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, February 6, 2026

TASK ASSIGNMENT TABLE

No.	Student ID	Full Name	Assigned Tasks
2	23012358	Quach Huu Nam	Part 2 – Requirements Analysis Key Quality Attributes Use Case Modeling: Student Use Cases Part 3 – Architecture Design C4 Container Diagram C4 Code-Level Design Technical Stack and Data Model Enrollment Service – Core Coordination Logic Part 4 – Testing Functional Testing

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

The project focuses on the design and implementation of an “Academic Management System” aimed at supporting the credit-based training model. The system addresses the management requirements for university, faculty, major, course, and student information; facilitates students in course registration and tracking academic results. Simultaneously, it allows lecturers to enter and manage grades, ensuring transparency and accuracy in the academic assessment process.

Throughout the development process, the system has successfully transitioned from a Monolithic architecture to a Microservices Architecture.

The final system comprises:

+ Independent Microservices: Identity, Student, Course, Grade, Enrollment, Email, Faculty, University, and Major Service (including KKT Service).
+ Modern Frontend: Built using Vanilla JavaScript (Single Page Application), enabling smooth interaction for both Students and Lecturers without page reloads.
+ Centralized Database with Logical Separation: The system currently uses a shared MySQL database schema, where each microservice accesses only the tables relevant to its own business domain. This design ensures clear logical separation of data ownership while simplifying deployment and development. The architecture is designed to be transition-ready toward a full Database-per-Service model in future iterations.
+ Hybrid Communication: Combines REST API (Synchronous) and Threading (Asynchronous).

The achieved result is a system with high availability, capable of handling large traffic volumes during peak course registration periods.

2. Project Requirements & Goals

2.2 Key Quality Attributes (Architectural Goals)

❖ Performance & Scalability

The system must respond to standard operations (e.g., viewing grades, logging in, data lookup) within ≤ 3 seconds under normal load conditions.

The system must support at least 500 concurrent users without significant performance degradation.

The system architecture must allow for horizontal scaling (scale-out) as the user base grows.

❖ Security

The system must require users to authenticate via username and password before granting access.

Role-Based Access Control (RBAC) must be strictly enforced between roles: Admin, Lecturer, and Student.

User passwords must be encrypted (hashed) before storage in the database.

The system must prevent unauthorized access and invalid data modification attempts.

❖ Reliability & Availability

The system must ensure 24/7 stability, excluding planned maintenance windows.
Data must be backed up periodically to prevent loss in the event of a failure.
In case of system errors, a mechanism for logging and notification must be in place to facilitate rapid troubleshooting.

❖ Maintainability & Extensibility

The system must be designed using a modular/Microservices architecture to facilitate maintenance and upgrades.
Source code must adhere to coding standards, ensuring readability and ease of modification.
The system must allow for the addition of new features (e.g., email sending, report generation) without significantly impacting existing functionality.

❖ Usability

The user interface (UI) must be intuitive and user-friendly, suitable for non-technical users.
Key functions must be easily accessible and accompanied by clear instructions.

2.4 Use Case Modeling: Use case Student

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

+ 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: → Student 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.

3. Architectural Design & Implementation

3.3 Architectural Views – C4 Model Representation

3.3.1 C4 – Container Diagram

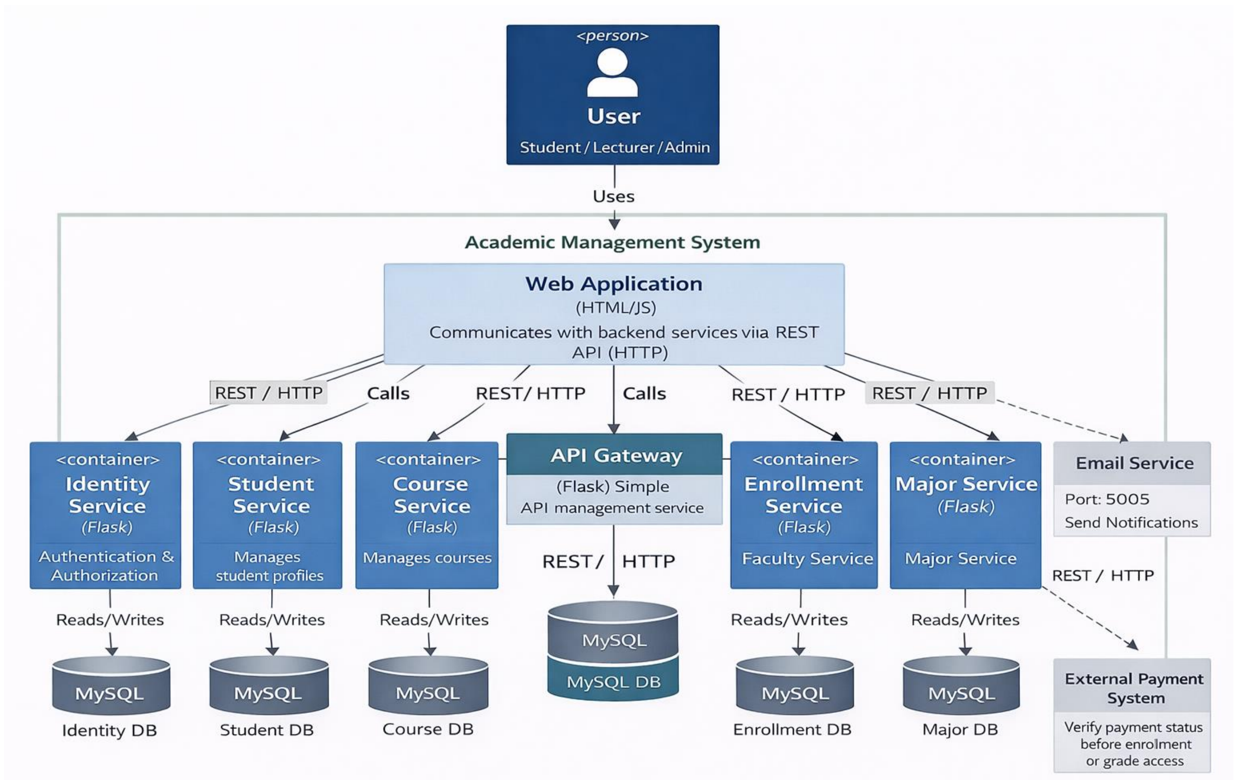
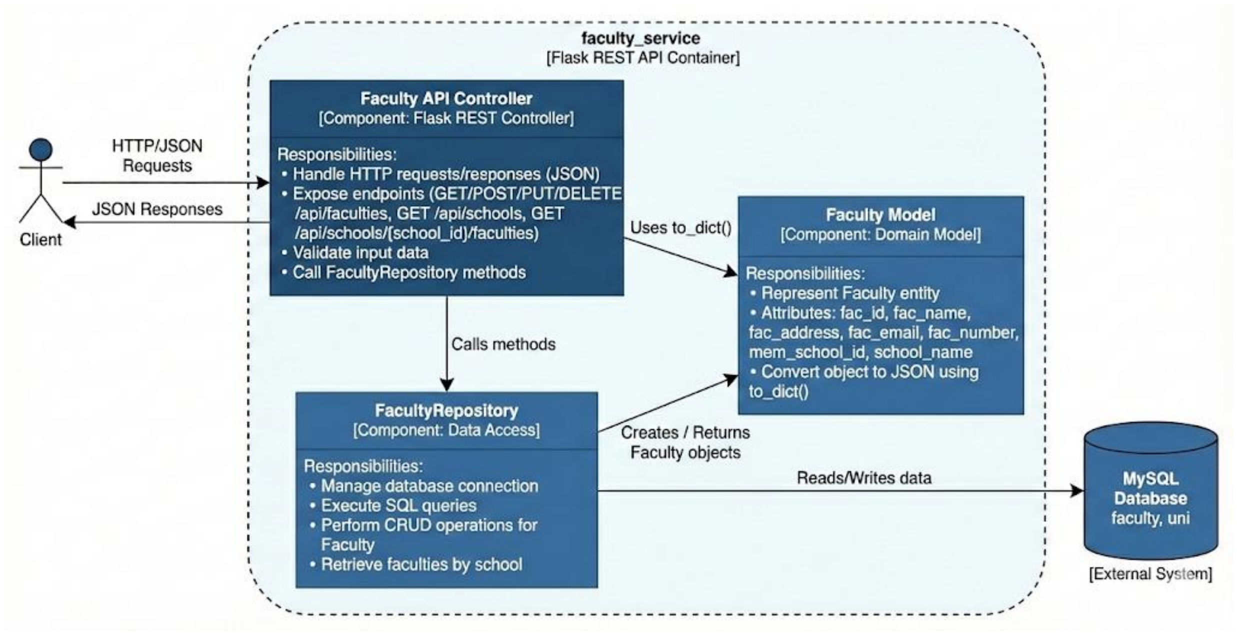


Figure presents the C4 – Container Diagram, showing the Web Application, the set of backend microservices, and the shared MySQL database.

3.3.3 C4 – Code Level



faculty_service
[Flask REST API Container]
Client
HTTP/JSON Requests
JSON Responses
Faculty API Controller
[Component: Flask REST Controller]
Responsibilities:
• Handle HTTP requests/responses (JSON)
• Expose endpoints (GET/POST/PUT/DELETE /api/faculties, GET /api/schools, GET /api/schools/{school_id}/faculties)
• Validate input data
• Call FacultyRepository methods
Uses to_dict()
Calls methods
Faculty Model
[Component: Domain Model]
Responsibilities:
• Represent Faculty entity
• Attributes: fac_id, fac_name, fac_address, fac_email, fac_number, mem_school_id, school_name
• Convert object to JSON using to_dict()
FacultyRepository
[Component: Data Access]
Responsibilities:
• Manage database connection
• Execute SQL queries
• Perform CRUD operations for Faculty
• Retrieve faculties by school
Creates / Returns Faculty objects
Reads/Writes data
MySQL Database
faculty, uni
[External System]

The figure illustrates the C4 Code Level for the Faculty Services

3.4 Technical Stack and Data Model

❖ Technical Stack

The Academic Management System is implemented using the following technologies:

Frontend:

+ Vanilla JavaScript (ES6+) for client-side logic and dynamic DOM manipulation, providing a lightweight Single Page Application (SPA) experience.

+ HTML5 and CSS3 for structure and presentation.

+ Fetch API for asynchronous RESTful communication with backend services.

Backend:

+ Python Flask is used to implement all microservices.

+ Each service exposes RESTful APIs using JSON over HTTP and runs independently on a dedicated port.

Database:

+ MySQL is used as the relational database management system.

+ The system currently employs a centralized database schema (sa), which is physically shared across services.

Communication:

+ Synchronous HTTP communication is used between the Frontend and backend Microservices.

+ Certain non-critical operations, such as email notifications, are handled asynchronously using background threads. In this case, the Grade Service initiates a non-blocking HTTP request to the Email Service to avoid delaying user-facing operations.

❖ Data Model

All microservices interact with a centralized MySQL database schema (sa). Although the database is physically shared, data ownership is logically separated by service boundaries. Each microservice accesses and manipulates only the tables related to its own business domain.

The core domain entities include:

+ User *(Identity Service)*: (id, username, password, role)

+ Student *(Student Service)*: (student_id, name, date_of_birth, class_name, email)

+ Course *(Course Service)*: (course_id, course_name, credits)

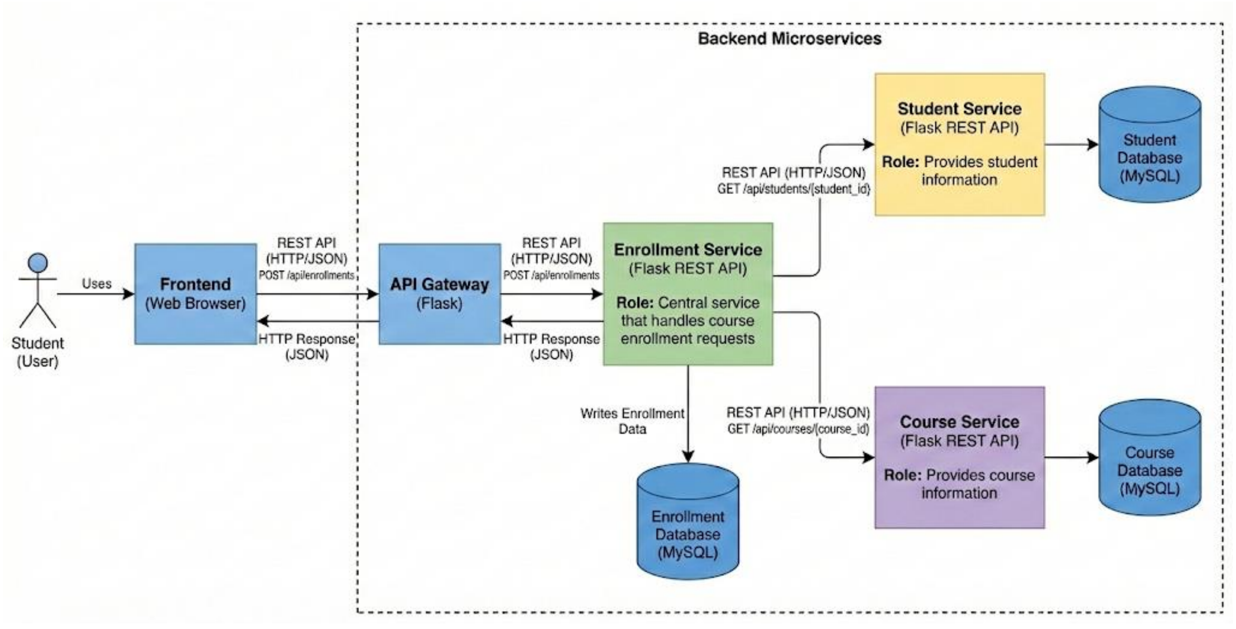
+ Enrollment *(Enrollment Service)*: (enrollment_id, student_id, course_id, semester, status)

+ Grade *(Grade Service)*: (grade_id, enrollment_id, attendance_score, midterm_score, final_score, total_score, letter_grade)

This design enforces clear logical data ownership while simplifying development and deployment. The architecture is intentionally designed to be evolution-ready, allowing a future transition toward a full Database-per-Service model when system scale and operational requirements increase.

3.5 Implementation Design

3.5.3 Enrollment Service – Core Coordination Logic


Backend Microservices
Student (User)
Uses
Frontend (Web Browser)
REST API (HTTP/JSON) POST /api/enrollments
HTTP Response (JSON)
API Gateway (Flask)
REST API (HTTP/JSON) POST /api/enrollments
HTTP Response (JSON)
Enrollment Service (Flask REST API)
Role: Central service that handles course enrollment requests
REST API (HTTP/JSON) GET /api/students/{student_id}
Student Service (Flask REST API)
Role: Provides student information
Student Database (MySQL)
Writes Enrollment Data
Enrollment Database (MySQL)
REST API (HTTP/JSON) GET /api/courses/{course_id}
Course Service (Flask REST API)
Role: Provides course information
Course Database (MySQL)

The figure illustrates the detailed architecture for the Student Course Enrollment function.

4. Testing & Verification

4.4 Functional Testing

Functional Testing is conducted to verify that each Microservice correctly implements its assigned business functions according to the system requirements.

❖ Test Cases – Identity Service

The Identity Service is responsible for authenticating users and enforcing role-based access control using JWT tokens.

Test Case ID	Description	Input / Action	Expected Result
TC_ID_01	Login with valid credentials	Submit correct username & password	JWT token returned
TC_ID_02	Login with invalid credentials	Submit wrong password	HTTP 401 Unauthorized
TC_ID_03	Access protected API without token	Call API without JWT	HTTP 401 Unauthorized
TC_ID_04	Access API with insufficient	Student accesses admin API	HTTP 403 Forbidden

	role		

❖ Test Cases – Student Service

The Student Service manages student profile information, including creation, update, and retrieval.

Test Case ID	Description	Action	Expected Result
TC_STU_01	Create new student	POST /api/students	Student created successfully
TC_STU_02	Get student profile	GET /api/students/{id}	Correct student data returned
TC_STU_03	Update student profile	PUT /api/students/{id}	Student data updated
TC_STU_04	Invalid student data	Missing required fields	HTTP 400 Bad Request

❖ Test Cases – Course Service

The Course Service manages course information such as course name, credits, and semester availability.

Test Case ID	Description	Action	Expected Result
TC_CRS_01	View course list	GET /api/courses	Course list displayed
TC_CRS_02	Add new course	POST /api/courses	Course created
TC_CRS_03	Update course info	PUT /api/courses/{id}	Course updated
TC_CRS_04	Delete course	DELETE /api/courses/{id}	Course removed

❖ Test Cases– Course Registration

The Course Registration function allows students to register for courses during a semester.

The system requires confirmation before completing registration and provides feedback upon success.

Test Case ID	Feature Name	Test Description	Prerequisites	Input Data / Actions	Expected Result	Actual Result	Status
TC_ENR_01	Confirm Course	Student selects a	Student is logged in.	Click the “Register”	A registration	The “Confirm	Pass

	Registration	course to register.		button on the course *Basic English 2*.	confirmation popup appears.	Registration” popup is displayed with correct content.	
TC_ENR_02	Cancel Registration	Student cancels the registration action.	Confirmation popup is displayed.	Click the “Cancel” button.	The popup closes; the course is not registered.	The course was not registered.	Pass
TC_ENR_03	Confirm Registration	Student confirms the course registration.	Confirmation popup is displayed.	Click the “Confirm” button.	The system processes the course registration.	The registration was processed successfully.	Pass
TC_ENR_04	Successful Registration Notification	Check the notification after registration.	Registration is successful.	Observe the system.	Display message: “Registered for course ... successfully”.	The notification is displayed correctly.	Pass
TC_ENR_05	Update Registered Courses List	Check the list after registration.	Registration is successful.	Access the “Course Registration” page.	The course appears in the list of registered courses.	The list is updated correctly.	Pass
TC_ENR_06	Update Class Count & Credits	Check summary information.	Course has been registered.	Observe summary information.	The number of classes and total credits	Displayed: “Registered Classes: 6 – Total	Pass

					increase accordingly .	Credits: 25”.	

❖ Test Cases – Grade Service

The Grade Service manages grade entry, grade locking, GPA calculation, and triggers notification events.

Test Case ID	Description	Action	Expected Result
TC_GRD_01	Enter student grades	POST /api/grades	Grades saved
TC_GRD_02	Calculate GPA	Save valid scores	GPA calculated correctly
TC_GRD_03	Lock grade sheet	Click “Save & Lock”	Grades locked
TC_GRD_04	Update locked grades	Modify locked grades	Operation rejected
TC_GRD_05	Trigger email event	Lock grades	Async event published

❖ Test Cases – Email Service

This service processes asynchronous events and sends student-related notifications.

Test Case ID	Description	Action	Expected Result
TC_EML_01	Receive grade event	Consume event from queue	Event processed
TC_EML_02	Send email log	Grade event received	Email log displayed
TC_EML_03	Invalid event format	Malformed message	Error logged, system continues

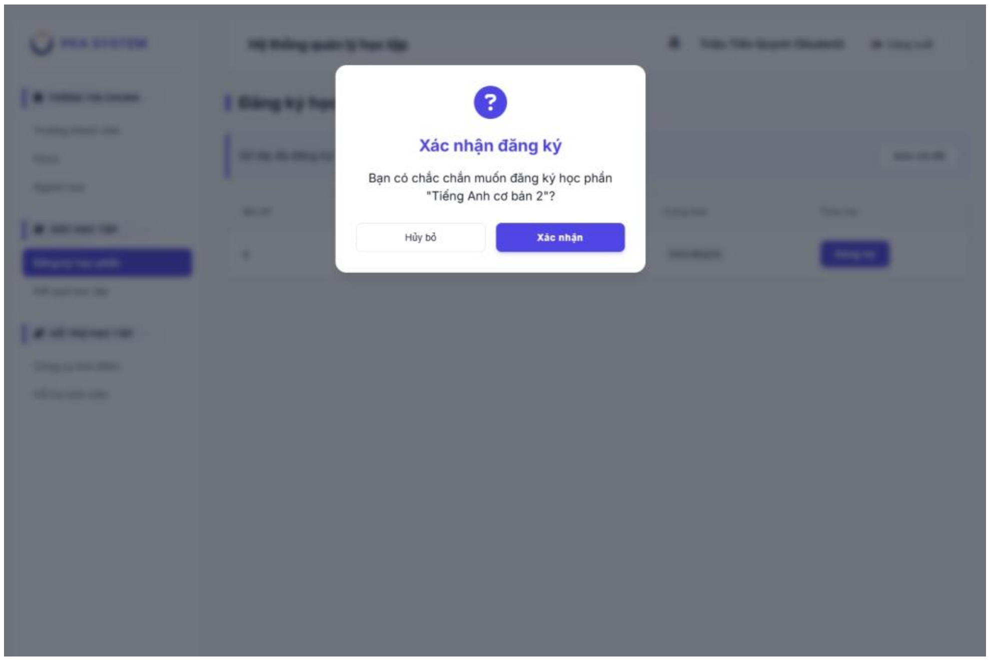
Test Result Comments

The Course Registration function fully satisfies business requirements.

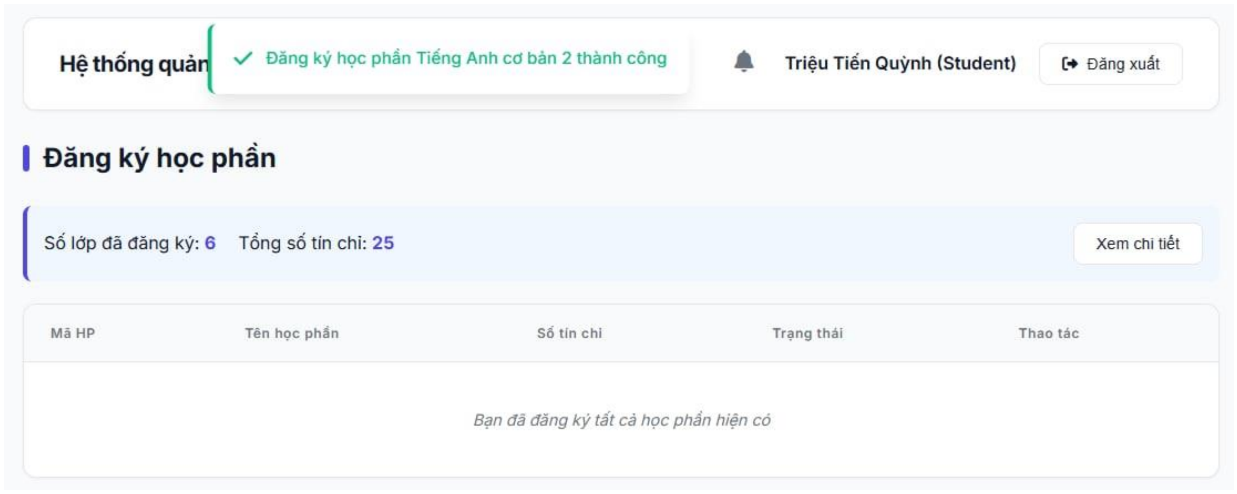
Confirmation popup prevents accidental registration.

UI feedback and data synchronization are accurate.

The feature operates reliably within the microservices architecture.


Xác nhận đăng ký
Bạn có chắc chắn muốn đăng ký học phần "Tiếng Anh cơ bản 2"?
Hủy bỏ
Xác nhận

The figure illustrates Course Registration Interface (Student View)



Mã HP	Tên học phần	Số tín chỉ	Trạng thái	Thao tác
Bạn đã đăng ký tất cả học phần hiện có				

The figure illustrates "Registration Successful" notification on web interface