PHENIKAA UNIVERSITY
PHENIKAA SCHOOL OF COMPUTING

SOFTWARE ARCHITECTURE

Lab 5: Implementing the Product Microservice

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

Tháng 12 năm 2025

Lab 5: Implementing the Shipment Microservice (QuickShip)
This lab focuses on the practical implementation of the **Shipment Service**, one of the core services identified in Lab 4. This service will be completely independent, owning its data via a dedicated SQLite database and exposing a RESTful API.

Objectives

1. Set up a standalone Flask application dedicated to shipment management.
2. Implement the Shipment Service logic and persistence using **SQLAlchemy**.
3. Expose REST API endpoints for tracking and listing shipments.
4. Test the service in isolation.

Activity 1: Project Setup and Data Modeling

Goal: Create the project structure and define the Shipment database schema.

1. Create Service Directory:

Bash

```
# Ensure you are outside previous lab directories
mkdir shipment_service
cd shipment_service
python -m venv venv
source venv/bin/activate # On Windows: venv\Scripts\activate
pip install Flask Flask-SQLAlchemy
touch app.py
```

2. Initialize Flask and SQLAlchemy:

Open app.py and configure the SQLite database dedicated to this service.

File: app.py (Setup & Model)

Python

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
# Configure a dedicated database for this service (Encapsulation)
app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:///shipments.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

```
# 3. Define the Shipment Model
class Shipment(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    sender_name = db.Column(db.String(100), nullable=False)
    receiver_name = db.Column(db.String(100), nullable=False)
    origin_address = db.Column(db.String(200), nullable=False)
    destination_address = db.Column(db.String(200), nullable=False)
    weight = db.Column(db.Float, nullable=False)
    status = db.Column(db.String(20), default='Pending') # Pending, In
Transit, Delivered

    def to_dict(self):
        return {
            'id': self.id,
            'sender': self.sender_name,
            'receiver': self.receiver_name,
            'origin': self.origin_address,
            'destination': self.destination_address,
            'weight': self.weight,
            'status': self.status
        }
```

4. Create Database and Initial Data:

Run the following in your terminal to initialize the database:

Bash

```
python
>>> from app import app, db, Shipment
>>> with app.app_context():
...     db.create_all()
...     # Add sample data for QuickShip
...     db.session.add(Shipment(sender_name='Alice Smith',
receiver_name='Bob Jones',
...                 origin_address='New York', destination_address='Los
Angeles', weight=2.5))
...     db.session.add(Shipment(sender_name='Charlie Brown',
receiver_name='Diana Prince',
...                 origin_address='Chicago', destination_address='Miami',
weight=12.0))
...     db.session.commit()
...     print("QuickShip Database Initialized!")
>>> exit()
```

Activity 2: Implementing the Service API

Goal: Implement the REST API endpoints to read and search shipment data.

File: app.py (Add these routes):

Python

```
# Endpoint: List all shipments or filter by sender name
@app.route('/api/shipments', methods=['GET'])
def list_shipments():
    sender = request.args.get('sender')
    if sender:
        # Case-insensitive search for sender
        shipments =
Shipment.query.filter(Shipment.sender_name.like(f'%{sender}%')).all
()
    else:
        shipments = Shipment.query.all()

    return jsonify([s.to_dict() for s in shipments]), 200

# Endpoint: Retrieve a single shipment by Tracking ID
@app.route('/api/shipments/<int:shipment_id>', methods=['GET'])
def get_shipment(shipment_id):
    shipment = Shipment.query.get(shipment_id)
    if shipment:
        return jsonify(shipment.to_dict()), 200

    return jsonify({'error': 'Tracking ID not found'}), 404

# Run on port 5001 to avoid conflicts with other services
if __name__ == '__main__':
    app.run(port=5001, debug=True)
```

Activity 3: Isolation Testing

Goal: Verify that the service operates correctly and independently.

1. **Start the Service:**

Bash
python app.py
The service will be running at: http://127.0.0.1:5001

2. **Test Listing Shipments (GET):**
    - **Command:** curl -X GET http://127.0.0.1:5001/api/shipments
    - **Expected Result:** A JSON array containing Alice and Charlie's shipments.
3. **Test Shipment Lookup (GET):**
    - **Command:** curl -X GET http://127.0.0.1:5001/api/shipments/1
    - **Expected Result:** Details for the shipment from "Alice Smith".
4. **Test Search Functionality:**
    - **Command:** curl -X GET "http://127.0.0.1:5001/api/shipments?sender=Alice"
    - **Expected Result:** Only shipments associated with "Alice".
