PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

Event-Driven Architecture – EDA

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 7, 2026

Lab 7 Event-Driven Architecture – EDA

1. Overview & Objectives

In previous Labs, inter-service communication (e.g., Grade Service calling Email Service) was implemented using synchronous HTTP Requests or simple Threading. This approach presents a significant drawback: if the receiving Service fails or becomes overloaded, the sending Service is negatively impacted, leading to system hangs or data loss.

Lab 7 Objectives:

Understand and implement the Producer-Consumer model via a Message Broker.

Utilize RabbitMQ as the message relay middleware.

Fully decouple the Grade Service (Producer) from the Email Service (Consumer).

Ensure that email delivery tasks are processed asynchronously and reliably.

2. Technology & Tool Installation

We use Python and the Pika library to interact with the RabbitMQ message broker.

Tool	Purpose	Installation/Setup Guide
RabbitMQ	The Message Broker that holds and routes events.	Recommended: Use Docker. Run: docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
Python 3.x	Core programming language.	Ensure Python 3 is installed.
Pika	Python client library for connecting to RabbitMQ.	Run: pip install pika
Terminals	To run multiple services concurrently.	Open at least two separate terminals in VS Code.

3. Activity Practice 1: Setup and Broker Connection

Goal: Ensure RabbitMQ is running and set up the basic connection configuration for both services.

1. Start RabbitMQ: Run the Docker command above. Verify the broker is running by opening the management console: http://localhost:15672 (Login: guest/guest).
2. Define Broker Parameters: In both services, we use the following configuration:

RABBITMQ_HOST = 'localhost'

QUEUE_NAME = 'grade_events'

4. Activity Practice 2: Grade Service (Event Producer)

Goal: Update the Grade Service to publish a GRADE_UPDATED event whenever a faculty member submits a grade

File : app.py (grade_service)

```python
# ==========================================================
# HÀM HỖ TRỢ: PHÁT SỰ KIỆN QUA RABBITMQ (EDA)
# ==========================================================
def publish_grade_event(student_id, grade, course_id):
    """
    Kết nối RabbitMQ và đẩy thông tin điểm vào hàng đợi.
    Đây là mô hình Producer trong EDA.
    """
    try:
        connection = pika.BlockingConnection(pika.ConnectionParameters(host=RABBITMQ_HOST))
        channel = connection.channel()
        channel.queue_declare(queue=QUEUE_NAME)

        message = json.dumps({
            "student_id": student_id,
            "grade": grade,
            "course_id": course_id,
            "event": "GRADE_UPDATED"
        })

        channel.basic_publish(
            exchange='',
            routing_key=QUEUE_NAME,
            body=message
        )
        print(f" [x] [RabbitMQ] Sent Grade Event for Student {student_id}")
        connection.close()
    except Exception as ex:
        print(f" [!] RabbitMQ Error: {ex}")
```

5. Activity Practice 3: Notification Service (Event Consumer)

Goal: Implement the Notification Service to listen for events, persist them to the MySQL database, and trigger external notifications (like Email).

File : app.py (notification_service)

```
def rabbitmq_consumer():
    """Lắng nghe sự kiện điểm số từ RabbitMQ và gọi Email Service"""
    try:
        connection = pika.BlockingConnection(pika.ConnectionParameters(host=RABBITMQ_HOST))
        channel = connection.channel()
        channel.queue_declare(queue=QUEUE_NAME)

        def callback(ch, method, properties, body):
            try:
                data = json.loads(body)
                student_id = data.get("student_id")
                grade = data.get("grade")
                course_id = data.get("course_id")

                print(f" [x] [Notification Service] Received Grade Update for {student_id}")

                # 1. Lưu thông báo vào Database để sinh viên xem trên Web
                title = "Cập nhật điểm mới"
                message = f"Bạn vừa có điểm mới cho học phần {course_id}. Kết quả: {grade}"
                try:
                    notif_id = repo.add_notification(student_id, title, message)
                    print(f" [v] Notification saved to DB with ID: {notif_id}")
                except Exception as e:
                    print(f" [!] Error saving notification to DB: {e}")

                # 2. Gọi sang Email Service để gửi thông báo (Nếu cần)
                payload = {
                    "student_id": student_id,
                    "grade": grade,
                    "course_id": course_id
                }

                try:
                    response = requests.post(EMAIL_SERVICE_URL, json=payload)
                    if response.status_code == 200:
                        print(f" [v] Email Service notified successfully")
                    else:
```

6. Activity Practice 4: Testing Asynchronous Decoupling

Goal: Observe how the Grade Service completes its task instantly while the Notification Service processes the event in the background.

1. Start the Consumer: Terminal 1 -> python src/microservices/notification_service/app.py.
2. Start the Producer: Terminal 2 -> python src/microservices/grade_service/app.py.
3. Trigger Event: Update a student's grade via the Web UI or Postman.
4. Observation:
    - Grade Service: The HTTP response is returned to the UI immediately after the database update.
    - Notification Service: You will see the "Received grade event" logs appear shortly after, demonstrating that the time-consuming notification tasks did not block the grading process.

7. Conclusion:

This architecture ensures high availability. Even if the Notification Service is temporarily down, the Grade Service continues to function, and messages stay queued in RabbitMQ until the consumer is back online