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
2	23012358	Quach Huu Nam	Key Quality Attributes (Architectural Goals) Testing, Verification & Deployment

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 report presents the development of an Academic Management System based on Microservices Architecture. The report first introduces the overall project context and defines key quality attributes, including performance, scalability, security, reliability, maintainability, and usability. It then describes the testing and verification strategy, covering unit testing, integration testing, and end-to-end test scenarios to validate system behavior across multiple services. Finally, the report presents deployment configurations, service port mapping, and practical implementation results demonstrated through real execution scenarios.

2. Key Quality Attributes (Architectural Goals)

- ❖ Performance & Scalability

Particularly critical for UC4 (Course Registration) due to anticipated traffic spikes.

NFR-01 (Response Time): The system must respond to data read requests (GET) within < 2 seconds (under stable network conditions). For write transactions (such as Grade Submission, Course Registration), processing time must not exceed 3 seconds.

NFR-02 (Concurrency / Load Capacity): The system must support a minimum of 500 - 1,000 Concurrent Users (CCU) during peak registration periods without crashing or service interruption.).

NFR-03 (API Gateway Latency): The API Gateway must not introduce a latency exceeding 50ms when routing requests from the Client to backend Microservices.

NFR-04 (Asynchronous Processing): Time-consuming tasks (such as sending Email notifications, importing large Excel files) must be handled as Background Tasks (Asynchronous) to prevent blocking the user interface.

- ❖ Security

Crucial as the system stores personal information and academic records.

NFR-05 (Authentication & Authorization): All APIs accessing resources (except for login/public pages) must be protected via JWT (JSON Web Token) mechanisms. The API Gateway must reject requests without a valid token (returning HTTP 401).

NFR-06 (Role-Based Access Control - RBAC): The system must strictly enforce authorization based on roles:

- o Student: Can only view their own grades and register for themselves.
- o Lecturer: Can only enter grades for classes they teach.
- o Administratoristrator: Has full Administratoristratoristrative privileges.

NFR-07 (Data in Transit Security): All communication between the Client and Server (API Gateway) must be encrypted via HTTPS protocol (TLS 1.2 or higher).

NFR-08 (Password Hashing): User passwords in the Identity Service must not be stored in plain text but must be hashed using strong algorithms (e.g., BCrypt or Argon2).

- ❖ Reliability & Availability

Ensures stable system operation in a distributed environment.

NFR-09 (Availability): The system guarantees an Uptime of 99% during business hours.

NFR-10 (Fault Tolerance): If a child Service (e.g., Email Service) fails, it must not cause the entire system to crash. Core functions (such as Viewing Grades) must continue to operate normally. (Apply Circuit Breaker Pattern).

NFR-11 (Data Consistency): In the Course Registration workflow, the system must ensure consistency between Slot Availability (Course Service) and the Class List (Enrollment Service). The scenario where a class is full (Slot = 0) but a student still successfully registers must not occur.

- ❖ Maintainability & Extensibility

Serving the Development Team.

NFR-12 (Microservices Architecture): Services (Student, Grade, Course, etc.) must be independently deployable. Updating the code of the Grade Service must not require restarting the Student Service.

NFR-13 (API Standard): APIs must adhere to RESTful standards, correctly utilizing HTTP Methods (GET, POST, PUT, DELETE) and Status Codes (200, 201, 400, 404, 500).

NFR-14 (Code Quality): Python source code must comply with PEP 8 standards, with full comments for complex business logic functions (e.g., GPA calculation, registration processing).

NFR-15 (Logging): The system must have a centralized Error Logging mechanism to facilitate Debugging when incidents occur.

- ❖ Usability

Serving the End-User Experience (Students/Lecturers).

NFR-16 (Responsive Design): The web interface must display correctly on both Desktop and Mobile/Tablet devices, especially for Grade Viewing and Course Registration functions.

NFR-17 (User-Friendly Error Messages): When a system error (Backend Error) occurs, the interface must display an understandable message to the user (e.g., *"System is busy, please try again later"*) instead of showing technical error codes or Stack Traces.

NFR-18 (Minimal Interaction/Efficiency): Lecturers can enter and save grades for an entire class with a single button (Bulk Save), without being forced to save line by line.

4. Testing & Verification

4.1 Testing Strategy (Unit/Integration Test)

Due to the specific nature of the distributed Microservices architecture, the team applied a multi-layer testing strategy to ensure each service functions correctly before system-wide integration.

4.1.1 Unit Testing (Kiểm thử Đơn vị)

This focuses on testing the smallest components within each Microservice, ensuring internal data processing logic is accurate.

Model Testing: Tests Entity classes (such as Student, Course, Grade) in models.py to ensure data is correctly initialized and converted to JSON (to_dict).

Repository Testing: Tests data access methods in repository.py. This ensures SQL statements (INSERT, SELECT, UPDATE) function correctly with the MySQL database and handle Exceptions (e.g., connection failures) properly.

4.1.2 Integration Testing

Tests the interaction between components within a Service and between Services via API.

API Testing: Uses tools like Postman and cURL to send HTTP Requests (GET, POST, PUT, DELETE) to each Endpoint.

- *Objective:* Verify HTTP Status Codes (200, 201, 400, 404, 500) and the returned JSON structure.
- *Example:* Sending a POST /api/grades request missing required data to check if the server correctly returns a 400 Bad Request error.

Inter-service Communication: Tests the asynchronous communication flow between the Grade Service and Email Service.

- *Scenario:* When a Lecturer finishes entering grades, verify if the Email Service receives the signal and sends the email successfully (by observing the logs of both services).

4.2 Deployment Configuration (Ports & Env)

The system is designed for easy deployment on a local environment (Localhost) with clear port configurations and environment variables.

4.2.1 Environment Variables

To ensure security and facilitate configuration changes without modifying the code, the system uses a .env file to manage Database connection information. All repository.py files utilize the python-dotenv library to read these variables.

File cấu hình (.env):

DB_HOST=localhost

DB_USER=root

DB_PASSWORD=your_password

DB_NAME=sa

4.2.2 Port Mapping

Each Microservice is configured to run on a distinct port to avoid conflicts. Below is the port configuration table extracted from the app.py files:

Service Name	Port	Main Function	Configuration File
Identity Service	5004	Authentication & Authorization	identity_service/app.py
Student Service	5001	Student Profile Management	student_service/app.py
Course Service	5002	Course Module Management	course_service/app.py
Grade Service	5003	Grade & GPA Management	grade_service/app.py
Email Service	5005	Email Notifications	email_service/app.py
Enrollment Service	5006	Credit Registration	enrollment_service/app.py
Faculty Service	5007	Faculty Management	faculty_service/app.py
University Service	5008	University Management	uni_service/app.py
Major Service	5009	Major Management	major_service/app.py
KKT Service	5010	Knowledge Block Management	kkt_service/app.py

4.3 End-to-End Test Scenarios

This section describes an End-to-End test scenario simulating a real-world business process from the Frontend interface (Vanilla JS) down to the Database.

Kịch bản: Quy trình Đăng ký học và Nhập điểm

Scenario: Course Registration and Grade Entry Process Objective: Verify data flow across multiple Services: Identity → Course → Enrollment → Grade → Email.

Step	Actor	System Action	Expected Result
1	Student	Logs into the system with a student account.	Receives an Authentication Token; redirected to the Student Dashboard.
2	Student	Accesses "Course Registration" and selects the course "Software Architecture".	System calls Course Service to retrieve info, calls Enrollment Service to save registration. Returns notification "Registration Successful".
3	Lecturer	Logs in and accesses "Enter Grades" for the "Software Architecture" class.	The student list (registered in Step 2) is fully displayed on the

			grade sheet.
4	Lecturer	Enters process scores, exam scores, and clicks "Save & Lock".	Grade Service saves scores to MySQL, calculates GPA, and triggers the email sending flow (Async).
5	System	(Automated) Grade Service calls Email Service in the background.	Email Service console log displays: *"Email sent to [Student Email]: Your grade has been updated."*
6	Student	Checks mailbox and returns to "View Grades" page.	Receives email notification and sees updated scores on the web interface.

Actual Results:

The test scenario was successfully executed in the Localhost environment. Login, course registration, and grade entry/locking functions operated according to design.

Regarding the email notification function, the system successfully triggered the asynchronous email flow from the Grade Service to the Email Service, and success was recorded via Email Service logs. Actual email delivery to student inboxes was not implemented within the scope of this project; therefore, results are confirmed at the system-level rather than the end-user level.

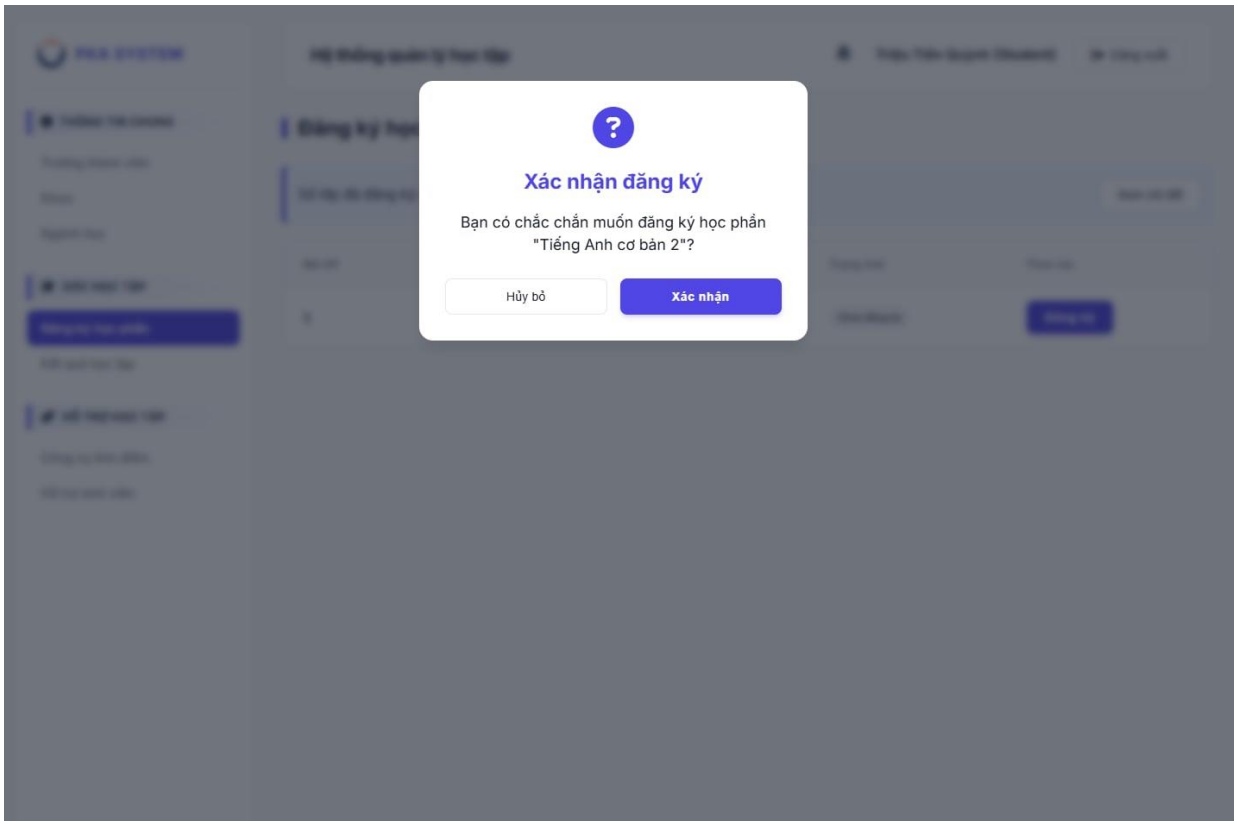

Figure 4.3.1 Course Registration Interface (Student View)

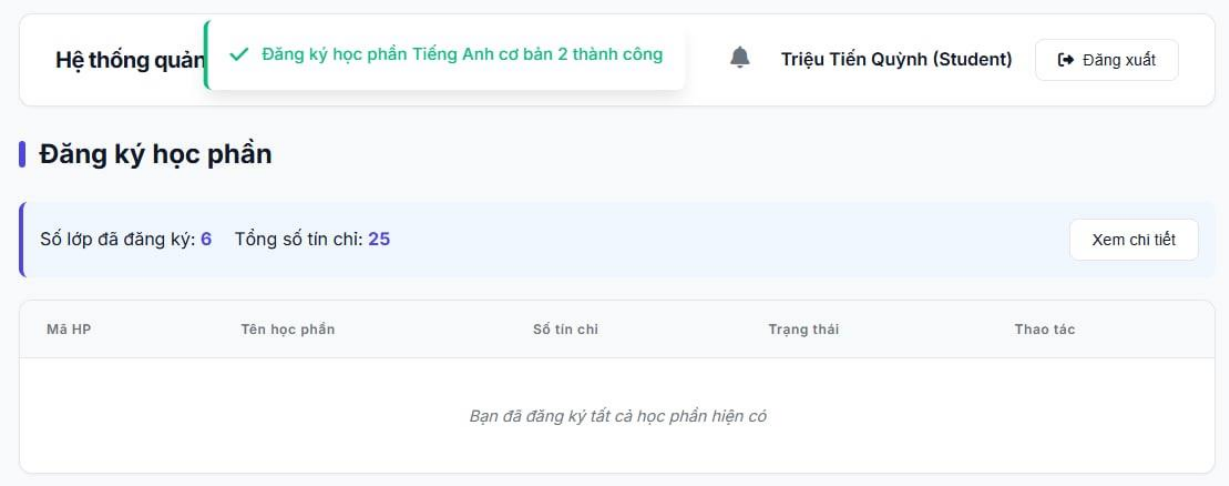

Figure 4.3.2 "Registration Successful" notification on web interface

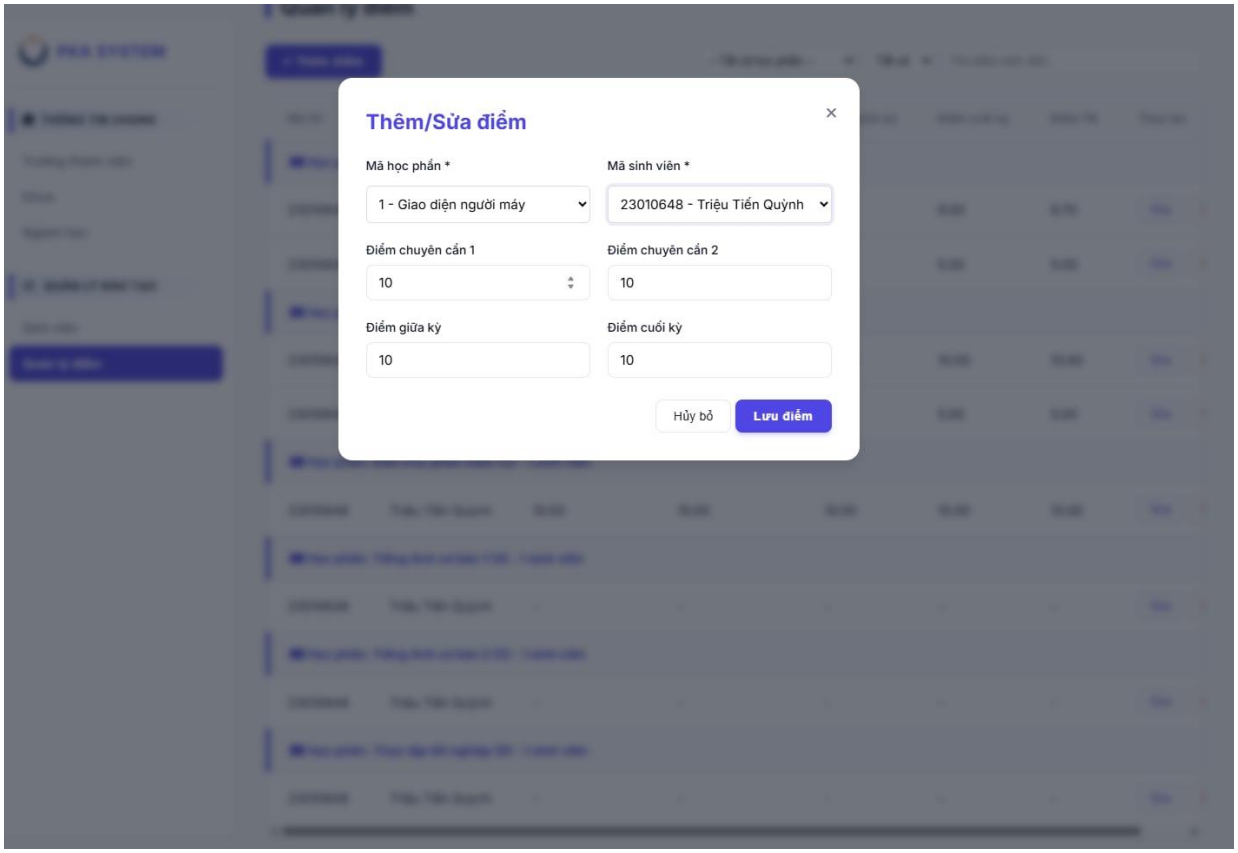

Figure 4.3.1 Lecturer's Grade Entry Interface (Lecturer View)