Scenario: You are building a simplified backend system for an e-commerce platform. When a customer places an order, the order details must be broadcast to multiple downstream services (e.g., inventory management, billing, and shipping) via RabbitMQ.

Requirements:

1. Producer Application (Cart Service):
    - Implement an API endpoint (/create-order) that accepts a request to create a new order. The request must strictly follow this format for the order object. For simplicity, the endpoint will receive **only** the orderId and number of items(you will generate the array internally) fields from the API, the rest should be randomly auto generated internally.

Order Object Structure:

```
{
  "orderId": "string",          // Unique identifier for the order
  "customerId": "string",       // Unique identifier for the customer
  "orderDate": "ISO8601",       // Timestamp in ISO 8601 format
  "items": [                    // Array of order items, with each item having:
    {
      "itemId": "string",       // Unique identifier for the item
      "quantity": "integer",    // Quantity of the item (must be greater than 0)
      "price": "decimal"        // Price of a single unit of the item
    }
  ],
  "totalAmount": "decimal",     // Total cost of the order (must equal the sum of item prices * quantities)
  "currency": "string",         // Currency code (e.g., USD, EUR)
  "status": "string"            // Status of the order (e.g., 'pending', 'confirmed')
}
```

Example Order Payload after internally generating the data:

```
{
  "orderId": "ORD-12345",
  "customerId": "CUST-67890",
  "orderDate": "2024-10-19T10:45:00Z",
  "items": [
    {"itemId": "ITEM-001", "quantity": 2, "price": 19.99},
    {"itemId": "ITEM-002", "quantity": 1, "price": 39.99}
  ],
  "totalAmount": 79.97,
  "currency": "USD",
  "status": "pending"
}
```

- When a new order is created, the producer should publish the order details as an event to RabbitMQ.
- The event must be broadcast to all consumers, and the order object should be serialized as JSON before sending.

Strict Validation Rules for the Producer:

- Ensure that the input fields are valid inputs.
- If any of these validation rules fail, return an appropriate error response with a clear message.

2. Consumer Application (Order Service):
    - The consumer should connect to RabbitMQ and receive **only** new order events where the order status is “new”.
    - Upon receiving an order event, the consumer will perform the following actions:

1. Calculate Shipping Costs: calculate the shipping costs: shippingCost = 2% of totalAmount.
2. The consumer should log the order details and store them in memory or a simple in-memory database for retrieval. including the shipping cost.

- Implement an API endpoint (/order-details) that gets an orderId input and returns the order details and shipping costs.

Submission instructions:

you need to submit 3 files, 2 docker-compose.yml files (one for the producer and one for the consumer), and one readme file.

In the readme file you will answer the following questions:

1. Your full name and ID number
2. The full Url's and type of API request to generate the producer and the consumer applications.
3. What type of exchange you chose and why?
4. Is there a binding key on the consumer? If so, what is it and why?
5. Which service declared the exchange and queue and why?

Important instructions:

1. You must handle basic errors that can be caused and return the correct response in the API (meaning error codes that are not 200OK like 500 with a message)

2. You can write the applications in any language that RabbitMQ supports and you can submit the exercise as requested.
3. A project that the docker-compose file will not work and the API functions are not reachable will get 0! you must verify your submission is valid!
4. Submission Due date: 15.12.24