**BÀI THỰC HÀNH 06**

**Xây dựng các service sau:**

* Product Service: Quản lý thông tin sản phẩm (tên, giá, mô tả, tồn kho, v.v.)
* Order Service: Quản lý đơn hàng (tạo, xem, hủy đơn hàng, v.v.)
* Customer Service: Quản lý thông tin khách hàng (tên, địa chỉ, thông tin liên lạc, v.v.)

**Giao tiếp giữa các microservice:**

* API Gateway: Đóng vai trò như một điểm truy cập duy nhất cho các client để tương tác với các dịch vụ. API Gateway sẽ chuyển tiếp các yêu cầu đến các Microservices thích hợp.
* REST API / gRPC: Các Microservices sẽ giao tiếp với nhau qua các giao thức như REST API hoặc gRPC.

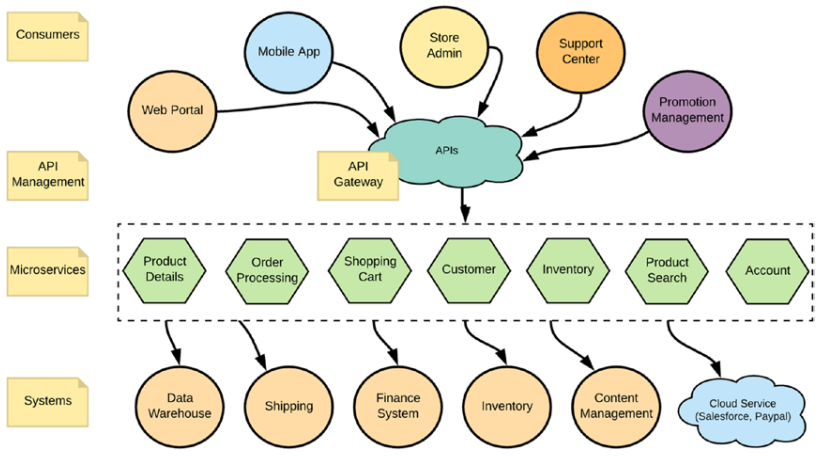
**Các bước phát triển hệ thống:**

* Mỗi Microservice sẽ có cơ sở dữ liệu riêng biệt của mình để tuân thủ nguyên lý "Database per Service" trong Microservices. Ví dụ:
  + Product Service có cơ sở dữ liệu lưu trữ thông tin sản phẩm.
* Mỗi dịch vụ sẽ có các API để thực hiện các thao tác CRUD (Create, Read, Update, Delete). Ví dụ:
  + Product Service: POST /products, GET /products/{id}, PUT /products/{id}, DELETE /products/{id}.

**Yêu cầu:**

* Containerization: Docker + Docker Compose
* Database: PostgreSQL tự do lựa chọn.
* Vẽ sơ đồ việc gọi nhau giữa các services

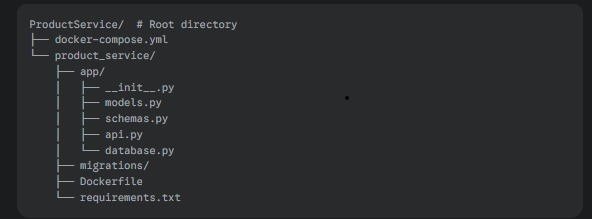
**PRODUCT SERVICE:**



**Giải thích sơ đồ:**

* **Client:** Người dùng hoặc ứng dụng client gửi yêu cầu đến hệ thống.
* **API Gateway:**
  + Đóng vai trò là cổng vào duy nhất, tiếp nhận tất cả các yêu cầu từ client.
  + Chuyển tiếp yêu cầu đến các microservice tương ứng (ở đây là Product Service).
  + Có thể thực hiện các chức năng như xác thực, ủy quyền, rate limiting, logging, v.v.
* **Product Service:**
  + Quản lý thông tin sản phẩm (tên, giá, mô tả, tồn kho).
  + Cung cấp các API CRUD (Create, Read, Update, Delete) để thao tác với dữ liệu sản phẩm.
  + Sử dụng cơ sở dữ liệu riêng (PostgreSQL).
* **PostgreSQL:** Cơ sở dữ liệu của Product Service, lưu trữ thông tin sản phẩm.
* **Giao tiếp:** Các thành phần giao tiếp với nhau thông qua giao thức HTTP (REST API).

Bước 1: Tạo cấu trúc thư mục dưới:



Bước 2: Thêm nội dung vào các file

Bước 3: Mở Terminal, chạy các lệnh dưới:

* docker-compose up -d
* docker-compose exec product\_service flask db init
* docker-compose exec product\_service flask db migrate
* docker-compose exec product\_service flask db upgrade

Bước 4: Kết nối với PostgreSQL

* Mở pgAdmin.
* Nhấp chuột phải vào "Servers" và chọn "Create" -> "Server...".
* Trong tab "General", đặt tên cho server (ví dụ: "Local PostgreSQL").
* Chuyển sang tab "Connection":
* **Host name/address:** localhost
* **Port:** 5432
* **Maintenance database:** postgres
* **Username:** user
* **Password:** password
* Nhấp "Save".

Bước 5: Chèn dữ liệu mẫu:

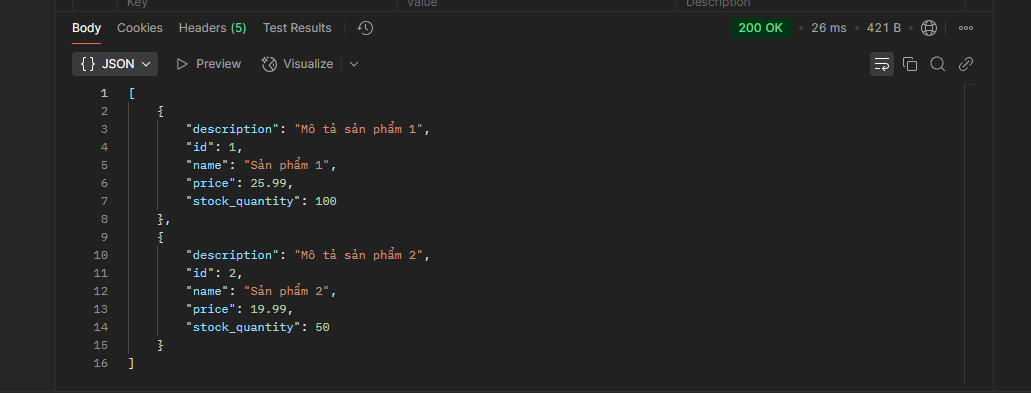
INSERT INTO products (name, price, description, stock\_quantity) VALUES

('Sản phẩm 1', 25.99, 'Mô tả sản phẩm 1', 100),

('Sản phẩm 2', 19.99, 'Mô tả sản phẩm 2', 50);

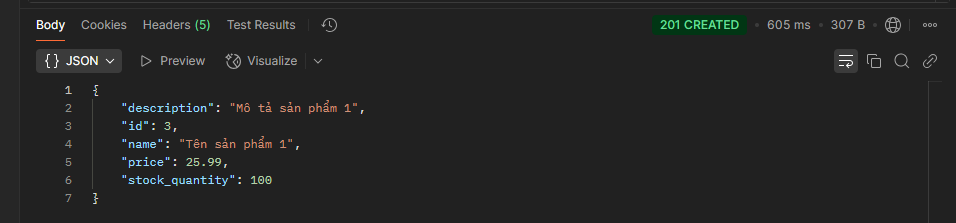
Bước 6: Test với Postman

* Lấy danh sách sản phẩm: GET http://localhost:5000/products

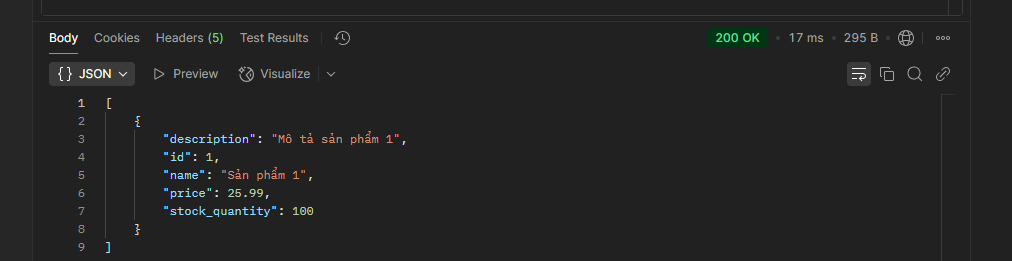


* Thêm sản phẩm: POST <http://localhost:5000/products>
  + file json:

{ "name": "Tên sản phẩm 1", "price": 25.99, "description": "Mô tả sản phẩm 1", "stock\_quantity": 100 }

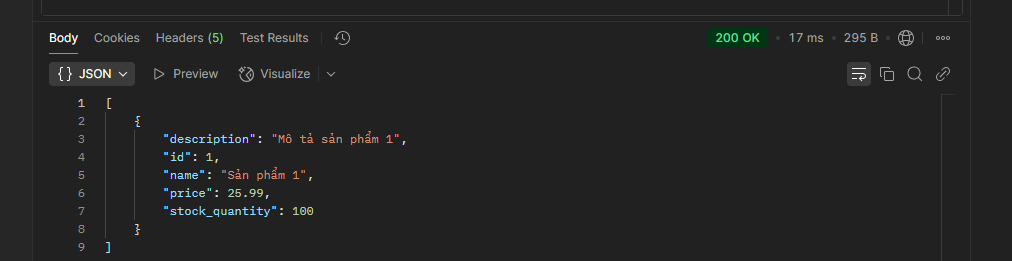


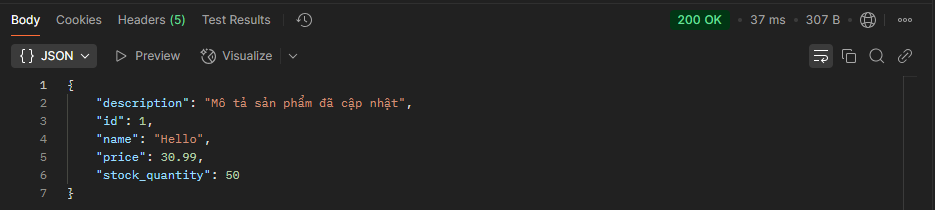
* Xóa sản phẩm: DELETE http://localhost:5000/products/2



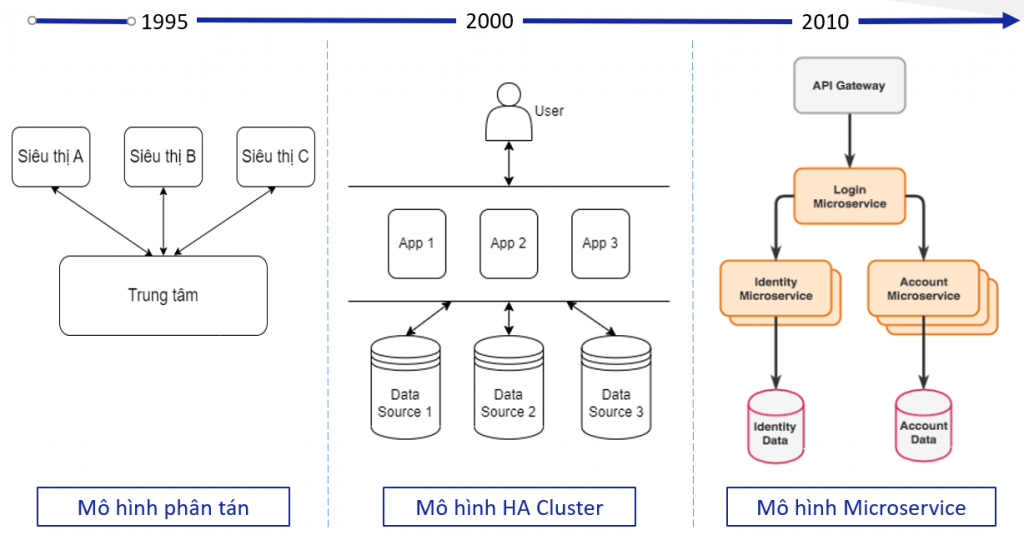
* Cập nhật sản phẩm: PUT <http://localhost:5000/products/1>
  + file json:

{ "name": "Tên sản phẩm đã cập nhật", "price": 30.99, "description": "Mô tả sản phẩm đã cập nhật", "stock\_quantity": 50 }





**ORDER SERVICE:**

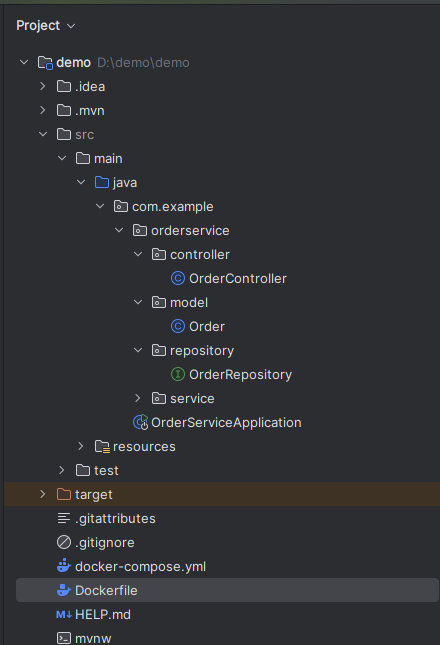


**Giải thích sơ đồ:**

* **Client:** Ứng dụng client (web, mobile) gửi yêu cầu đến hệ thống.
* **API Gateway:**
  + Đóng vai trò là cổng vào duy nhất, tiếp nhận tất cả các yêu cầu từ client.
  + Chuyển tiếp yêu cầu đến các microservice tương ứng (Product Service, Order Service).
  + Thực hiện các chức năng như xác thực, ủy quyền, load balancing, rate limiting,...
* **Product Service:**
  + Quản lý thông tin sản phẩm (tên, giá, mô tả, tồn kho).
  + Cung cấp các API CRUD để thao tác với dữ liệu sản phẩm.
  + Sử dụng cơ sở dữ liệu riêng (PostgreSQL).
* **Order Service:**
  + Quản lý đơn hàng (tạo, xem, hủy đơn hàng).
  + Cung cấp các API CRUD để thao tác với dữ liệu đơn hàng.
  + Sử dụng cơ sở dữ liệu riêng (PostgreSQL).
* **PostgreSQL:** Cơ sở dữ liệu được sử dụng cho cả Product Service và Order Service.
* **Giao tiếp:** Các thành phần giao tiếp với nhau thông qua giao thức HTTP (REST API).

Cấu trúc thư mục

Bước 1: Tạo theo cấu trúc



Bước 2: Thêm nội dung vào các file

Bước 3: Kết nối với PostgreSQL

* Mở pgAdmin.
* Nhấp chuột phải vào "Servers" và chọn "Create" -> "Server...".
* Trong tab "General", đặt tên cho server (ví dụ: "Local PostgreSQL").
* Chuyển sang tab "Connection":
* **Host name/address:** localhost
* **Port:** 5432
* **Maintenance database:** postgres
* **Username:** user
* **Password:** password
* Nhấp "Save".
* Tạo db order-db

Bước 4: Mở Terminal, chạy các lệnh dưới:

* mvc clean package
* docker-compose up -d

Bước 5: Chèn dữ liệu mẫu:

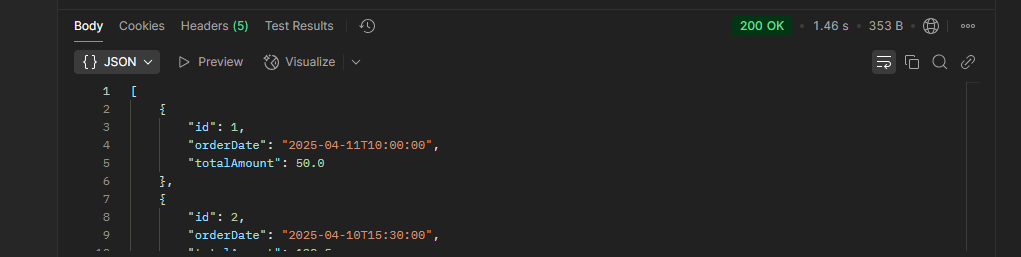
INSERT INTO orders (order\_date, total\_amount) VALUES ('2025-04-11 10:00:00', 50.00);

INSERT INTO orders (order\_date, total\_amount) VALUES ('2025-04-10 15:30:00', 120.50);

INSERT INTO orders (order\_date, total\_amount) VALUES ('2025-04-11 11:45:00', 75.25);

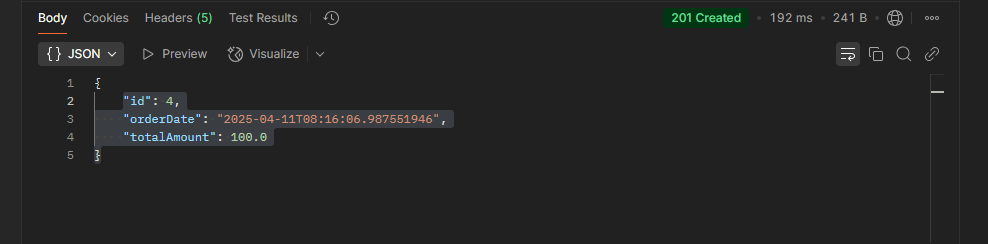
Bước 6: Test Postman

* Lấy danh sách order: GET http://localhost:8080/orders

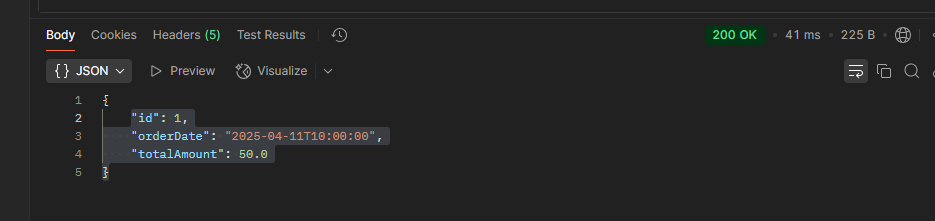


* Thêm order: POST <http://localhost:8080/orders>
  + file json:

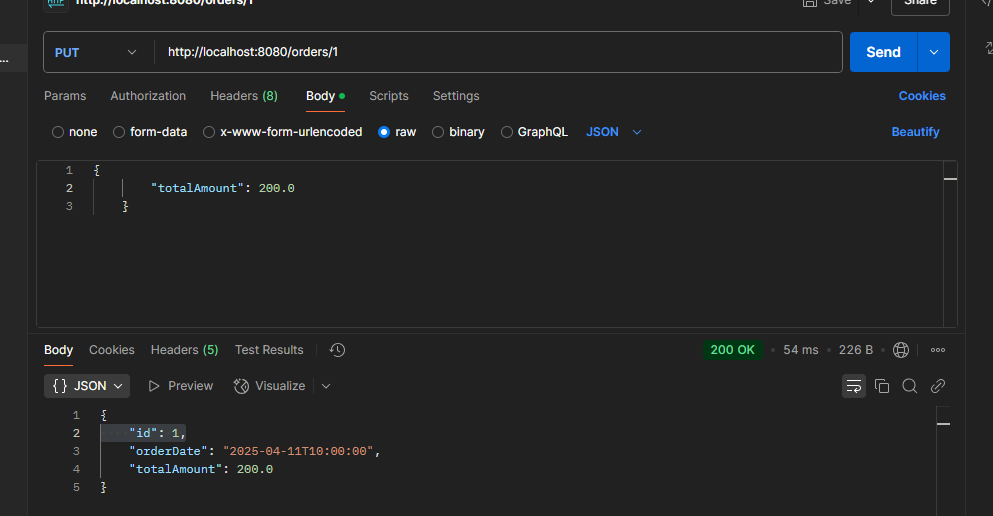
{ "orderDate": "2025-04-11T15:20:00", "totalAmount": 75.50 }



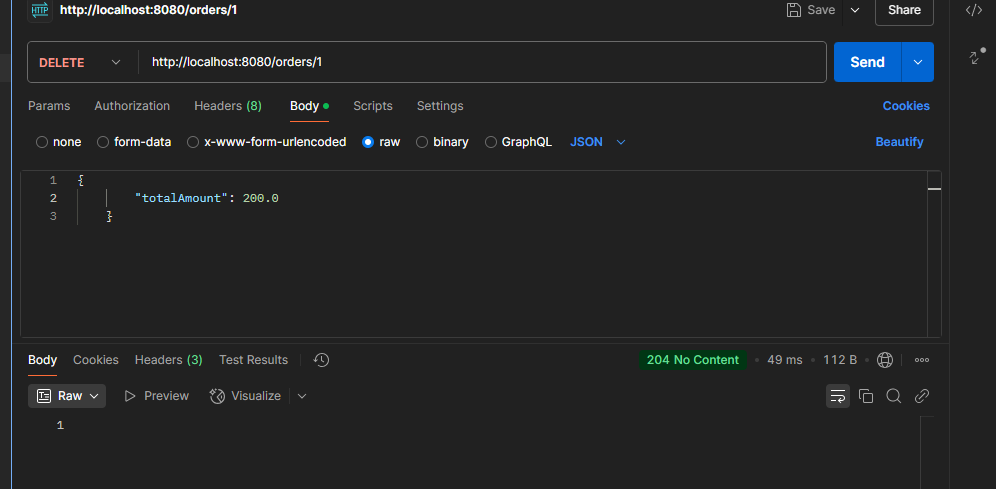
* Xem order theo id: GET http://localhost:8080/orders/1



* Cập nhật order: PUT http://localhost:8080/orders/1



* Xóa order:



**CUSTOMER SERVICE:**

Bước 1: Tạo cấu trúc thư mục dự án:



Bước 2: Thêm nội dung các file

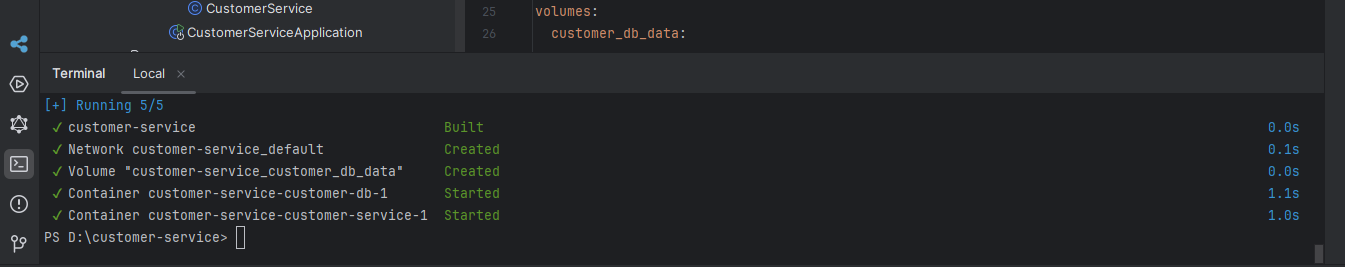
Bước 3: Kết nối với PostgreSQL

* Mở pgAdmin.
* Nhấp chuột phải vào "Servers" và chọn "Create" -> "Server...".
* Trong tab "General", đặt tên cho server (ví dụ: "Local PostgreSQL").
* Chuyển sang tab "Connection":
* **Host name/address:** localhost
* **Port:** 5432
* **Maintenance database:** postgres
* **Username:** postgres
* **Password:** 123456789
* Nhấp "Save".
* Tạo db customerdb

Bước 4: Tạo file docker-compose.yml ở thư mục gốc của project.

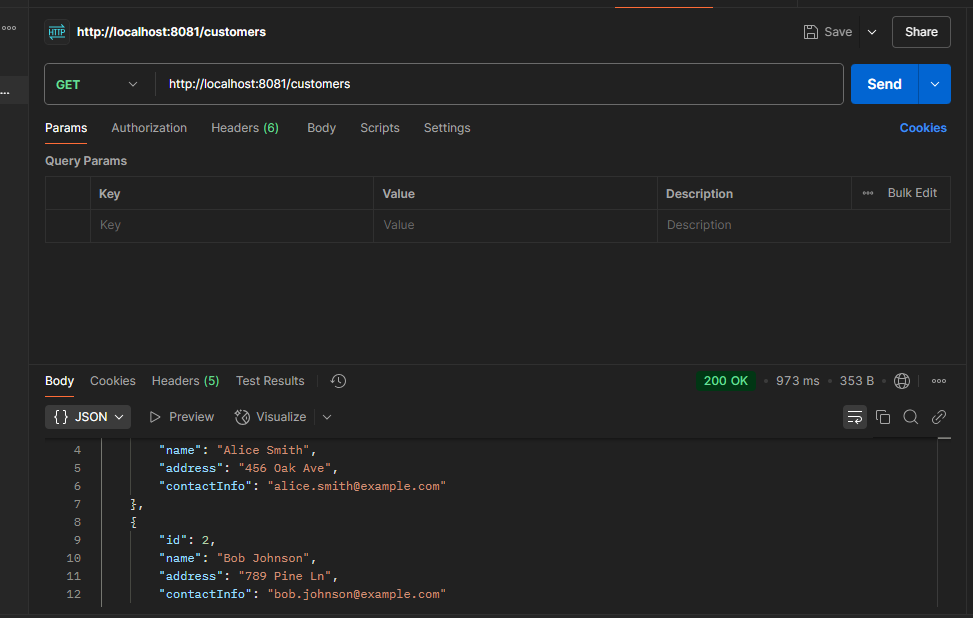
Bước 5**:** chạy lệnh mvc clean package để build file JAR.

Bước 6: Chạy Docker Compose từ terminal bằng lệnh docker-compose up -d trong thư mục gốc của project.

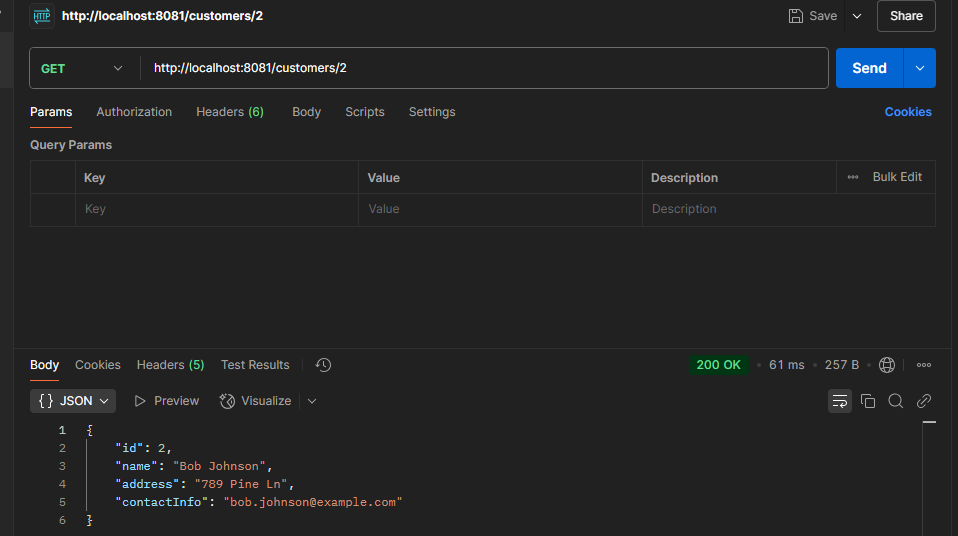


Bước 7:

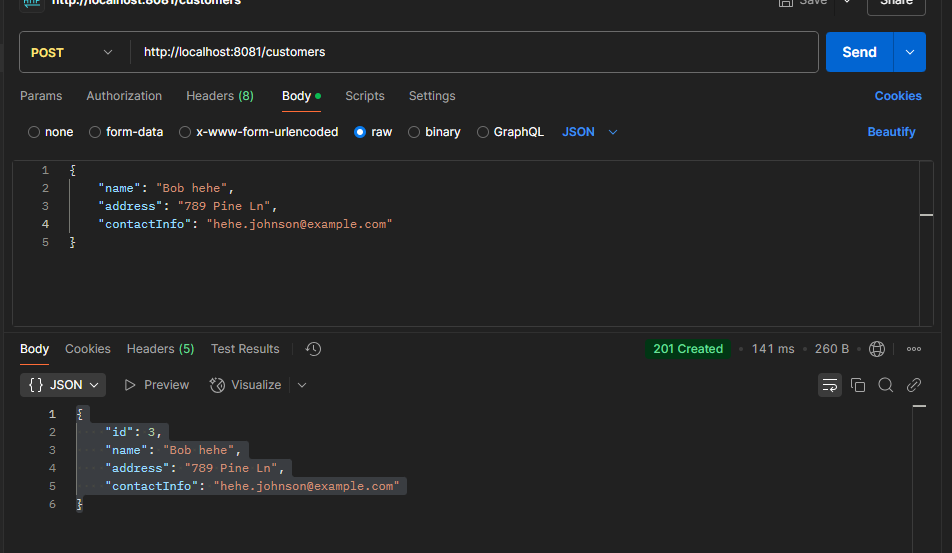
* Lấy danh sách customer



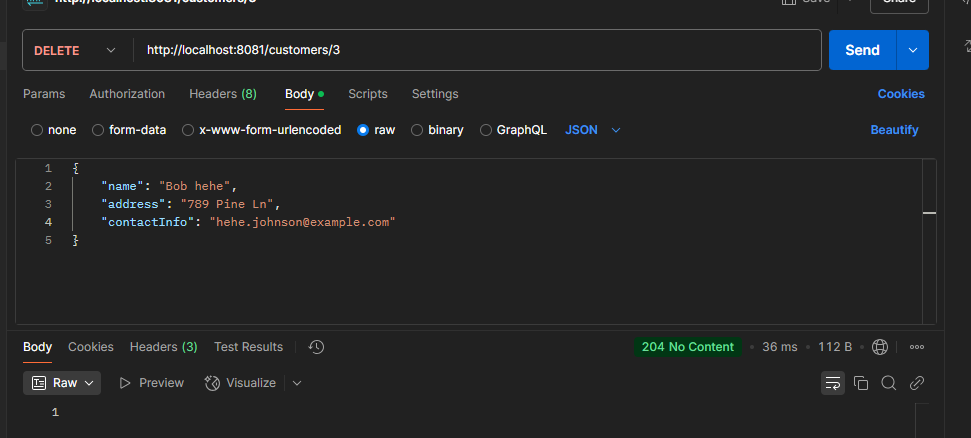
* lấy customer theo id:



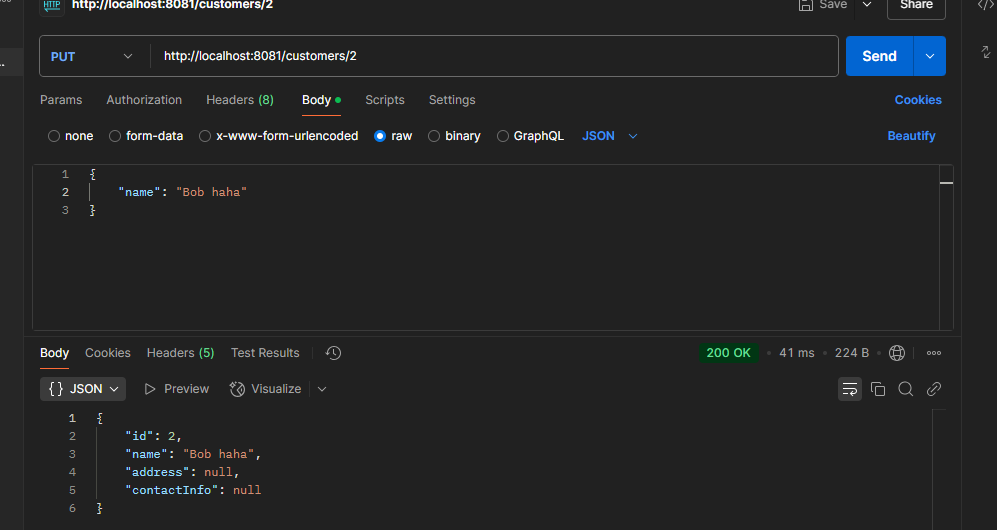
* Thêm customer:



* xóa customer



* cập nhật customer:



**BÀI THỰC HÀNH 07**

**PAYMENT SERVICE**

Yêu cầu: Xây dựng thêm service như sau:

- Payment Service: Xử lý các giao dịch thanh toán (xác nhận thanh toán, hoàn tiền, v.v.)

Giao tiếp giữa các Microservices

- API Gateway: Đóng vai trò như một điểm truy cập duy nhất cho các client để tương tác với các dịch vụ. API Gateway sẽ chuyển tiếp các yêu cầu đến các Microservices thích hợp.

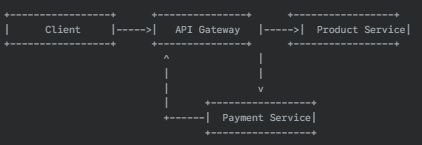
Xử lý thanh toán và giao hàng

- Payment Service: Xử lý các giao dịch thanh toán và cập nhật trạng thái thanh toán (đã thanh toán, chưa thanh toán).

Yêu cầu:

- Containerization: Docker + dockercompose

**Kiến trúc tổng quan:**



**Luồng xử lý chính:**

1. Client gửi yêu cầu đến API Gateway.
2. API Gateway định tuyến yêu cầu đến Microservice phù hợp (Product Service hoặc Payment Service).
3. **Xử lý thanh toán:**
   * Client gửi yêu cầu thanh toán (thông tin đơn hàng, phương thức thanh toán) đến API Gateway.
   * API Gateway chuyển yêu cầu đến Payment Service.
   * Payment Service xử lý giao dịch thanh toán (gọi đến các cổng thanh toán nếu cần).
   * Payment Service cập nhật trạng thái thanh toán vào database của nó.
   * Payment Service có thể gửi thông báo (ví dụ: sự kiện) đến các service khác nếu cần (ví dụ: Order Service để tạo đơn hàng).
4. **Xử lý giao hàng (ví dụ, tích hợp với Shipping Service - giả định):**
   * Sau khi thanh toán thành công, Order Service (hoặc Payment Service) có thể tương tác với Shipping Service để bắt đầu quá trình giao hàng.

**Xây dựng payment-service**

**Bước 1:** Xây dựng cấu trúc dự án

payment-root/

├── api\_gateway/

│ └── main.py

│ └── requirements.txt

├── payment\_service/

│ └── main.py

│ └── requirements.txt

├── docker-compose.yml

**Bước 2:** Thêm nội dung các file

**Bước 3**: Chạy câu lệnh **docker build -t payment-service .**

**Xây dựng API Gateway**

**Bước 5:** Thêm nội dung các file

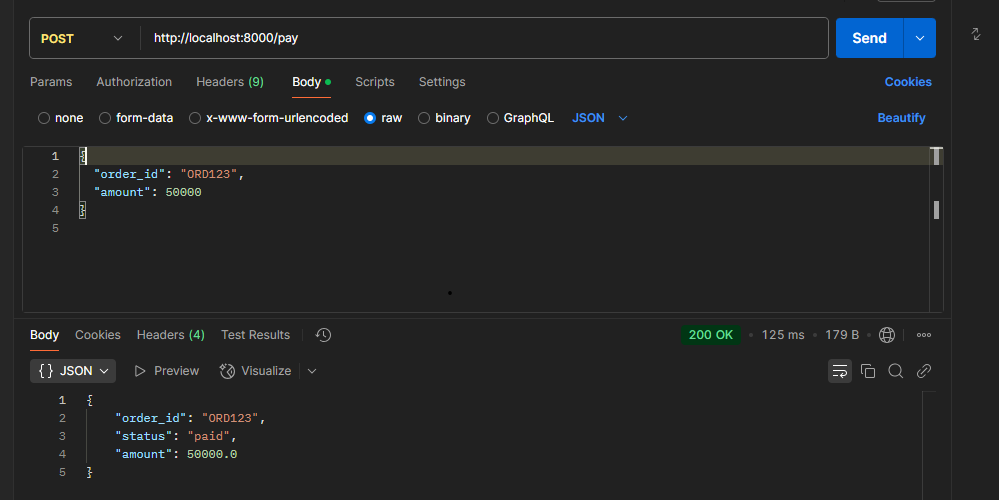
**Bước 6:** Chạy câu lệnh **docker build -t api-gateway .**

**Bước 7:** Tạo docker-compose.yml

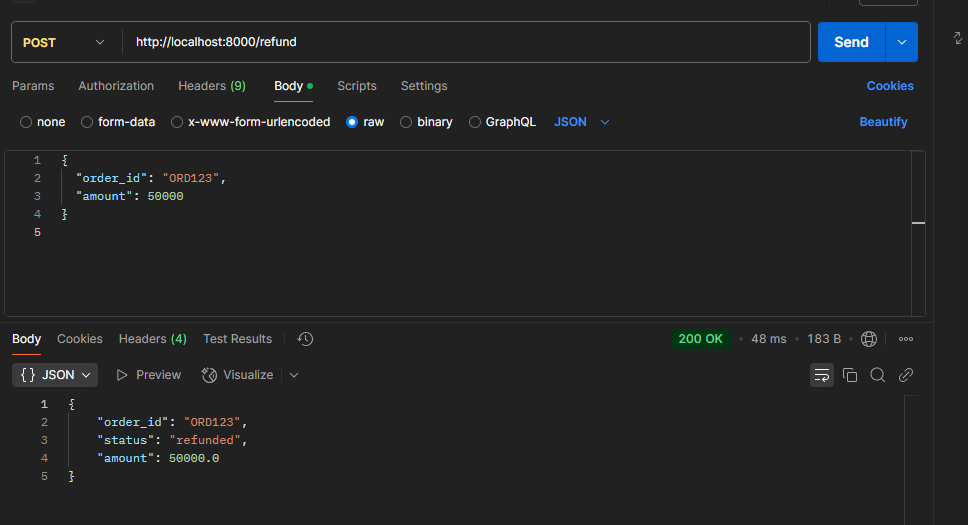
**Bước 8**: Chạy lệnh docker-compose up -d

Bước 9: Test postman

* Thanh toán:



* Hoàn tiền:



**Inventory Service**

**Bước 1**: Thêm vào payment-root

├── inventory\_service/

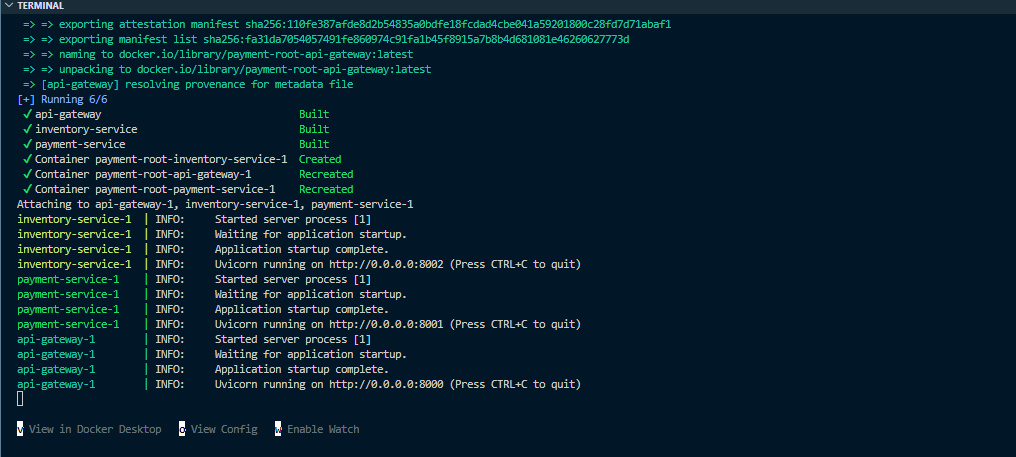
│ └── main.py

│ └── requirements.txt

│ └── Dockerfile

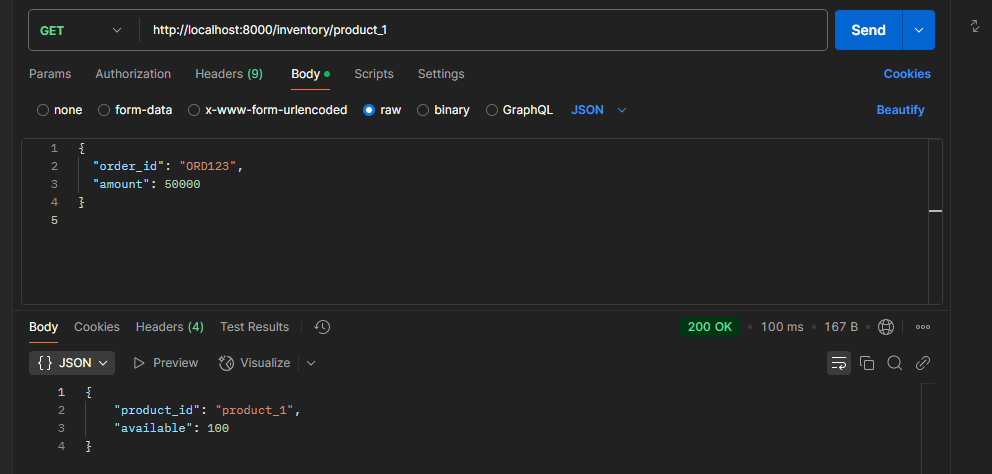
**Bước 2**: Thêm nội dung các file

**Bước 3:** Chạy lại lệnh **docker-compose up --build**



**Bước 4**: Test postman

* Lấy tồn kho: GET http://localhost:8000/inventory/product\_1



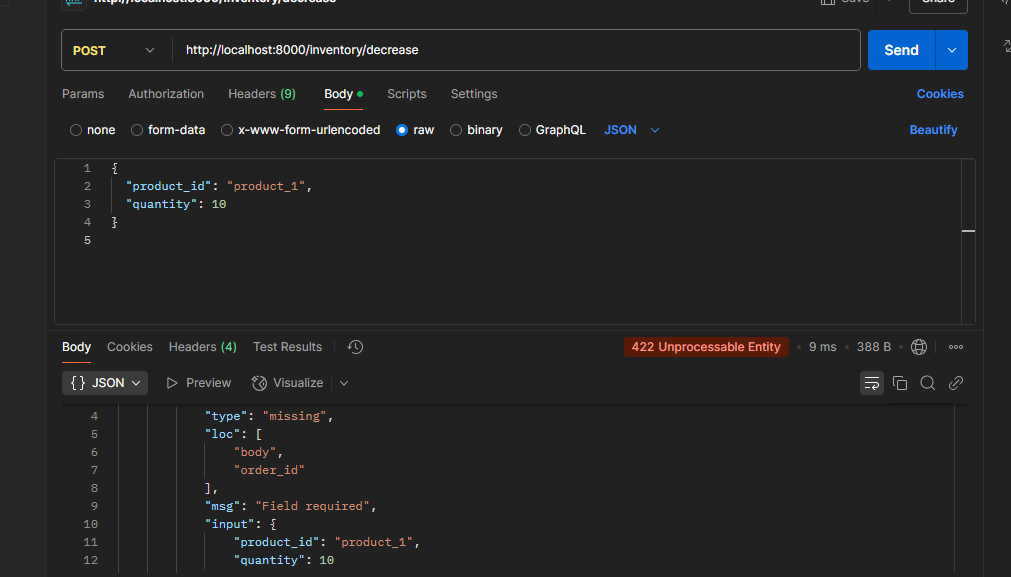
* Trừ tồn kho: POST <http://localhost:8000/inventory/decrease>
* file json:

{

"product\_id": "product\_1",

"quantity": 10

}



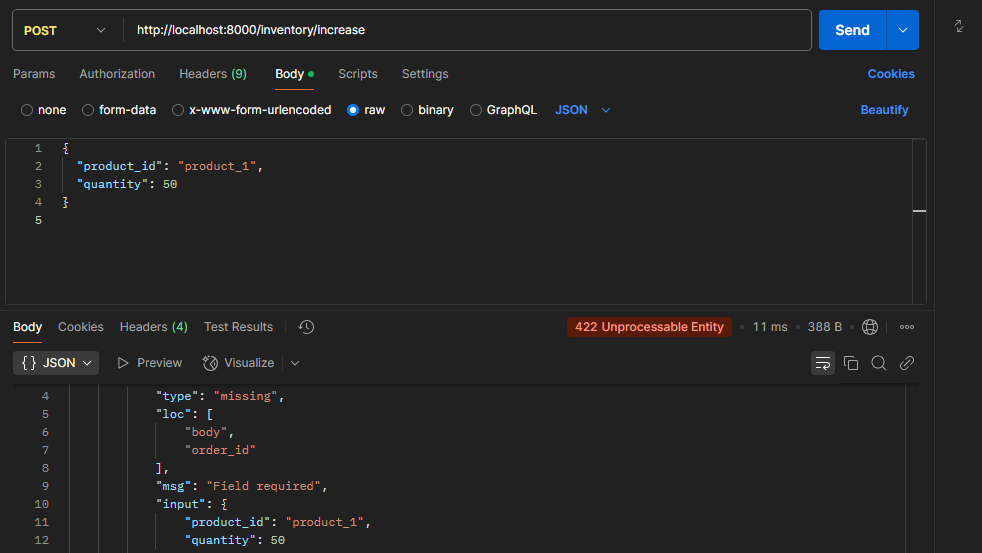
* Tăng tồn kho: POST [http://localhost:8000/inventory/](http://localhost:8000/inventory/decrease)increase
* file json:

{

"product\_id": "product\_1",

"quantity": 50

}



**SHIPPING SERVICE**

Bước 1: Thêm vào payment-root

payment-root/

├── shipping\_service/

│ └── main.py

│ └── requirements.txt

└── Dockerfile

├── docker-compose.yml

Bước 2: Thêm nội dung các file, cập nhật lại docker-compose.yml

Bước 3: Chạy lại lệnh

Bước 4: Test postman

* Tạo đơn giao hàng: POST http://localhost:8000/shipping/create

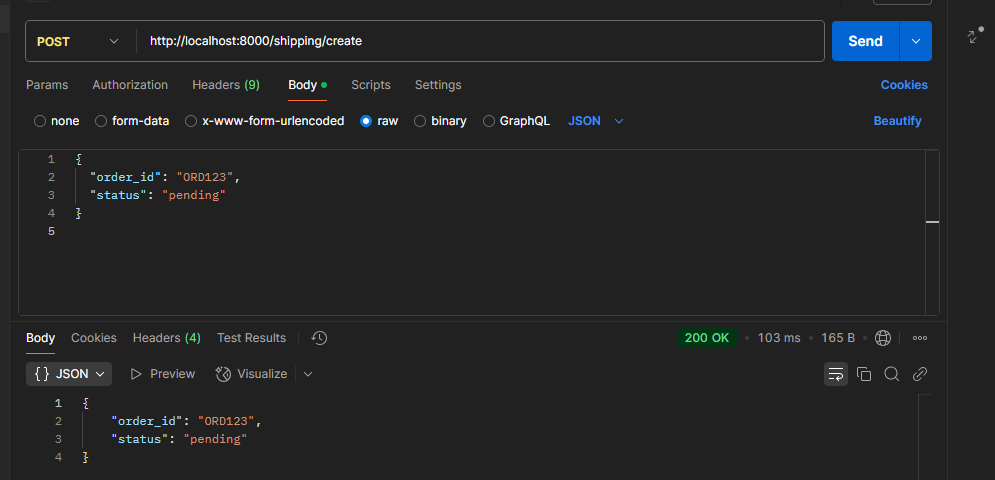
file json:

{

"order\_id": "ORD123",

"status": "pending"

}



* Cập nhật trạng thái:PUT http://localhost:8000/shipping/update

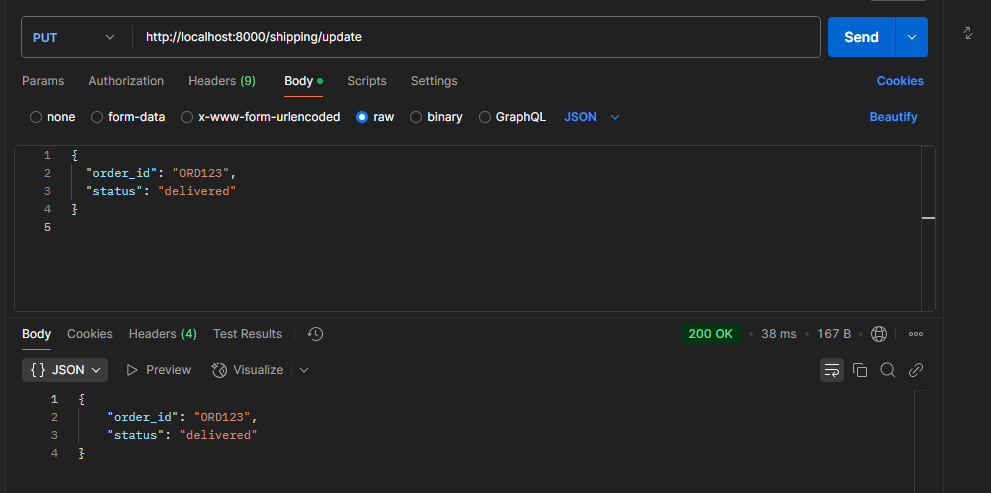
file json:

{

"order\_id": "ORD123",

"status": "delivered"

}



* Lấy trạng thái đơn hàng:GET http://localhost:8000/shipping/ORD123

