PHENIKAA UNIVERSITY
PHENIKAA SCHOOL OF COMPUTING

SOFTWARE ARCHITECTURE

Lab 7: Event-Driven Architecture (EDA) & Integration (QuickShip)

Instructor: Vũ Quang Dũng
Course: Software Architecture – Class N02
Team Members: Group 11
Nguyễn Trọng Thái – 23010457
Nguyễn Minh Quang – 23010419

Tháng 1 năm 2026

Lab 7: Event-Driven Architecture (EDA) & Integration (QuickShip)

This lab introduces the **Event-Driven Architecture (EDA)** pattern, focusing on asynchronous communication. We will use **RabbitMQ** as a message broker to decouple the **Shipment Service** (Producer) from the **Notification Service** (Consumer). In QuickShip, this ensures that creating a shipment isn't delayed by the time it takes to send an SMS or Email.

Objectives

1. Understand the roles of Producers, Consumers, and Message Brokers.
2. Set up a local **RabbitMQ** broker using Docker.
3. Implement a Shipment Service that acts as the event Producer.
4. Implement a Notification Service that acts as the event Consumer.
5. Demonstrate how services operate independently (decoupling).

Activity Practice 1: Setup and Broker Connection

Goal: Ensure RabbitMQ is running and set up the Python environment.

1. **Start RabbitMQ (via Docker):**

Bash

```
docker run -d --hostname quickship-rabbit --name quickship-rabbit -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

2. **Project Structure:**

Bash

```
mkdir quickship_eda
cd quickship_eda
python -m venv venv
source venv/bin/activate
pip install pika
touch shipment_producer.py
touch notification_consumer.py
```

Activity Practice 2: Shipment Service (Event Producer)

Goal: Simulate the Shipment Service publishing a ShipmentCreated event whenever a new delivery is registered.

File: shipment_producer.py

Python

```
import pika
import json
import time

RABBITMQ_HOST = 'localhost'
QUEUE_NAME = 'shipment_events'

def publish_shipment_created(shipment_data):
    """Connects to RabbitMQ and publishes the shipment event."""
    try:
        connection =
pika.BlockingConnection(pika.ConnectionParameters(host=RABBIT
MQ_HOST))
        channel = connection.channel()

        # Declare the queue
        channel.queue_declare(queue=QUEUE_NAME)

        message = json.dumps(shipment_data)

        channel.basic_publish(
            exchange='',
            routing_key=QUEUE_NAME,
            body=message
        )
        print(f" [x] Shipment Service: Published event for ID
{shipment_data['shipment_id']}")

        connection.close()
    except Exception as e:
        print(f" [!] Connection Error: {e}")

if __name__ == '__main__':
    print("Shipment Service (Producer) is starting...")
    # Simulate creating 3 shipments
    for i in range(1, 4):
        shipment_info = {
```

```
            "shipment_id": f"QS-77{i}",
            "customer_email": f"user{i}@quickship.com",
            "type": "Express",
            "timestamp": time.time()
        }
        publish_shipment_created(shipment_info)
        time.sleep(1)
```

Activity Practice 3: Notification Service (Event Consumer)

Goal: Implement a service that listens for shipment events and simulates sending email confirmations.

File: notification_consumer.py

Python

```
import pika
import json
import time

RABBITMQ_HOST = 'localhost'
QUEUE_NAME = 'shipment_events'

def callback(ch, method, properties, body):
    """Callback function triggered when a message is received."""
    shipment_data = json.loads(body)
    shipment_id = shipment_data.get('shipment_id')
    email = shipment_data.get('customer_email')

    print(f" [x] Notification Service: Received event for
{shipment_id}")

    # Simulate network latency in sending an email (2 seconds)
    time.sleep(2)

    print(f" [✓] ALERT SENT: Confirmation for {shipment_id}
delivered to {email}")

    # Acknowledge completion
    ch.basic_ack(delivery_tag=method.delivery_tag)

if __name__ == '__main__':
    try:
```

```
        connection =
pika.BlockingConnection(pika.ConnectionParameters(host=RABBIT
MQ_HOST))
        channel = connection.channel()

        channel.queue_declare(queue=QUEUE_NAME)
        channel.basic_qos(prefetch_count=1) # Don't give more than 1
message to a worker at a time
        channel.basic_consume(queue=QUEUE_NAME,
on_message_callback=callback)

        print(' [*] Waiting for ShipmentEvents. To exit press CTRL+C')
        channel.start_consuming()
    except Exception as e:
        print(f" [!] Error: {e}")
```

Activity Practice 4: Testing Asynchronous Decoupling

1. Open Terminal 1: Run the consumer:
```
python notification_consumer.py
```
2. Open Terminal 2: Run the producer:
```
python shipment_producer.py
```
3. **Observe the Behavior:**
    - The **Shipment Service** finishes publishing all 3 events almost instantly.
    - The **Notification Service** processes them slowly (one every 2 seconds).
    - **Conclusion:** The time-consuming task (sending emails) did not block the main business process (creating shipments). This is the power of **Asynchronous Integration**.