Microservices Workflow (E-Commerce Example)

1. Product Service (8081) – Product Catalog Management

Entity: `Product`
Responsibilities:

- Add new products to the catalog.
- Update product details (price, stock, etc.).
- View product details by ID or list all products.

Example Workflow:

1. Admin adds a product: *Laptop, $1200, stock=10*.
2. Customers can view all products or search for specific products.
3. Stock decreases when an order is successfully placed (through **order-service**).

2. Order Service (8082) – Order Management

Entity: `Order`
Responsibilities:

- Accept order requests from customers.
- Validate product availability via **product-service**.
- Calculate total price based on product details.
- Forward payment request to **payment-service**.
- Update order status after payment confirmation.

Example Workflow:

1. Customer places an order for Product ID `P101` (Quantity = 2).
2. **Order Service → Product Service**:
    - Checks if Product `P101` exists and stock is ≥ 2.
3. If available, the order is **tentatively created** with status `"PENDING_PAYMENT"`.
4. **Order Service → Payment Service**: Sends payment request for the total amount.
5. If payment is **successful**, order status is updated to `"CONFIRMED"`.
6. **Order Service → Product Service**: Reduces stock count by the quantity ordered.

3. Payment Service (8083) – Payment Processing

Entity: `Payment`
Responsibilities:

- Receive payment requests from **order-service**.
- Validate payment details (amount, order ID).
- Simulate/execute payment transaction (e.g., with a payment gateway).
- Send payment confirmation back to **order-service**.

Example Workflow:

Receives payment request from **order-service**:

```
{
  "orderId": "O5001",
  "amount": 2400,
  "paymentMethod": "Credit Card"
}
```

4. Processes payment and returns status: `"SUCCESS"`.
5. In case of failure, returns `"FAILED"`, and order remains `"PENDING_PAYMENT"`.

🔷 End-to-End Flow (Order Placement Example)

Scenario:

A customer buys **2 laptops** costing **$1200 each**.

Step 1 – Order Request

- Customer sends request to **order-service**:
  `POST /orders` → `{ "productId": "P101", "quantity": 2 }`

Step 2 – Validate Product Availability

- **Order Service → Product Service**: `GET /products/P101`
- **Product Service**: Returns `{ "name": "Laptop", "price": 1200, "stock": 10 }`

Step 3 – Calculate Price & Request Payment

- **Order Service**: Calculates total price = `1200 × 2 = $2400`.
- Sends payment request to **payment-service**:
  `POST /payments` → `{ "orderId": "O5001", "amount": 2400 }`

Step 4 – Process Payment

- **Payment Service**: Confirms `"SUCCESS"`.

Step 5 – Update Order & Reduce Stock

- **Order Service**: Updates order status to `"CONFIRMED"`.
- **Order Service** → **Product Service**: Sends `PUT /products/P101/reduceStock?qty=2` to update stock from 10 → 8.

Step 6 – Response to Customer

Returns:

```
{
  "orderId": "O5001",
  "status": "CONFIRMED",
  "totalAmount": 2400
}
```

🔷 How Eureka Helps in This Workflow

Without Eureka:

- **Order Service** would need hardcoded URLs for Product & Payment services.

With Eureka:

- **Order Service** simply calls:
  - `http://product-service/products/P101`
  - `http://payment-service/payments`
- Eureka resolves actual IP:Port dynamically and supports multiple instances (load balancing).