PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

ACADEMIC MANAGEMENT SYSTEM

Course: Software Architecture

Course Class: CSE703110-1-2-25(N02)

Group 7:

1. Le Thi Kieu Trang **ID: 23010502**

2. Quach Huu Nam **ID: 23012358**

3. Trieu Tien Quynh **ID: 23010648**

Hanoi, January 8, 2026

TASK ASSIGNMENT TABLE

No.	Student ID	Full Name	Assigned Tasks
1	23010502	Le Thi Kieu Trang	Use Case Modeling Conclusion & Reflection

PREFACE

In the context of rapid digital transformation, the application of information technology in higher education management is no longer just an option but an inevitable trend. Traditional academic management systems, or legacy Monolithic systems, often face significant challenges regarding scalability, maintenance, and the integration of new features as the volume of students and data continues to grow.

Recognizing these challenges, our team decided to undertake the project titled "Building an Academic Management System based on Microservices Architecture." The project focuses on addressing the management of student information, courses, credit enrollment, grading, and more, by decomposing the system into independent services that communicate flexibly via standard RESTful APIs.

This report details the system analysis, design, and implementation process. It covers the construction of core services such as Identity, Student, and Grade Services using Python (Flask), as well as the design of a highly interactive Single Page Application (SPA) user interface. Notably, the project delves into specific distributed architecture techniques, such as Asynchronous Communication and shared database management.

We hope that the results of this project will serve as a useful reference model for the application of modern software architecture to real-world management problems.

To complete this project, alongside the efforts of our team members, we have received dedicated attention, encouragement, and guidance from M.S.Vu Quang Dung. Throughout the implementation of this topic, he spared no effort in imparting knowledge, guiding our problem-solving mindset, and providing valuable feedback to help us refine the system, ranging from Microservices organization to source code optimization.

Although the team has made every effort to apply learned knowledge to practice, due to time constraints and limited practical experience, the project inevitably contains shortcomings. We sincerely look forward to receiving your feedback to further improve the topic and gain valuable lessons for our future careers.

We would like to express our sincere gratitude!

1. Overview

This document analyzes the functional requirements of an Academic Management System using use case modeling. The use case diagram illustrates how users interact with the system to perform core academic tasks, while internal system behaviors such as authentication, GPA calculation, and notification handling are modeled as included use cases. This approach ensures a clear separation between user-triggered actions and internal processing, providing a concise and structured overview of the system's functionality.

2. Use Case Modeling

- ❖ Use Case Diagram

Figure 1: Use Case Diagram – User Interaction

This diagram presents the primary interactions between external actors and the system, focusing on architecturally significant use cases rather than detailed operational behaviors.

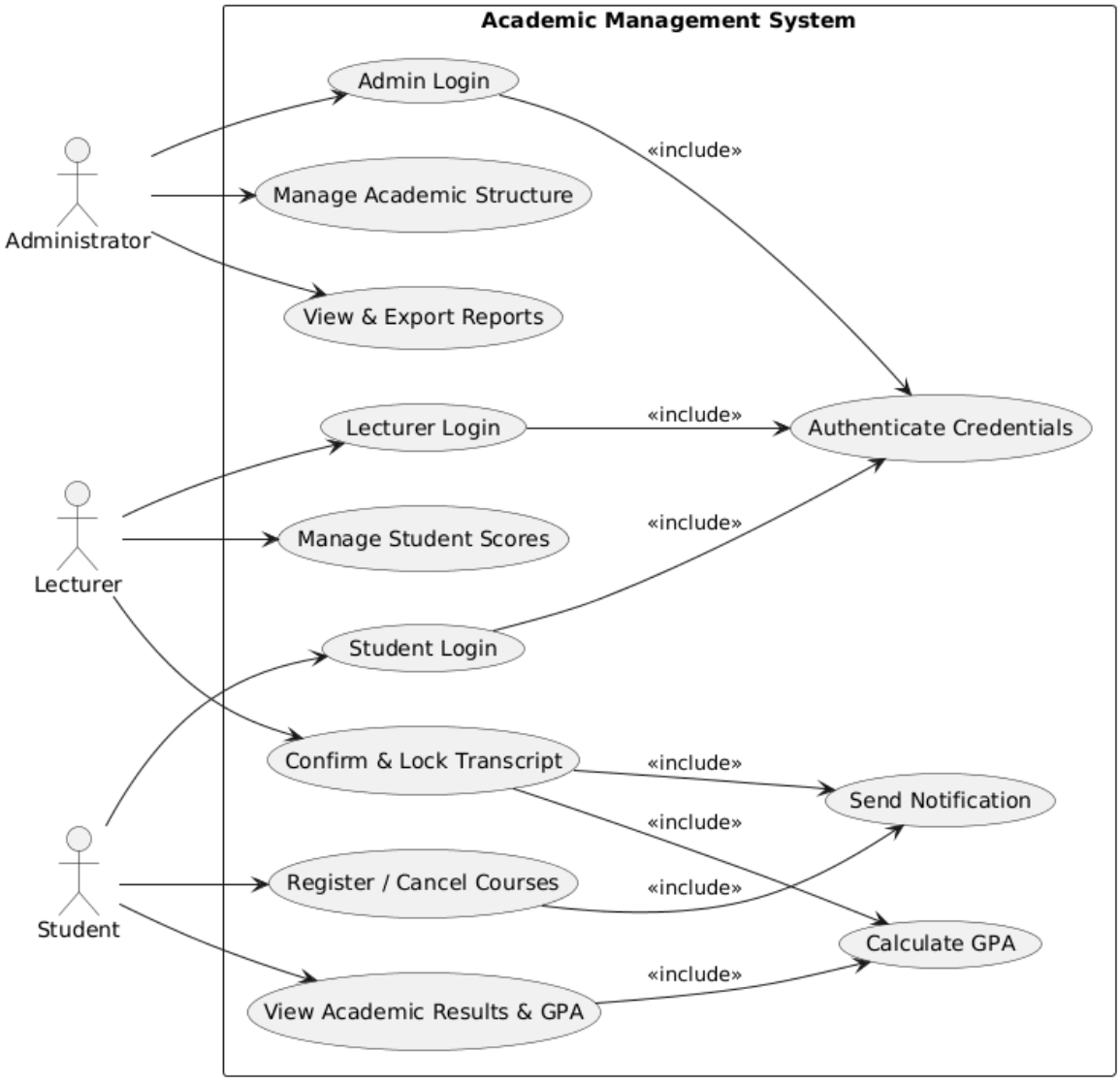
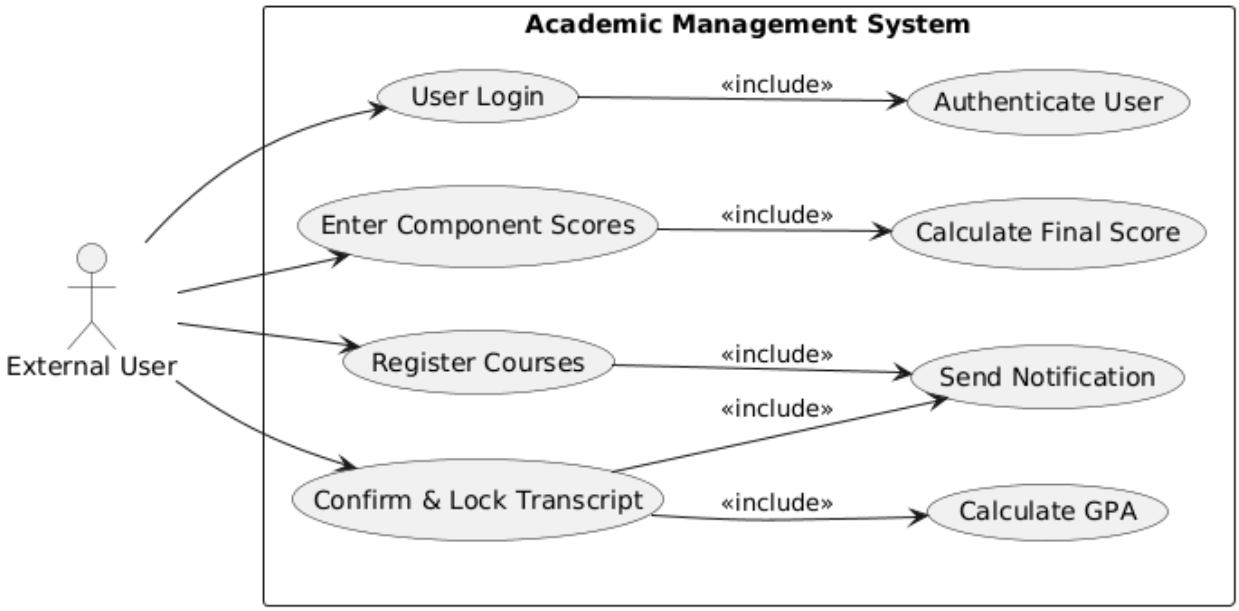


Figure 2: Use Case Diagram – System Functionalities

In the Use Case Diagram – System Functionalities, the system itself is not modeled as an actor. Instead, internal system behaviors such as authentication, GPA calculation, and notification sending are represented as included use cases using the <<include>> relationship, as they are automatically executed as part of user-initiated actions.

The User actor represents a generalized external user, while role-specific interactions are detailed in other diagrams.



- Specification and Construction of Administrator Use Case Group

Overview Description

The Administrator is the actor responsible for managing core academic and organizational data within the system. Administrator interactions mainly focus on system access control and high-level data management, including universities, faculties, majors, courses, and student records. In addition, the Administrator is authorized to monitor academic outcomes by viewing and exporting confirmed grade sheets and generating summary reports to support management and assessment activities.

Use Case Specification

+ UC-A1 Administrator Login

Use Case Name	Administrator Login		
Created By	Development Team	Last Updated By	System Officer
Created Date	Jan 1, 2026	Last Modified Date	Jan 8, 2026
Description	Allows the Administrator to authenticate their identity using a username and password		

	to access the system's administrative functions. This function utilizes the centralized Identity Service.
Actors	- Administratoristrator (Primary Actor) - Identity Service (Authentication System)
Pre-conditions	1. The Identity Service is operational. 2. The user has accessed the Login page. 3. The Administrator account exists in the system database.
Post-conditions	Success: 1. The system returns an Access Token (e.g., fake-jwt-token-for-Administratoristrator). 2. The user is redirected to the Administratoristrator Dashboard. 3. The Administratoristratoristration menu displays full functions according to permissions. Failure: 1. The user remains on the login page. 2. The system displays a corresponding error message.
Main Flow	1. Access Page: - Administrator accesses the admin portal URL; the system displays the Login Form. 2. Enter Credentials: - Administrator enters Username and Password. - Clicks the "Login" button. 3. Validation: - The System (Frontend) performs preliminary validation: Fields must not be empty. 4. Send Request: - Frontend sends an HTTP POST request to the API. - Payload: { "username": "admin", "password": "..." } 5. Authentication (Backend): - Identity Service receives the request. - Checks if the username exists. - Compares the submitted password with the stored hash in the Database. 6. Generate Token: - If credentials are correct, Identity Service generates a success response. - Response: { "message": "Login successful", "token": "...", "role": "admin" } 7. Redirect:

	- Frontend receives the Token and saves it (LocalStorage/Cookie). - Frontend checks the role: If role == "admin" → Redirect to the Dashboard. 8. Use Case Ends.
Alternative Flow	Step 5a. Incorrect credentials: → Identity Service does not find the user or the password matches. → Returns HTTP 401 with JSON: { "error": "Invalid credentials" }. → Frontend displays a red notification: *"Incorrect username or password."* *Step 7a. Non-Administratoristrator privileges:* → User logs in with a valid account but the role is "Student" or "Lecturer". → Syste System detects incorrect role permission. → Notification: *"You do not have permission to access the Administrator page"* and redirects to the appropriate portal.
Exceptions	1. Service Connection Loss: → Identity Service (Port 5004) is down. → Frontend receives a connection error (Connection Refused). → Displays notification: *"System is under maintenance, please try again later."*
Requirements	1. Security: Passwords sent must be encrypted via HTTPS (in Production environment). 2. Performance: Login response time must be < 1 second.

+ *UC-A2* Manage Academic Structure

Use Case Name	Manage Academic Structure		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Administrators to manage the hierarchical organization of the institution. This includes managing Universities (Root), Faculties (Child of University), and Majors (Child of Faculty).		
Actors	- Administrator (Primary Actor) - Uni Service (Manages Universities) - Faculty Service (Manages Faculties) - Major Service (Manages Majors)		
Pre-conditions	1. Administrator has successfully logged in. 2. All structure-related services (uni, faculty, major) are operational.		

Post-conditions	Success: 1. Structure data is created/updated/deleted in the respective Databases. 2. The hierarchical tree is refreshed on the interface. Failure: 1. Data remains unchanged. 2. Error message displayed (e.g., Constraint Violation).
Main Flow	1. Select Entity Type: - Administrator selects "Academic Structure" menu. - Chooses the entity to manage: University, Faculty, or Major. 2. View List (Read): - System calls the corresponding API based on selection: + University: GET /api/universities + Faculty: GET /api/faculties (requires University ID filter) + Major: GET /api/majors (requires Faculty ID filter) - Displays the data table. 3. Add New Entity (Create): - Admin clicks "Add New". - If University: Enters Code, Name, Address. - If Faculty: Selects Parent University → Enters Code, Name - If Major: Selects Parent Faculty → Enters Code, Name. - Clicks "Save". - System calls POST to the respective Service. 4. Update Entity (Update): - Admin selects a row and modifies information. - Clicks "Update". - System calls PUT API to save changes. 5. Delete Entity (Delete): - Admin clicks "Delete". - System checks for child data (Referential Integrity). - If safe, System calls DELETE API. - System notifies: *"Deleted successfully"*. *6. Use case kết thúc.*
Alternative Flow	Step 3a. Duplicate Code: → Administrator enters a code that already exists.

	→ Service returns 409 Conflict. → System notifies: *"Entity Code already exists."* *Step 5a. Data Integrity Violation (Delete Blocked):* → Administrator tries to delete a Faculty that still has Majors assigned. → System blocks the request. → Notification: *"Cannot delete this Faculty because it contains Majors. Please remove child data first."*
Exceptions	1. Service Unavailable: → One of the microservices is down. → System displays: *"Unable to load structure data. Please try again later."*
Requirements	1. Hierarchy Rule: A Major must belong to a Faculty; a Faculty must belong to a University. 2. Uniqueness: Codes must be unique within their scope (e.g., Major Codes must be unique within a Faculty).

+ *UC-A3* View & Export Reports

Use Case Name	View & Export Reports		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Administrators to view academic statistics (e.g., Enrollment status, Grade distribution, Student list per major) and export these reports to external files (Excel) for administrative purposes.		
Actors	- Administrator (Primary Actor) - Grade Service (Data Source) - Enrollment Service (Data Source) - Student Service (Data Source)		
Pre-conditions	1. Administrator has successfully logged into the system. 2. Data exists in the system (Students, Grades, Enrollments). 3. Relevant services are operational.		
Post-conditions	Success: 1. Statistical data is displayed visually (Table/Chart). 2. The report file is successfully downloaded to the Administrator's device. Failure:		

	1. Error notification is displayed. 2. File download fails.
Main Flow	1. Access Report Dashboard: - Administrator selects the "Reports & Statistics" menu. - System displays a list of available report types (e.g., "Semester GPA Summary", "Enrollment Statistics", "Student List"). 2. Configure Parameters: - Administrator selects a Report Type. - Sets filters: Semester (e.g., Fall 2025), Faculty (e.g., IT), Major. - Clicks "Generate Report". 3. Data Retrieval: - System calls the relevant API (e.g., GET /api/grades/reports/gpa or GET /api/enrollment/stats). - The Service aggregates data and returns JSON. 4. View Report: - System renders the data into a Table or Chart on the screen. 5. Export Data: - Administrator clicks "Export to Excel" (or PDF). - System calls the Export API (e.g., GET /api/grades/reports/export?format=xlsx). - The Backend generates the file and streams it back. - The Browser initiates the file download. 6. Use Case Ends.
Alternative Flow	Step 3a. No Data Found: → The filter criteria yield no results (e.g., A new semester with no grades yet). → System notifies: *"No data available for the selected criteria."* → Export button is disabled. *Step 5a. Large Dataset (Async Export):* → The report contains thousands of records (taking > 30s to generate) → System notifies: *"Report is being generated. You will be notified when it is ready."* → The task is pushed to a Background Queue.
Exceptions	1. Service Timeout: → The aggregation query takes too long, causing a Gateway Timeout (504). → System suggests: *"Data is too large. Please narrow down the date range or use Export feature.*

	2. File Generation Error: → Error occurs during file creation (e.g., Library library failure). → Notification: *"Failed to generate file. Please contact IT support."*
Requirements	1. Formats: Must support .xlsx (Excel) for data manipulation. 2. Accuracy: Statistics must reflect real-time data (or cached data not older than 24 hours, depending on configuration).

❖ Specification and Construction of Lecturer Use Case Group

Overview Description

The Lecturer Module is a core component responsible for assessing and recording student academic results. The primary actor of this module is the Lecturer, who directly teaches and is responsible for the accuracy of grades.

The main objective of this module is to provide a closed-loop process ranging from receiving assigned classes, entering grades (manually or via Excel import), automatically calculating final grades, to locking the grade sheet for official archiving.

Use Case Specification

+ UC-L1 Lecturer Login

Use Case Name	Lecturer Login		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Lecturers to authenticate their account information to access the teaching management and grading module.		
Actors	- Lecturer		
Pre-conditions	1. The Lecturer has been granted an account (Username/Password) in the system. 2. The Identity Service is operational.		
Post-conditions	Success: 1. The system issues an Authentication Token. 2. The Lecturer is redirected to the Lecturer Homepage (Lecturer Dashboard). Failure: 1. The user remains on the login page. 2. An error message is displayed.		
Main Flow	1. Access: - Lecturer accesses the general system Login page. 2. Input: - Enters Username (or Official Email) and Password. - Clicks the "Login" button. 3. Authentication Processing: - The system calls API POST /api/login to the Identity Service.		

	- The Service checks login credentials against the Database. 4. Authorization: - After the password is verified, the system checks the account's Role. - Confirms the Role is "Lecturer". - The system returns the Token and navigation information. 5. Redirect: - The interface redirects to the Class List screen (Instead of the Administrator screen). 6. Use Case Ends.
Alternative Flow	Step 3a. Invalid Information: → Wrong Username or Password entered. → System error: *"Incorrect username or password."* Step 4a. Unauthorized Role: → Account credentials are correct but the Role is "Student" or "Administrator" (in cases where specific access is restricted). → System notifies: *"This account does not have permission to access Lecturer functions."*
Exceptions	1. Identity Service timeout: → System displays: *"Cannot connect to authentication server."* 2. Account Locked: → The Lecturer has resigned or the account is locked. → System notifies: *"Your account has been disabled. Please contact the Administrator."*
Requirements	1. Passwords must be encrypted during transmission. 2. Response time < 3 seconds. 3. The login interface should be user-friendly and support a "Forgot Password" feature (Optional).

+ UC-L2 Manage Student Scores

Use Case Name	Manage Student Scores		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Lecturers to view the list of course classes assigned to them by the university for the current semester or previous semesters.		
Actors	- Lecturer (Primary Actor) - Grade Service (System) - Course Service (Verifies assignment)		
Pre-conditions	1. Lecturer has logged in successfully. 2. The Lecturer is officially assigned to the class. 3. The Grade Sheet is NOT yet Locked (Status is OPEN or DRAFT).		

Post-conditions	Success: 1. Grades are saved to the Database. 2. The total_score (Course Grade) is automatically re-calculated based on weights. Failure: 1. Data remains unchanged. 2. Error message displayed (e.g., Invalid score range).
Main Flow	1. Select Class: - Lecturer navigates to "My Courses" and selects a specific class. - System displays the list of students enrolled in that class. 2. Input Scores: - Lecturer enters values into the columns: Attendance (CC), Mid-term (GK), Final (CK). - *Note:* Lecturer can enter scores for multiple students at once. 3. Validation (Frontend): - As the Lecturer types, the interface checks if the value is numeric and within the range [0, 10]. 4. Save: - Lecturer clicks "Save Grades". - System calls PUT /api/grades/batch-update to Grade Service. 5. Processing: - Grade Service validates the Grade Sheet status (must be Unlocked). - Updates records in the Database. - Triggers a background calculation for the Final Total Score (UC-SYS3). 6. Feedback: - System notifies: *"Grades saved successfully."* - The interface refreshes with the updated data. 7. Use Case Ends.
Alternative Flow	Step 3a. Invalid Input: → Lecturer enters a score like "11" or "-5". → System highlights the cell in red and disables the "Save" button. → Message: *"Score must be between 0 and 10."* Step 5a. Concurrent Edit Conflict: → Two lecturers (e.g., Main and Assistant) try to save scores for the same student at the same time.

	→ System detects version conflict. → Notification: *"Data has been modified by another user. Please refresh and try again."*
Exceptions	1. Locked Grade Sheet: → Lecturer attempts to save, but the Grade Sheet was locked (by Admin or previously by Lecturer). → Grade Service returns 403 Forbidden. → System notifies: *"This grade sheet is locked and cannot be edited."*
Requirements	1. Range: Scores must be strictly between 0.0 and 10.0. 2. Log: Every score change must be logged (Who changed, Old Value, New Value) for audit purposes. 3. Batch Processing: The API should support batch updates (saving the whole class at once) to reduce network latency.

+ UC-L3 Confirm & Lock Transcript

Use Case Name	Confirm & Lock Transcript		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Lecturers to finalize the grading process for a specific class. Once locked, the grades become official, the Lecturer can no longer edit them, and the system automatically triggers GPA calculation for the students.		
Actors	- Lecturer (Primary Actor) - Grade Service (System) - Notification Service (System)		
Pre-conditions	1. Lecturer has logged in and selected the class. 2. Component scores have been entered. 3. The Grade Sheet status is currently OPEN or DRAFT.		
Post-conditions	Success: 1. Grade Sheet status changes to LOCKED. 2. Editing is disabled for the Lecturer. 3. System triggers UC-SYS4 (Calculate GPA). 4. Students receive notifications (via UC-SYS5). Failure:		

	1. Status remains OPEN. 2. Error message displayed.
Main Flow	1. Review Grades: - Lecturer reviews the calculated Total Scores and ensures all data is correct. 2. Initiate Lock: - Lecturer clicks the "Finalize & Lock" button. 3. System Confirmation: - System displays a warning modal: *"Are you sure you want to lock this grade sheet? This action cannot be undone. Grades will be published to students immediately."* 4. Confirmation: - Lecturer selects "Confirm". - System calls API POST /api/grades/lock/{class_id}. 5. Validation & Processing: - Grade Service checks if all required grades are filled (optional, depending on policy). - Updates status to LOCKED in the Database. - Triggers an asynchronous event: GRADE_SHEET_LOCKED. 6. Feedback: - System notifies: *"Grade sheet locked successfully."* - The input fields become read-only. 7. Use Case Ends.
Alternative Flow	Step 5a. Missing Grades (Validation Error): → System detects that some students have missing component scores (Null) without a specific exemption. → System denies the Lock action. → Notification: *"Cannot lock grade sheet. Please enter grades for all students or mark them as Exempt."* Step 5b. Already Locked: → The grade sheet was already locked by an Admin. → System refreshes the page to Read-only mode.
Exceptions	1. Trigger Failure: → The Lock is successful, but the trigger for GPA Calculation (UC-SYS4) fails due to Message Queue error. → Interface still shows "Locked" to the Lecturer.

Requirements	1. Irreversibility: Once locked, the Lecturer cannot unlock it themselves. Unlocking requires an Administrator (UC-A-Unlock). 2. Audit Log: Must record exactly *when* and *who* locked the transcript to resolve disputes

- ❖ Specification and Construction of Student Use Case Group

Overview Description

The Student Module is the primary information portal for learners, serving as the interface where students interact with the university to perform credit-based training processes.

Unlike the Lecturer module (which focuses on data entry), the Student module's primary activities are Information Lookup (Viewing Grades, Viewing Schedules) and executing Registration Transactions (Registering/Canceling courses). The goal of this module is to provide transparent, accurate information and ensure fairness in credit registration.

Use Case Specification

Only representative use cases are selected for each actor to highlight their primary interactions with the system.

+ UC-S1 Student Login

Use Case Name	Student Login		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Students to authenticate their account credentials (Student ID/Password) to access learner-specific functions such as: Viewing Grades, Course Registration, and Viewing Class Schedules.		
Actors	- Student - Identity Service		
Pre-conditions	1. The Student has an existing account in the system (usually granted upon enrollment). 2. The Identity Service is operational.		
Post-conditions	Success: 1. Student receives an Authentication Token. 2. System redirects to the Student Dashboard. Failure: 1. User remains on the login page. 2. Error message is displayed.		
Main Flow	1. Access: - Student accesses the Academic Management Website. 2. Input Information: - Enters Username (Usually Student ID, e.g., SV001) and Password.		

	- Clicks the "Login" button. 3. Authentication: - System calls API POST /api/login to Identity Service. - Service verifies credentials against the Database. 4. Role Check: - System confirms the account is valid. - System verifies the account Role is "Student". - Returns Token and User info. 5. Redirect: - Interface redirects the student to the "Personal Info & Academic Results" screen (or Main Dashboard). 6. Use Case Ends.
Alternative Flow	Step 3a. Invalid Credentials: → Wrong Student ID or Password entered. → System notifies: *"Incorrect username or password."* Step 4a. Account Locked/Reserved: → Student is currently under disciplinary action or academic suspension. → System notifies: *"Account is temporarily locked. Please contact the Student Affairs Office."*
Exceptions	1. Connection Error: → Cannot connect to the server. → Notification: *"System error, please try again later."*
Requirements	1. Login interface must be simple and clear. 2. Must have a "Forgot Password" feature (reset password via student email). 3. Information security (Password hashing).

+ UC-S2 Register / Cancel Courses

Use Case Name	Register / Cancel Courses		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Students to view the list of open classes for the current semester, register for new courses, or cancel existing registrations within the allowed timeframe.		
Actors	- Student (Primary Actor) - Enrollment Service (Main Handler) - Course Service (Check Slots) - Notification Service (Confirmations)		
Pre-conditions	1. Student has logged in successfully. 2. The Course Registration Period is currently active (Open). 3. Student has paid tuition fees (if required by policy).		

Post-conditions	Success: 1. Register: New record added to enrollments, Class current_slots increases by 1. 2. Cancel: Record removed from enrollments, Class current_slots decreases by 1. 3. Student receives a confirmation email. Failure: 1. Registration rejected (Class full, Schedule conflict). 2. Error message displayed.
Main Flow	1. View Open Classes: - Student selects "Course Registration". - System displays list of available classes for the semester (showing Subject Name, Lecturer, Schedule, Current/Max Slots). 2. Select Class: - Student selects a class to register - Clicks "Register". 3. Validation (Enrollment Service): - Check Slots: Is current_slots < max_slots? - Check Schedule: Does the class time overlap with already registered courses? - Check Prerequisite: Has the student passed the required prerequisite subjects? - Check Credit Limit: Does total credits exceed the max allowed (e.g., 24 credits)? 4. Process Registration: - If all checks pass, System creates a "Pending" transaction. - Atomic Update: System increments the slot count in Course Service. - Saves registration record in DB. 5. Feedback: - System notifies: *"Registration Successful: [Subject Name]"*. - The class appears in the "Registered Courses" table. - Notification Service sends a confirmation email. 6. Use Case Ends.
Alternative Flow	Step 3a. Class Full: → Stud System detects current_slots >= max_slots. → Notification: *"Class is full. Please choose another class."* Step 3b. Schedule Conflict: → The selected class overlaps with an existing one. → Notification: *"Schedule conflict with [Existing Subject]."*

	Step 1a. Cancel Registration: → (Refer to UC-S5 details if separated, or here as a sub-flow) → Student clicks "Cancel" on a registered course. → System validates deadline → Removes record → Decrements slot count.
Exceptions	1. Race Condition (Concurrency): → Student A and B click "Register" at the exact same moment for the last slot. → Database applies Row Locking or Optimistic Locking. → One student succeeds, the other receives: *"Registration failed. The class just became full."* 2. System Overload: → Too many requests (> 10,000 req/s). → System puts the request in a Queue or returns 503 Service Unavailable.
Requirements	1. Data Integrity: Slot counting must be strictly accurate (ACID properties) to prevent over-subscription. 2. Performance: The "Check Slot" query must be highly optimized. 3. User Experience: The interface should update slot numbers in near real-time (via WebSocket if possible).

+ UC-S3 View Academic Results & GPA

Use Case Name	View Academic Results & GPA		
Created By	Development Team	Created By	Development Team
Created Date	Jan 1, 2026	Created Date	Jan 1, 2026
Description	Allows Students to view their complete academic history, including detailed grades for each course (Component scores, Final scores) and system-calculated performance metrics (Semester GPA, Cumulative CPA, Total Credits).		
Actors	- Student (Primary Actor) - Grade Service (System)		
Pre-conditions	1. Student has successfully logged in. 2. Academic data (Grades) exists in the system. 3. Grade Service is operational.		
Post-conditions	Success: 1. Academic transcript is displayed correctly. 2. GPA/CPA indicators match the latest calculation.		

	Failure: 1. "No data available" message is displayed. 2. Error notification (if service fails).
Main Flow	1. Access Academic Profile: - Student selects "Academic Results" from the dashboard. 2. Load Data: - System calls API GET /api/student/academic-profile to Grade Service. - Grade Service aggregates data: + *Summary:* Total Credits, GPA (Semester), CPA (Cumulative), Academic Status (Normal/Warning). + *Details:* List of courses grouped by Semester. 3. Display Overview (Dashboard): - System displays Key Performance Indicators (KPIs): + CPA: e.g., 3.24/4.0 + Total Credits: e.g., 85/150 + Status: "Good" 4. Display Detailed Transcript: - System displays a table of subjects. - Columns: Subject Name, Credits, Component Scores (Att/Mid/Final), Letter Grade (A, B, C...). 5. Filter (Optional): - Student filters by "Semester 1 - 2025". - System updates the view to show only that semester's grades and GPA. 6. Use Case Ends.
Alternative Flow	Step 2a. Financial Hold (Tuition Debt): → System checks with Tuition Service (if integrated) and finds unpaid fees. → System blocks the view of final grades. → Notification: *"Please complete tuition payment to view final results."* (Only Component scores might be visible). Step 2b. No Data (Freshman): → Student has not completed any courses yet. → System displays: *"No academic records found."* with GPA = 0.0.
Exceptions	1. Calculation Error: → Data corruption causes a division by zero in GPA calculation.

	→ System handles the exception gracefully, displaying "N/A" instead of crashing. 2. Display Error: → Subject names are missing due to sync issue with Course Service. → System displays "Unknown Subject [ID]" temporarily.
Requirements	1. Privacy: Students can strictly ONLY view their own grades. 2. Visualization: It is recommended to include a Line Chart showing GPA trends over semesters for better user experience. 3. Accuracy: The displayed GPA must be the result of the latest UC-SYS4 calculation.

5. Conclusion & Reflection

5.1 Lessons Learned

Through the implementation of the *Academic Management System based on Microservices Architecture*, the project team has gained valuable practical experience that goes beyond theoretical knowledge covered in class.

Firstly, the project helped the team clearly understand the differences between Monolithic and Microservices architectures. While Microservices offer significant advantages in scalability, independent deployment, and fault isolation, they also introduce complexity in service coordination, data consistency, and inter-service communication. Designing clear service boundaries (Identity, Student, Course, Grade, Enrollment, etc.) proved to be a critical architectural decision that directly impacted system maintainability and extensibility.

Secondly, the team gained hands-on experience with distributed system challenges, particularly in ensuring data consistency and fault tolerance. Scenarios such as course registration under high concurrency required careful handling to avoid issues like over-enrollment. This reinforced the importance of architectural patterns such as Database-per-Service, asynchronous processing, and circuit breaker mechanisms to prevent cascading failures when a service becomes unavailable.

Thirdly, the project highlighted the importance of non-functional requirements (NFRs) in architectural design. Performance, security, and reliability were not treated as afterthoughts but were explicitly translated into concrete design decisions, such as JWT-based authentication, role-based access control (RBAC), background email processing, and response-time constraints. This approach helped the team appreciate how quality attributes drive architecture selection and implementation strategies.

Finally, teamwork and collaboration were also key lessons learned. Dividing responsibilities among team members while maintaining consistent API standards and coding conventions required effective communication and documentation. The experience emphasized the value of clear interface contracts, proper logging, and structured testing to support teamwork in real-world software development.

5.2 Future Improvements

Although the system has achieved its primary objectives, several areas can be improved and extended in future iterations.

From an architectural perspective, the system could benefit from introducing a message broker (such as RabbitMQ or Kafka) to replace basic threading for asynchronous tasks. This would improve reliability, scalability, and observability for background processes like email notifications and large data imports.

In terms of scalability and deployment, containerization using Docker and orchestration with Kubernetes would allow services to scale independently based on workload, especially during peak course registration periods. Additionally, implementing centralized monitoring and logging (e.g., Prometheus, Grafana, ELK stack) would enhance system observability and simplify maintenance in production environments.

Regarding security, further enhancements could include token refresh mechanisms, rate limiting at the API Gateway, and more advanced auditing features to track sensitive operations such as grade unlocking or Administrator data changes.

From a functional standpoint, the system can be extended to support additional features such as online payment integration for tuition fees, advanced academic analytics dashboards, and integration with existing university systems (e.g., Learning Management Systems – LMS).

In conclusion, this project serves as a solid foundation and a practical case study for applying Microservices Architecture to a real-world academic management problem. The lessons learned and proposed improvements provide a clear roadmap for future development and deeper exploration of modern distributed software architectures.