:

- Student
- AdminOfficer
- AcademicProfile
- TeachingLoad
- FeeItem
- Enrollment
- AccountStatement
- DisciplinaryRecord
- PerformanceEvaluation

These classes store core university data such as student profiles, academic records, fee items, and teaching loads. Since they only carry data, they are not considered components. Other components can instantiate and pass these objects to perform their business operations.

Rule 2: Group a set of classes that together provide independent business functionality into a component.

Classes that work together to deliver a specific business function are grouped into business components. Each group of classes works independently to deliver a complete business function, although they may depend on entities from the Base Library.

Table 5.1.1: Business Components

No.	Component	Classes	Description

1	Finance & Billing Component	FinancialAid, Invoice, Payment, FeeStructure	These classes collectively manage all financial operations such as billing, student payments, financial aid management, and defining university fee structures.
2	Enrollment Component	Enrollment, Course, Semester	These classes manage student enrollment processes, including registering courses, handling semester data, and maintaining enrollment records.
3	Examination & Grading Component	Grade, Exam, Transcript	These classes handle examination management, grading, and transcript generation for students.
4	Scheduling & Venue Component	Timetable, Room, Lecturer	These classes manage class scheduling, lecturer allocation, room booking, and timetable generation.

Table 5.1.1 shows the list of business components, their classes, and the reasons for grouping them.

Rule 3: Expose the business functionality provided by the component.

Each business component exposes its functionality through a provided interface.

- The Finance & Billing Component exposes the BillingService interface, which provides methods for handling invoices, processing payments, managing financial aid, and retrieving account statements.
- The Enrollment Component exposes the EnrollmentService interface, which allows students to enroll in courses, manage semesters, and retrieve enrollment records.
- The Examination & Grading Component exposes the GradingService interface, which provides access to examination records, grade computation, and transcript generation.
- The Scheduling & Venue Component exposes the TimetableService interface, which allows management of timetables, classroom allocation, and lecturer scheduling.

These interfaces encapsulate internal implementations and expose only the necessary services to external components.

Rule 4: Group mutually dependent classes together.

There are no circular dependencies between the business components.

An Application Component is created to handle application-level logic and user interaction, which includes:

- StudentPortal
- OfficerConsole
- LecturerPortal

This component consumes the following interfaces:

- BillingService
- EnrollmentService
- GradingService
- TimetableService

The Application Component orchestrates interactions between the user interfaces and the underlying business components.

Rule 5: Add interface definitions into the base library.

Finally, all service interface definitions such as:

- BillingService
- EnrollmentService
- GradingService
- TimetableService

These interfaces are added to the Base Library together with the entity classes. This allows all components to reference the same interface contracts while keeping their implementations encapsulated.

6.0 Componentized Application Design using UML

6.1 Component Diagram

Figure 6.1 below illustrates the new architecture. The diagram clearly shows the separation of concerns, with components communicating strictly within a hierarchical structure.
Link for Component Diagram

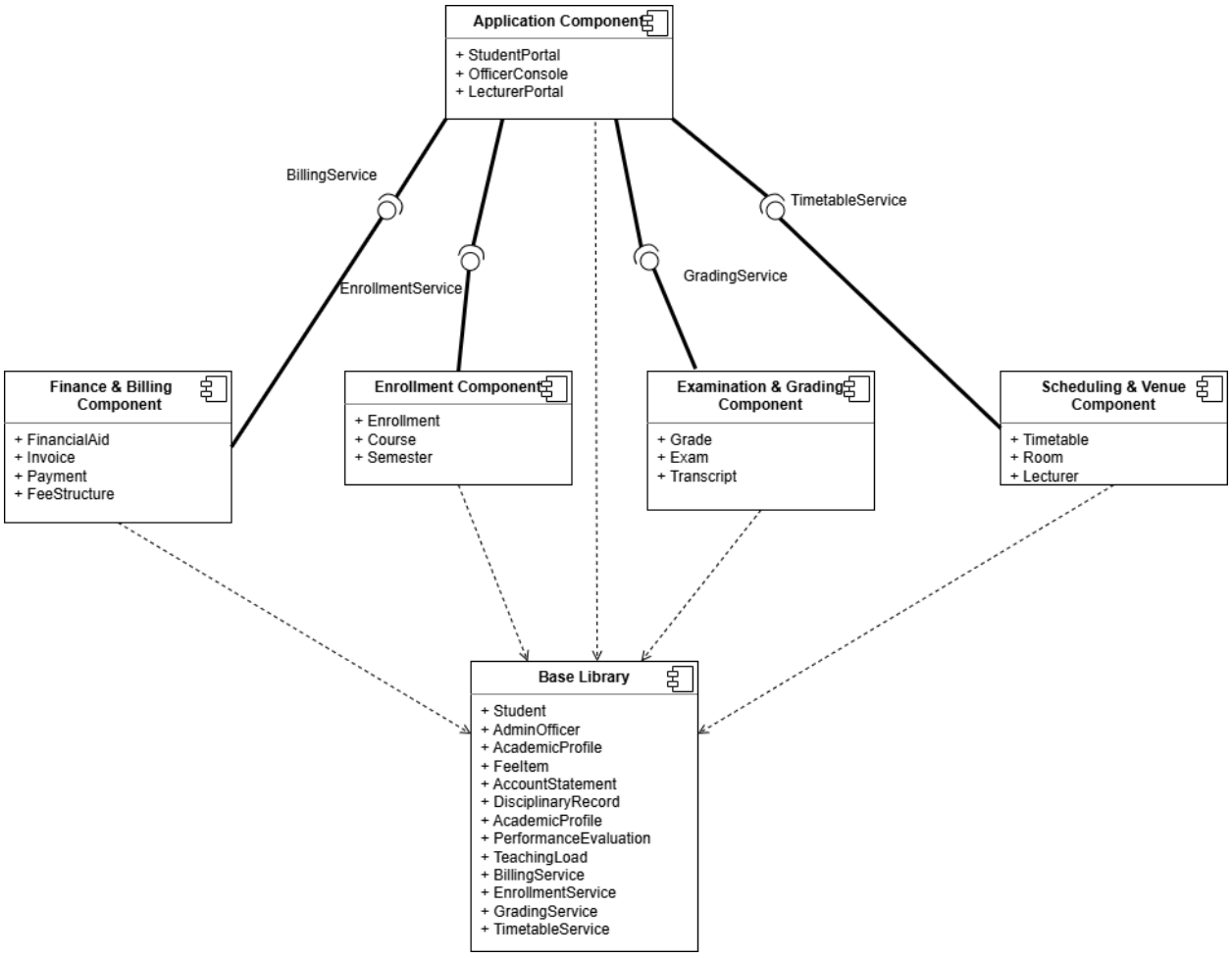

Figure 6.1: Component Diagram of the University Management System showing the transition from direct class association to interface-based coupling

7.0 Dependency Analysis of the Componentized Application

No	Class	Depends On	Dependency Depth
1	Student	-	0
2	AdminOfficer	-	0
3	AcademicProfile	-	0
4	TeachingLoad	-	0
5	FeeItem	-	0
6	Enrollment	-	0
7	AccountStatement	-	0
8	DisciplinaryRecord	-	0
9	Performance Evaluation	-	0
10	BillingService	-	0
11	EnrollmentService	-	0
12	GradingService	-	0
13	TimetableService	-	0
14	FinancialAid	BillingService	1
15	Invoice	BillingService	1
16	Payment	BillingService	1
17	FeeStructure	BillingService	1
18	Enrollment	EnrollmentService	1
19	Course	EnrollmentService	1
20	Semester	EnrollmentService	1
21	Grade	GradingService	1
22	Exam	GradingService	1

23	Transcript	GradingService	1
24	Timetable	TimetableService	1
25	Room	TimetableService	1
26	Lecturer	TimetableService	1
27	StudentPortal	BillingService, EnrollmentService, GradingService, TimetableService	1
28	OfficerConsole	BillingService, EnrollmentService, GradingService, TimetableService	1
29	LecturerPortal	BillingService, EnrollmentService, GradingService, TimetableService	1

Table 7.0.1 : Coupling Metrics of Componentized System

Coupling metrics tabulated in Table 7.0.1 are a result of a comprehensive dependency analysis performed on the componentized Student Management System. The architecture is characterized by a highly streamlined and flat hierarchy, where the maximum dependency depth is restricted to 1 across all 29 analyzed entities.

Foundational elements, including core data models such as Student, AdminOfficer, and AcademicProfile, along with the primary service interfaces—BillingService, EnrollmentService, GradingService, and TimetableService—are positioned at Depth 0. These components serve as the independent pillars of the system, possessing no external dependencies within this architectural scope.

The system's functional classes, such as Invoice, Grade, and Course, are positioned at Depth 1, relying directly on their respective service interfaces for operation. This structural pattern extends to the application’s user-facing layer, where the StudentPortal, OfficerConsole, and LecturerPortal depend on the various services to aggregate functionality for the end-user.

By routing all dependencies through stable service interfaces rather than allowing direct

coupling between concrete classes, the system ensures that its components remain decoupled and resilient. This design approach significantly improves the quality of dependencies, making the system less brittle. The resulting modular architecture provides a high degree of maintainability and scalability, ensuring that the system is robust and easily adaptable to future functional requirements.

No	Component	Depends on	Dependency Depth
1	**Base Library**	None	0
2	**Finance & Billing Component**	Base Library	1
3	**Enrollment Component**	Base Library	1
4	**Examination & Grading Component**	Base Library	1
5	**Scheduling & Venue Component**	Base Library	1
6	**Application Component**	Finance&Billing Component, Enrollment Component, Examination & Grading Component, Scheduling & Venue Component, Base Library	1

Table 7.0.2: Component Dependencies

At runtime, the Base Library remains independent with a dependency depth of 0. The Finance & Billing Component, Enrollment Component, Examination & Grading Component, and

Scheduling & Venue Component each rely on the Base Library, resulting in a dependency depth of 1.

The Application Component depends on all other components—including the Finance & Billing, Enrollment, Examination & Grading, and Scheduling & Venue components—as well as the Base Library, resulting in a runtime dependency depth of 2. This hierarchical structure ensures that the majority of components remain modular and rely only on the stable Base Library, while the Application Component integrates and coordinates the specific functionalities of the entire system into a unified user interface.

8.0 Conclusion

This project successfully demonstrated the transformation of a traditional, monolithic University Management System (UMS) into a modular, component-based architecture by applying the principles of Component-Based Software Engineering (CBSE). By following a structured approach—from identifying monolithic pain points to implementing specific componentization rules—the system's overall architectural health has been significantly improved.

The initial analysis of the monolithic design revealed critical vulnerabilities, most notably tight coupling and circular dependencies among core modules like Student Management and Finance & Billing. These issues led to high dependency depths (up to level 4), making the system brittle, difficult to scale, and prone to "ripple effect" failures where minor changes in one class necessitated extensive refactoring across the entire codebase.

The transition to a componentized architecture successfully addressed these challenges through several key strategies:

1. **Creation of a Base Library:** Separating core entity classes and service interfaces into a foundational, independent library (Depth 0) provided a stable "source of truth" for the entire system.
2. **Interface-Based Coupling:** By routing all inter-component communication through stable interfaces (BillingService, EnrollmentService, etc.) rather than direct class associations, the system achieved true encapsulation and horizontal decoupling.
3. **Tiered Hierarchy:** The adoption of a layered structure—consisting of foundational data, business logic services, and an orchestrating application layer—reduced the maximum dependency depth and simplified the system's runtime logic.

The quantitative results of the dependency analysis (Table 7.0.1 and 7.0.2) provide clear evidence of this improvement. Most system components now operate at a depth of 0 or 1, with the Application Component serving as the primary orchestrator at depth 2. This streamlined hierarchy ensures better fault isolation, independent scalability of individual modules, and enhanced maintainability.

In conclusion, the implementation of CBSE concepts has transformed the UMS from a rigid monolith into a robust, flexible, and adaptable platform. This modular foundation not only resolves existing technical debt but also positions the university to easily incorporate future technological advancements and evolving academic requirements with minimal risk to system stability.