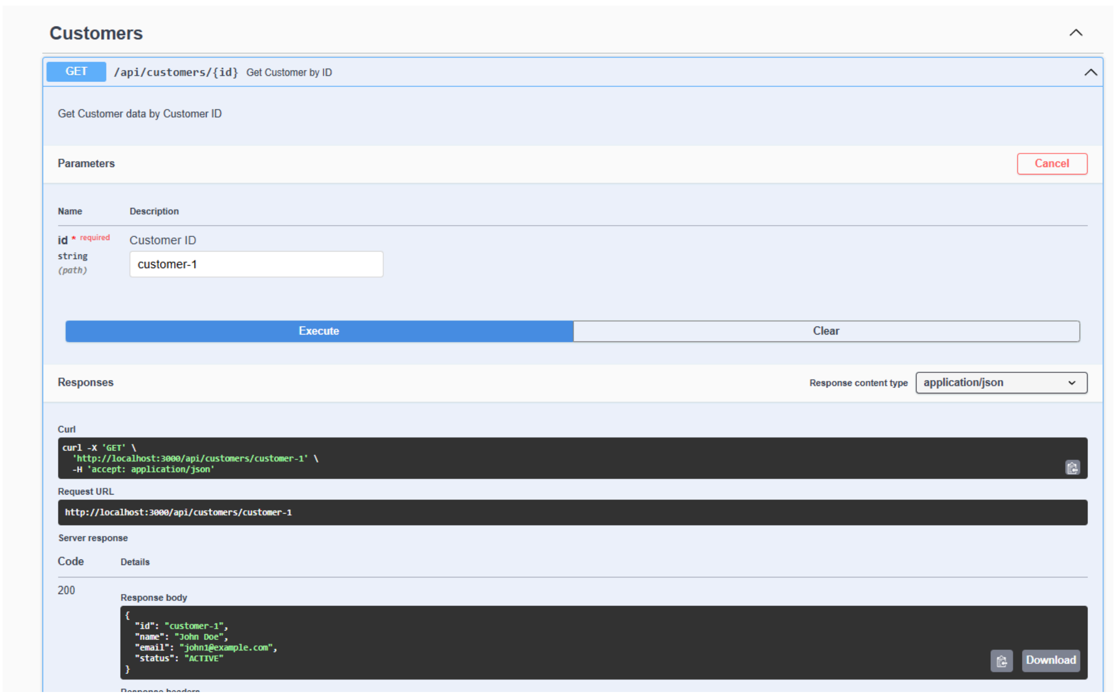
ACCEPTANCE TEST

1. Order created successfully

We validate a customer and a product through the Go API before sending the order to Kafka. http://localhost:3000/swagger/index.html



Customers

GET `/api/customers/{id}` Get Customer by ID

Get Customer data by Customer ID

Parameters

Name	Description
id * required string (path)	Customer ID: customer-1

Responses — Response content type: application/json

Curl

```
curl -X 'GET' \
  'http://localhost:3000/api/customers/customer-1' \
  -H 'accept: application/json'
```

Request URL

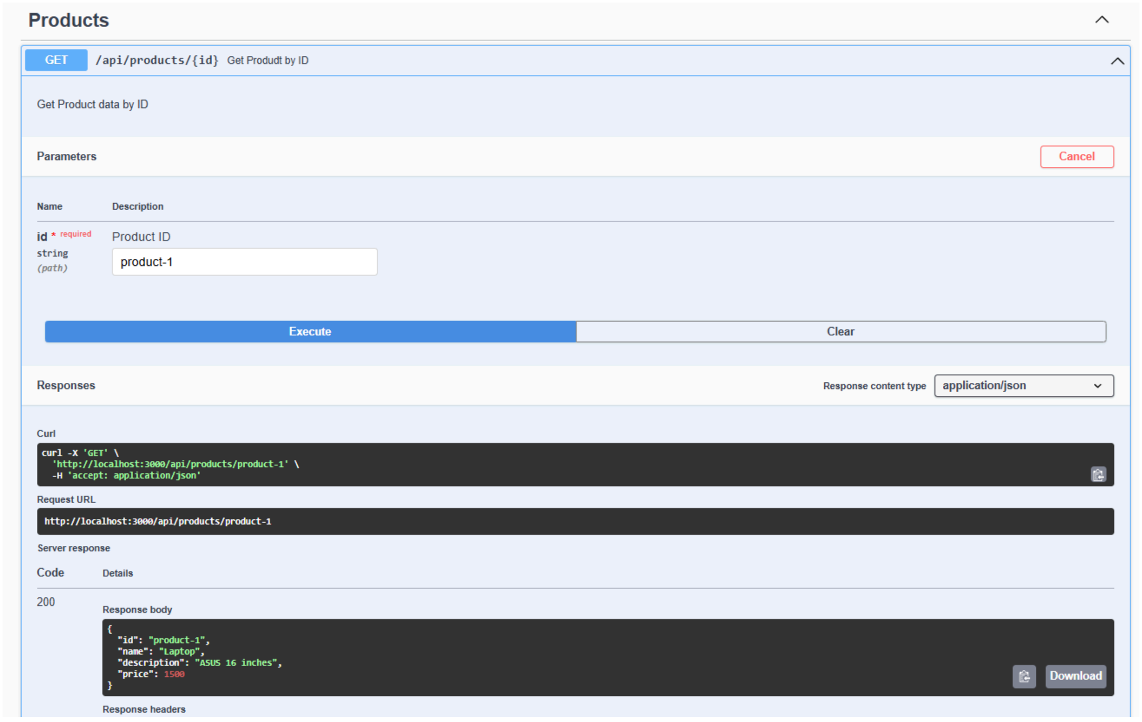
```
http://localhost:3000/api/customers/customer-1
```

Server response

Code: 200

Response body

```
{
  "id": "customer-1",
  "name": "John Doe",
  "email": "john1@example.com",
  "status": "ACTIVE"
}
```



Products

GET `/api/products/{id}` Get Produdt by ID

Get Product data by ID

Parameters

Name	Description
id * required string (path)	Product ID: product-1

Responses — Response content type: application/json

Curl

```
curl -X 'GET' \
  'http://localhost:3000/api/products/product-1' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:3000/api/products/product-1
```

Server response

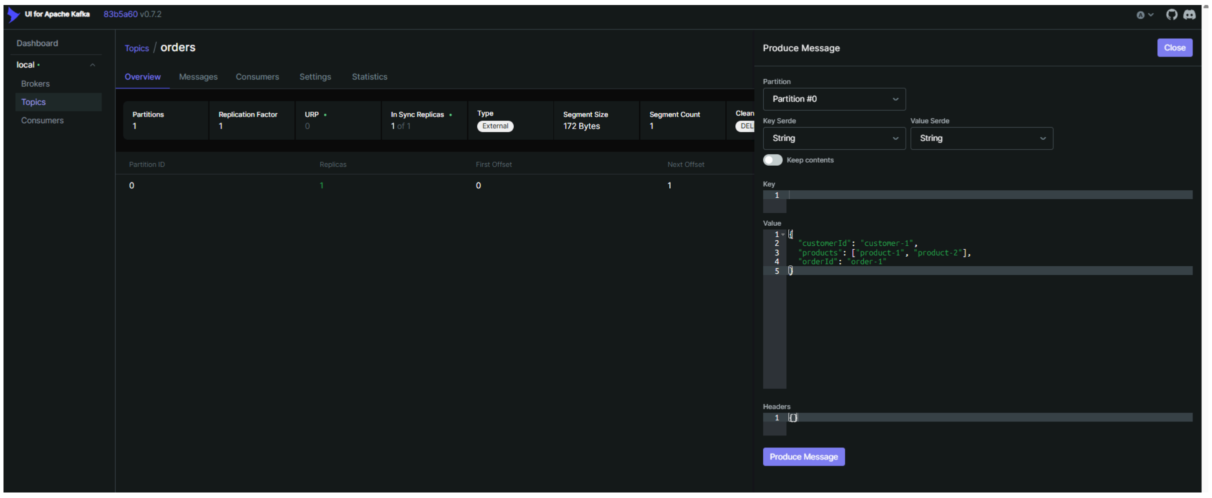
Code: 200

Response body

```
{
  "id": "product-1",
  "name": "Laptop",
  "description": "ASUS 16 inches",
  "price": 1500
}
```

Response headers

We access the Kafka UI at http://localhost:8081 and send a message through the created topic, as shown below.



Partitions	Replication Factor	URP	In Sync Replicas	Type	Segment Size	Segment Count
1	1	0	1 of 1	External	172 Bytes	1

Partition ID	Replicas	First Offset	Next Offset
0	1	0	1

Produce Message

Partition: Partition #0

Key Serde: String

Value Serde: String

Keep contents

Key

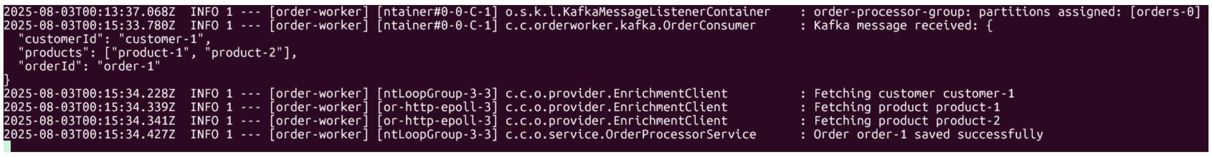
Value

```
{
  "customerId": "customer-1",
  "products": ["product-1", "product-2"],
  "orderId": "order-1"
}
```

Headers

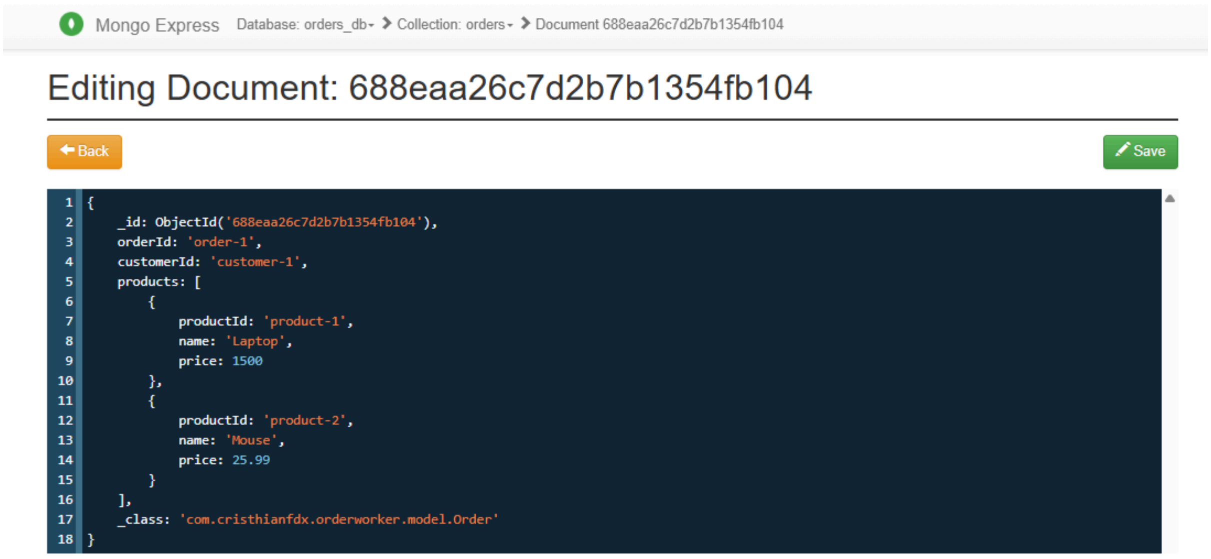
```
{}
```

We check the logs of the Java app using the command ***`docker logs -f java-api`*** and confirm that the order was created successfully



```
2025-08-03T00:13:37.068Z  INFO 1 --- [order-worker] [ntainer#0-0-C-1] o.s.k.l.KafkaMessageListenerContainer    : order-processor-group: partitions assigned: [orders-0]
2025-08-03T00:15:33.780Z  INFO 1 --- [order-worker] [ntainer#0-0-C-1] c.c.orderworker.kafka.OrderConsumer      : Kafka message received: {
  "customerId": "customer-1",
  "products": ["product-1", "product-2"],
  "orderId": "order-1"
}
2025-08-03T00:15:34.228Z  INFO 1 --- [order-worker] [ntLoopGroup-3-3] c.c.o.provider.EnrichmentClient          : Fetching customer customer-1
2025-08-03T00:15:34.339Z  INFO 1 --- [order-worker] [or-http-epoll-3] c.c.o.provider.EnrichmentClient          : Fetching product product-1
2025-08-03T00:15:34.341Z  INFO 1 --- [order-worker] [or-http-epoll-3] c.c.o.provider.EnrichmentClient          : Fetching product product-2
2025-08-03T00:15:34.427Z  INFO 1 --- [order-worker] [ntLoopGroup-3-3] c.c.o.service.OrderProcessorService      : Order order-1 saved successfully
```

We access the Mongo Express UI at http://localhost:8082 and look for the ***`orders_db`*** database, where we can see the created record

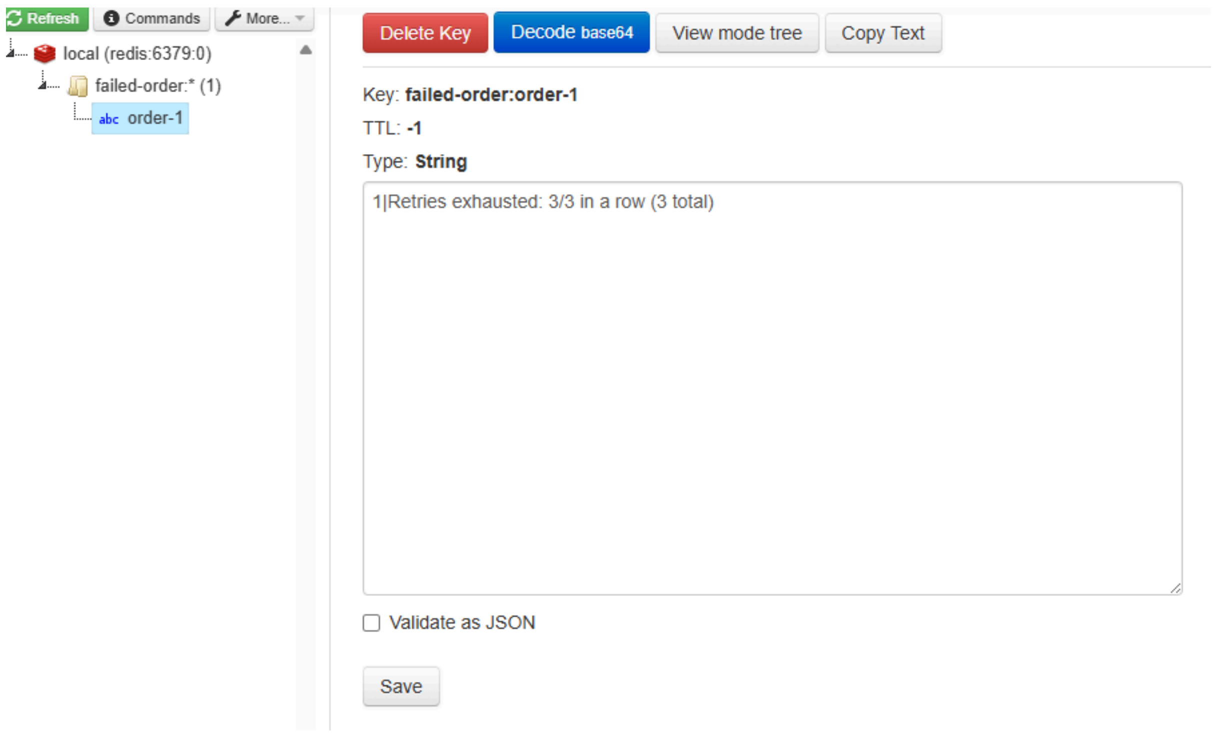


Editing Document: 688eaa26c7d2b7b1354fb104

```
{
    _id: ObjectId('688eaa26c7d2b7b1354fb104'),
    orderId: 'order-1',
    customerId: 'customer-1',
    products: [
        {
            productId: 'product-1',
            name: 'Laptop',
            price: 1500
        },
        {
            productId: 'product-2',
            name: 'Mouse',
            price: 25.99
        }
    ],
    _class: 'com.cristhianfdx.orderworker.model.Order'
}
```

2. Testing retries and logging in Redis

If the Go API is down, the system attempts retries but they fail — it has a maximum of three attempts.
Since the order could not be created, the error is logged in Redis.
You can visualize it by accessing Redis Commander at: http://localhost:8083



Key: **failed-order:order-1**
TTL: **-1**
Type: **String**

```
1|Retries exhausted: 3/3 in a row (3 total)
```