**BÁO CÁO THU HOẠCH (SUBMISSION REPORT)**

**MÔN HỌC:** CMU-CS 445 SYSTEM INTEGRATION PRACTICES

**A. THÔNG TIN SINH VIÊN**

|  |  |
| --- | --- |
| Hạng mục | Thông tin chi tiết |
| Họ và Tên | Lê Ngọc Sơn |
| Mã Sinh viên (MSSV) | 29211153774 |
| Lớp HỌC | CMU CS 445 TIS |
| Tên Bài Lab | **LAB 4: DIRECT INTEGRATION - REST API & SERVICE DISCOVERY** |
| Ngày thực hiện | 26/01/2026 |

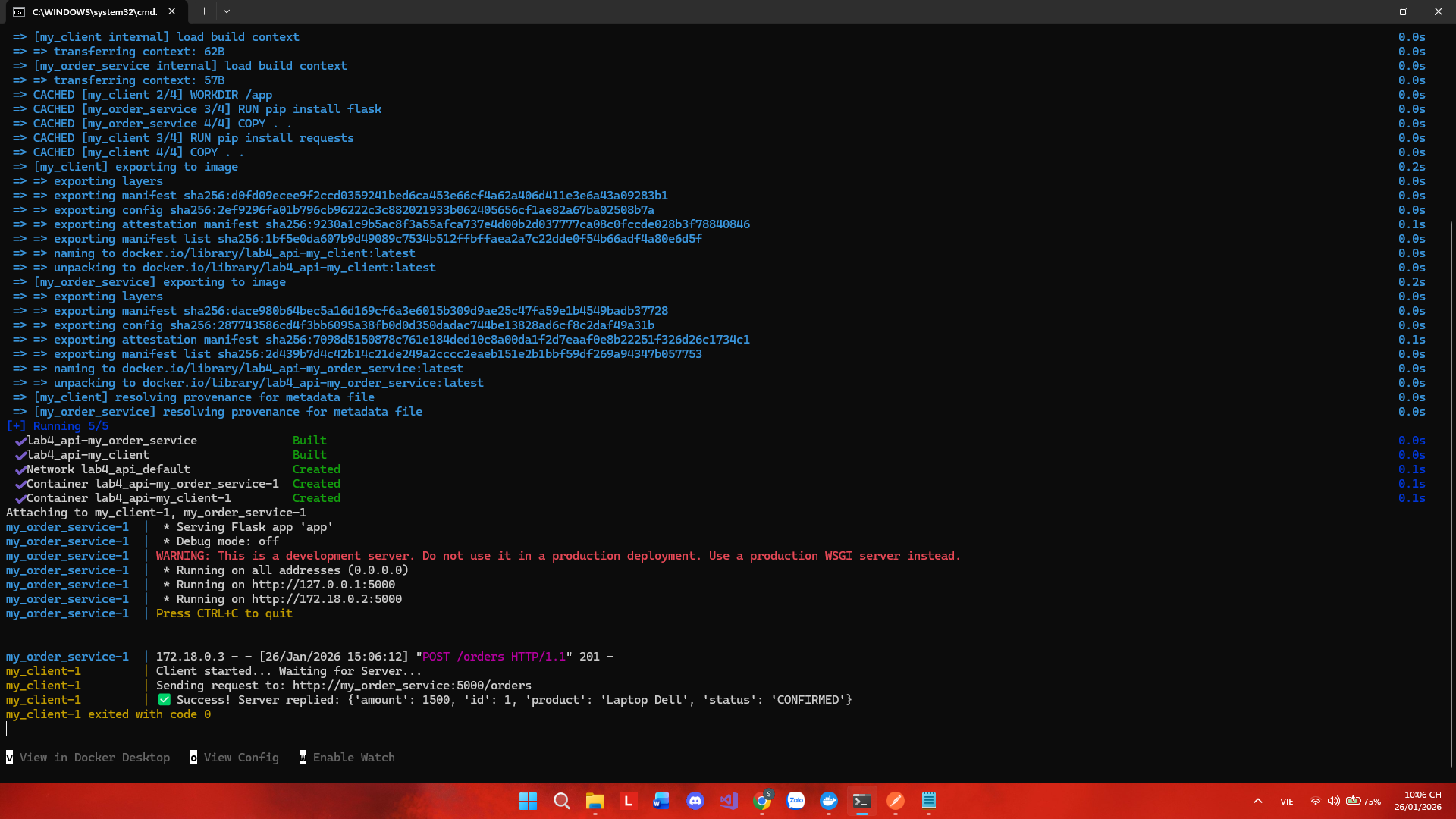
**B. NỘI DUNG BÁO CÁO**

**PHẦN 1: MINH CHỨNG THỰC HÀNH (EVIDENCE OF WORK)**

*Yêu cầu: Chụp ảnh màn hình chứng minh 2 Service đã kết nối thành công.*

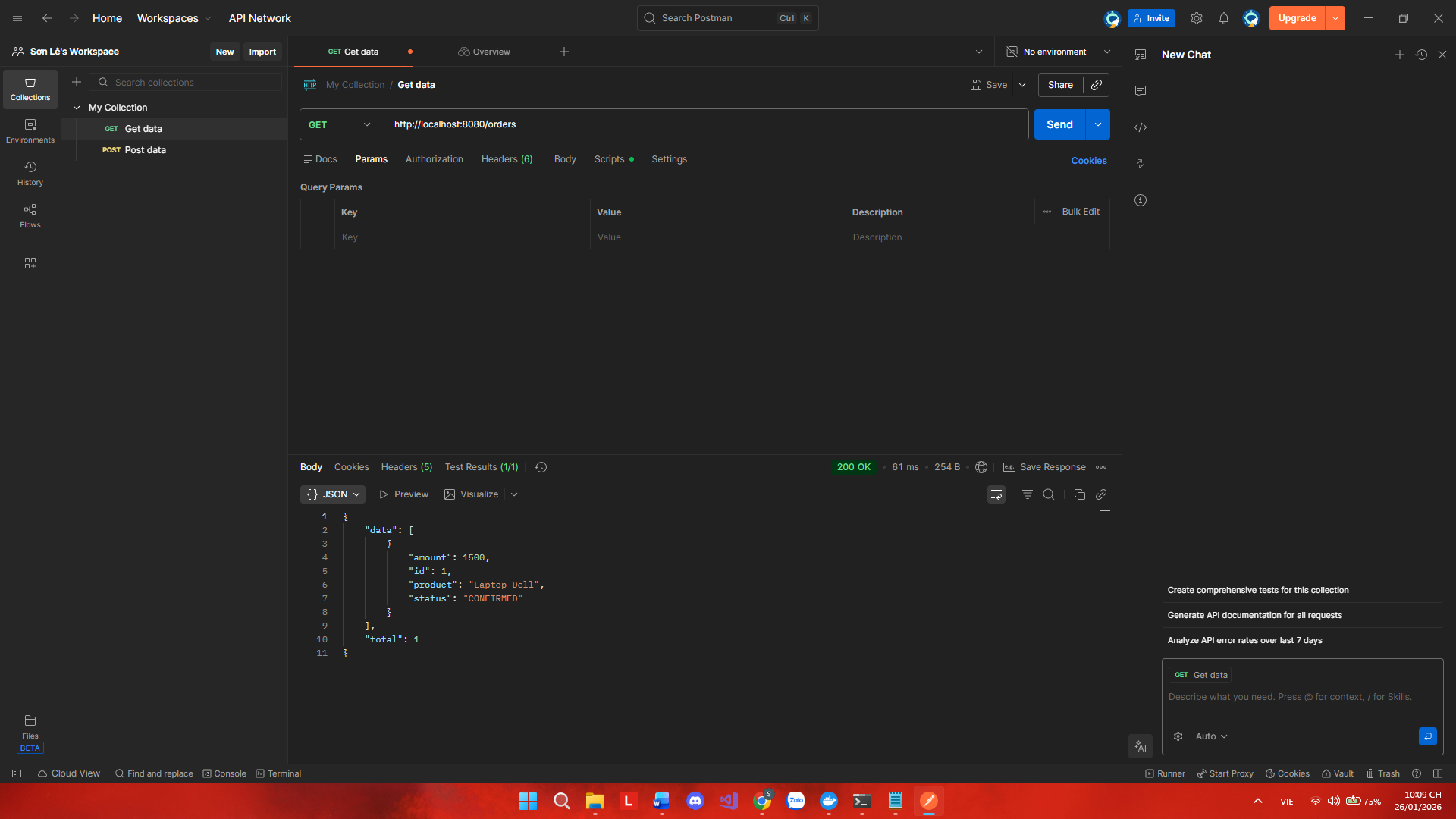
**1. Minh chứng Giao tiếp nội bộ (Internal Communication):**

Dán ảnh màn hình Terminal khi chạy lệnh docker-compose up. Ảnh phải hiện rõ log của Client: "Success! Server replied..." chứng tỏ Client đã gọi được Server.

2. Minh chứng Truy cập từ bên ngoài (External Access):

Yêu cầu: Sử dụng Postman hoặc Trình duyệt.

Dán ảnh kết quả gọi GET http://localhost:8080/orders từ máy thật. Kết quả phải trả về danh sách chứa đơn hàng "Laptop Dell" mà Client vừa tạo.



**PHẦN 2: GIẢI THÍCH CƠ CHẾ (MECHANISM EXPLANATION)**

*Yêu cầu: Giải thích các thành phần trong docker-compose.yml và code Python.*

|  |  |
| --- | --- |
| Đoạn Code / Cấu hình | Giải thích ý nghĩa (Tại sao cần dòng này?) |
| API\_URL = "http://my\_order\_service:5000..." | Dùng tên service để tận dụng **Service Discovery** của Docker. IP của container có thể thay đổi mỗi khi khởi động lại, trong khi tên service là cố định giúp kết nối luôn ổn định. |
| ports: "8080:5000" | **8080** là cổng trên máy Host (người dùng truy cập). **5000** là cổng nội bộ bên trong Container. |
| depends\_on: | Nếu không có dòng này và Client chạy trước khi Server kịp sẵn sàng, Client sẽ bị lỗi **Connection Error** do không tìm thấy dịch vụ để kết nối. |
| request.get\_json() | Hàm trong Flask dùng để **trích xuất (parse)** dữ liệu từ thân (body) của yêu cầu HTTP gửi đến dưới định dạng JSON thành một dictionary trong Python. |
| methods=['POST'] | Vì tạo đơn hàng là hành động **thay đổi trạng thái hệ thống** (thêm dữ liệu mới) và cần bảo mật thông tin. GET chỉ nên dùng để truy xuất dữ liệu, không dùng để gửi dữ liệu nhạy cảm hoặc tạo mới. |

**PHẦN 3: LIÊN HỆ LÝ THUYẾT (THEORY CONNECTION)**

*Yêu cầu: Dựa vào kiến thức Lecture 5 (Services & SOA) để trả lời.*

Câu 1 (Về Service Discovery):

Trong bài Lab, Client gọi Server bằng tên miền my\_order\_service. Ai (thành phần nào) đã đứng ra phân giải cái tên này thành địa chỉ IP để 2 container tìm thấy nhau? (Gợi ý: Internal DNS của Docker).

- Thành phần đứng ra phân giải tên miền là **Embedded DNS (DNS nội bộ)** của Docker.

Câu 2 (Về Tight Coupling - Ghép nối chặt):

Đây là kiểu tích hợp Synchronous (Đồng bộ). Nếu bạn tắt container Server (docker stop), sau đó chạy container Client, chuyện gì sẽ xảy ra? Client có hoạt động bình thường không hay bị lỗi? Điều này nói lên nhược điểm gì của kiểu tích hợp trực tiếp?

- Client sẽ bị lỗi ngay lập tức (thường là lỗi Connection Refused hoặc Timeout) vì không tìm thấy Server để thiết lập kết nối.

- Nhược điểm: Đây là sự phụ thuộc về tính sẵn sàng (Availability Dependency). Hệ thống bị lỗi dây chuyền (Cascading Failure): nếu một dịch vụ chết, các dịch vụ gọi nó cũng ngưng hoạt động, làm giảm độ tin cậy của toàn hệ thống.

Câu 3 (Về Interface Contract):

Client và Server giao tiếp với nhau qua cấu trúc dữ liệu JSON {"product": "...", "amount": ...}. Nếu Server đột ngột đổi chữ "amount" thành "price" mà không báo cho Client, hệ thống có chạy được không? Đây là khái niệm gì trong Tích hợp?

- Hệ thống không chạy được đúng kỳ vọng. Client sẽ không tìm thấy trường amount trong dữ liệu trả về, dẫn đến sai sót logic hoặc lỗi crash ứng dụng.

- Đây là vi phạm Interface Contract (Hợp đồng giao diện), bất kỳ thay đổi nào từ một phía mà không thống nhất sẽ làm phá vỡ sự tương thích (Breaking Changes).

Câu 4 (Liên hệ Đồ án):

Trong Đồ án Module 2, Web Store sẽ gọi sang RabbitMQ. Lúc đó, trong code Web Store bạn sẽ dùng URL là localhost hay rabbitmq? Tại sao?

- Dùng URL là **rabbitmq** (tên service/container name).

- Vì khi triển khai bằng Docker Compose, các dịch vụ nằm trong cùng một mạng ảo. Dùng localhost sẽ trỏ vào chính container Web Store (không có RabbitMQ ở đó). Dùng tên rabbitmq giúp Docker DNS tự động phân giải đúng địa chỉ IP của container RabbitMQ, đảm bảo tính linh hoạt khi container bị restart hoặc thay đổi IP.

**PHẦN 4: BÁO CÁO THỬ THÁCH (CHALLENGE REPORT)**

*Dành cho mục E trong hướng dẫn Lab: Lưu trữ dữ liệu bền vững (Persistence).*

1. Cấu hình Docker Compose đã sửa:

Copy đoạn cấu hình volumes bạn đã thêm vào.

YAML

# [Dán đoạn code volumes vào đây]

# services:

# my\_order\_service:

# volumes:

# - ./data:/app/data

2. Đoạn code Python xử lý file JSON:

Copy đoạn code app.py phần đọc/ghi file.

Python

import os

import json

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

# Đường dẫn file JSON bên trong container (phải khớp với Volume trong docker-compose)

DATA\_FILE = "/app/data/orders.json"

def load\_orders():

"""Đọc dữ liệu từ file JSON nếu có"""

if os.path.exists(DATA\_FILE):

try:

with open(DATA\_FILE, "r") as f:

return json.load(f)

except Exception:

return []

return []

def save\_orders(orders):

"""Ghi dữ liệu vào file JSON"""

# Đảm bảo thư mục /app/data tồn tại

os.makedirs(os.path.dirname(DATA\_FILE), exist\_ok=True)

with open(DATA\_FILE, "w") as f:

json.dump(orders, f, indent=4)

@app.route('/orders', methods=['POST'])

def create\_order():

data = request.get\_json()

if not data or 'product' not in data or 'amount' not in data:

return jsonify({"error": "Invalid Data"}), 400

# 1. Load dữ liệu hiện có từ file

orders = load\_orders()

# 2. Tạo đơn hàng mới

new\_order = {

"id": len(orders) + 1,

"product": data['product'],

"amount": data['amount'],

"status": "CONFIRMED"

}

orders.append(new\_order)

# 3. Lưu lại vào file JSON để không bị mất dữ liệu

save\_orders(orders)

print(f"✅ Đã lưu đơn hàng vào file: {new\_order}")

return jsonify(new\_order), 201

@app.route('/orders', methods=['GET'])

def get\_orders():

orders = load\_orders()

return jsonify({"total": len(orders), "data": orders})

if \_\_name\_\_ == '\_\_main\_\_':

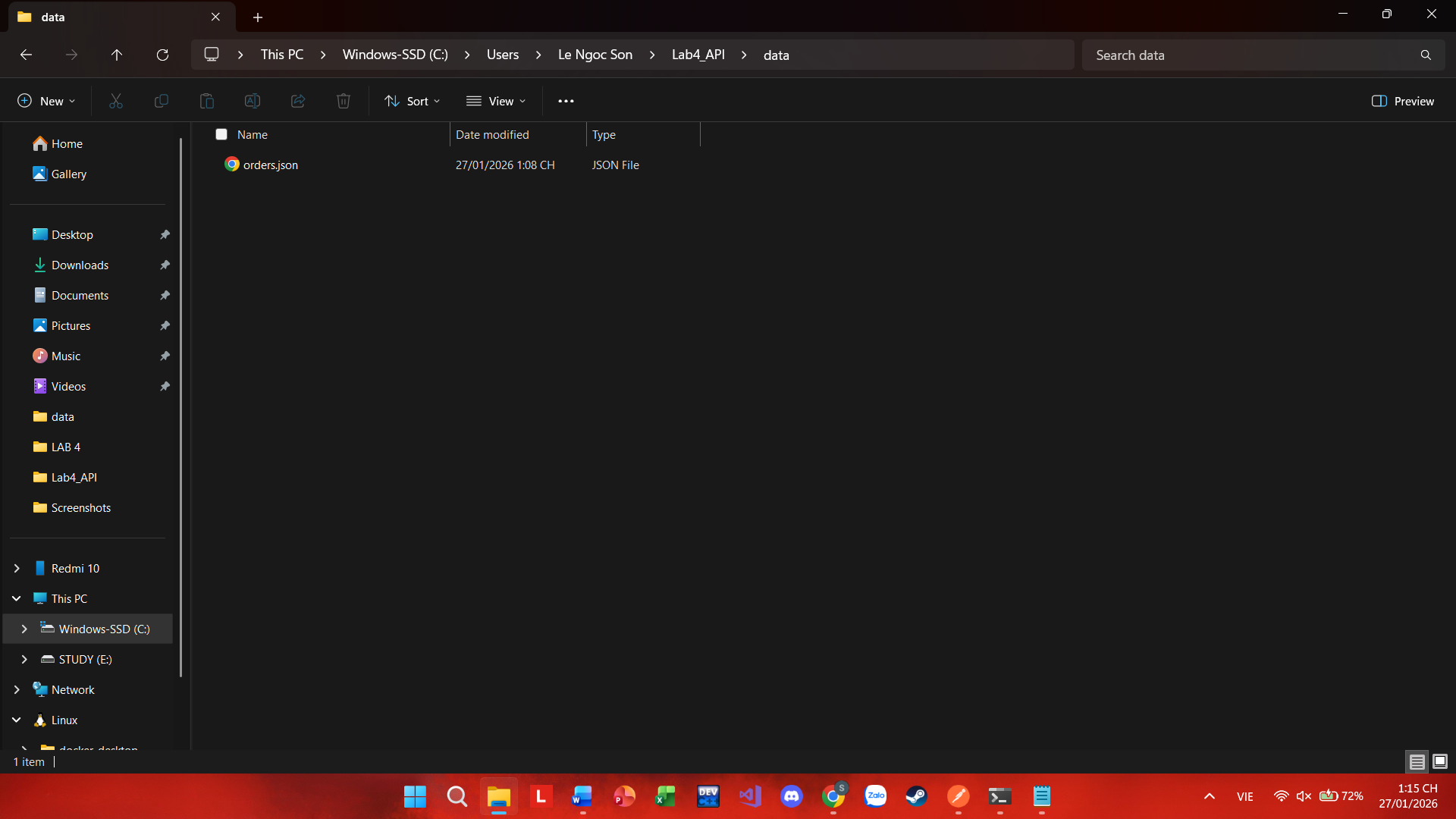
# Chạy trên host 0.0.0.0 để container khác có thể truy cập

app.run(host='0.0.0.0', port=5000)

**3. Kết quả kiểm chứng:**

Cách tôi test: Tạo đơn hàng -> tắt Docker -> bật lại -> vẫn thấy đơn hàng cũ

Dán ảnh minh chứng file orders.json được tạo ra trên máy thật.



**PHẦN 5: TỰ ĐÁNH GIÁ (SELF-REFLECTION)**

* **Mức độ hoàn thành:** [ 100% / 100% ]
* **Khó khăn lớn nhất:**

- Lỗi **Connection Refused** xảy ra khi container Client khởi động và thực hiện gọi API nhanh hơn thời gian container Server kịp sẵn sàng.

- Do đây là kiểu tích hợp trực tiếp , nếu một dịch vụ (Server) bị tắt hoặc gặp sự cố, toàn bộ luồng hoạt động của dịch vụ liên quan (Client) sẽ bị lỗi ngay lập tức.

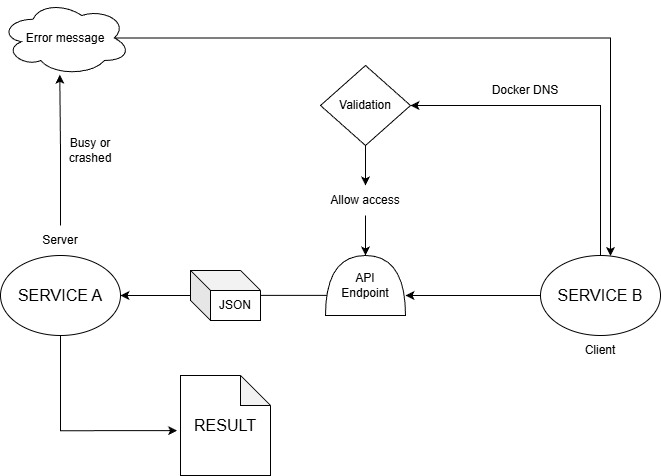
* **Bài học rút ra:**

- Luôn ưu tiên dùng **tên service** (tên service được Docker DNS phân giải tự động) thay vì địa chỉ IP tĩnh để kết nối giữa các container. Điều này giúp hệ thống linh hoạt và ổn định hơn vì IP có thể thay đổi mỗi khi khởi động lại

- Phải coi cấu trúc dữ liệu (Contract) là một cam kết bất biến giữa các bên. Mọi thay đổi về Interface phải được thống nhất để tránh lỗi **Breaking Changes** phá vỡ sự tương thích của hệ thống

**\*GIẢI THÍCH CƠ CHẾ BẰNG CÁCH VẼ KIẾN TRÚC**

Sinh viên chọn vẽ trên [draw.io](http://draw.io) hoặc trên giấy đều được (dùng activity diagram hoặc bất cứ diagram nào) để mô tả kiến trúc , luồng chạy, cơ chế luồng trên (gợi ý, dữ liệu là mạch máu, hệt hống là ống dẫn, vẽ cách dữ liệu được truyền dẫn)

****