PHENIKAA UNIVERSITY
PHENIKAA SCHOOL OF COMPUTING

SOFTWARE ARCHITECTURE

Lab 3: Layered Architecture Implementation (CRUD)

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

1. Project Setup

First, create the root directory for your **QuickShip project:**

```
mkdir quickship_layered
cd quickship_layered
# Tạo môi trường ảo
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
# Cài đặt Flask
pip install Flask
```

```
quickship_layered/
├── app.py                  # Presentation Layer (Controller)
├── business_logic/
│   ├── __init__.py
│   ├── models.py           # Shipment Model
│   └── shipment_service.py # Business Logic Layer
└── persistence/
    ├── __init__.py
    └── shipment_repository.py  # Persistence Layer
```

2. Model Definition (Shipment Model)

This class defines the data structure of a shipment in the QuickShip system.

```
class Shipment:
    def __init__(self, shipment_id, sender, receiver, weight,
status="Pending"):
        self.id = shipment_id
        self.sender = sender
        self.receiver = receiver
        self.weight = weight
        self.status = status

    def to_dict(self):
        return {
            "id": self.id,
            "sender": self.sender,
            "receiver": self.receiver,
            "weight": self.weight,
            "status": self.status
        }
```

3. The Repository Layer (Persistence Layer)
This layer manages the storage of data into a "database" (in this case, a Dictionary in memory).

```
from business_logic.models import Shipment

shipment_db = {}
next_id = 1001 # Mã vận đơn bắt đầu từ 1001

class ShipmentRepository:
    def create(self, sender, receiver, weight):
        global next_id
        shipment_id = str(next_id)
        new_shipment = Shipment(shipment_id, sender, receiver,
weight)
        shipment_db[shipment_id] = new_shipment
        next_id += 1
        return new_shipment

    def find_all(self):
        return list(shipment_db.values())

    def find_by_id(self, shipment_id):
        return shipment_db.get(shipment_id)

    def update_status(self, shipment_id, new_status):
        if shipment_id in shipment_db:
            shipment_db[shipment_id].status = new_status
            return shipment_db[shipment_id]
        return None

    def delete(self, shipment_id):
        if shipment_id in shipment_db:
            del shipment_db[shipment_id]
            return True
        return False
```

4. Business Logic Layer (Service)
This is where QuickShip's rules are processed (e.g., weight cannot be negative, sender/recipient verification).

```
from persistence.shipment_repository import ShipmentRepository

class ShipmentService:
    def __init__(self):
        self.repo = ShipmentRepository()

    def create_shipment(self, sender, receiver, weight):
        # Business Rule: Trọng lượng phải lớn hơn 0
        if weight <= 0:
            raise ValueError("Trọng lượng hàng hóa phải lớn hơn 0kg.")

        # Business Rule: Tên người gửi/nhận không được để trống
        if not sender or not receiver:
            raise ValueError("Thông tin người gửi và người nhận là bắt
buộc.")

        return self.repo.create(sender, receiver, weight)

    def get_all_shipments(self):
        return self.repo.find_all()

    def get_shipment_details(self, shipment_id):
        shipment = self.repo.find_by_id(shipment_id)
        if not shipment:
            raise ValueError(f"Không tìm thấy mã vận đơn:
{shipment_id}")
        return shipment

    def update_shipment_status(self, shipment_id, status):
        valid_statuses = ["Pending", "In Transit", "Delivered",
"Cancelled"]
        if status not in valid_statuses:
            raise ValueError(f"Trạng thái '{status}' không hợp lệ.")

        updated = self.repo.update_status(shipment_id, status)
        if not updated:
            raise ValueError("Cập nhật thất bại. Vận đơn không tồn tại.")
        return updated
```

5. Presentation Layer (Flask Controller)
This layer receives HTTP requests and coordinates data through the Service.

```
from flask import Flask, request, jsonify
from business_logic.shipment_service import ShipmentService

app = Flask(__name__)
shipment_service = ShipmentService()

@app.route('/api/shipments', methods=['POST'])
def create_shipment():
    data = request.json
    try:
        shipment = shipment_service.create_shipment(
            data.get('sender'), data.get('receiver'), data.get('weight')
        )
        return jsonify(shipment.to_dict()), 201
    except ValueError as e:
        return jsonify({"error": str(e)}), 400

@app.route('/api/shipments', methods=['GET'])
def get_shipments():
    shipments = shipment_service.get_all_shipments()
    return jsonify([s.to_dict() for s in shipments]), 200

@app.route('/api/shipments/<shipment_id>', methods=['GET'])
def get_shipment(shipment_id):
    try:
        shipment = shipment_service.get_shipment_details(shipment_id)
        return jsonify(shipment.to_dict()), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 404

@app.route('/api/shipments/<shipment_id>/status',
methods=['PATCH'])
def update_status(shipment_id):
    data = request.json
    try:
        updated =
shipment_service.update_shipment_status(shipment_id,
data.get('status'))
```

```
        return jsonify(updated.to_dict()), 200
    except ValueError as e:
        return jsonify({"error": str(e)}), 400

if __name__ == '__main__':
    app.run(debug=True)
```