Pubsub-order : Simple Order REST Service

Introduction

This document is created for detailed demonstration of **pubsub-order** REST service with screenshots.
The service contains three endpoints:

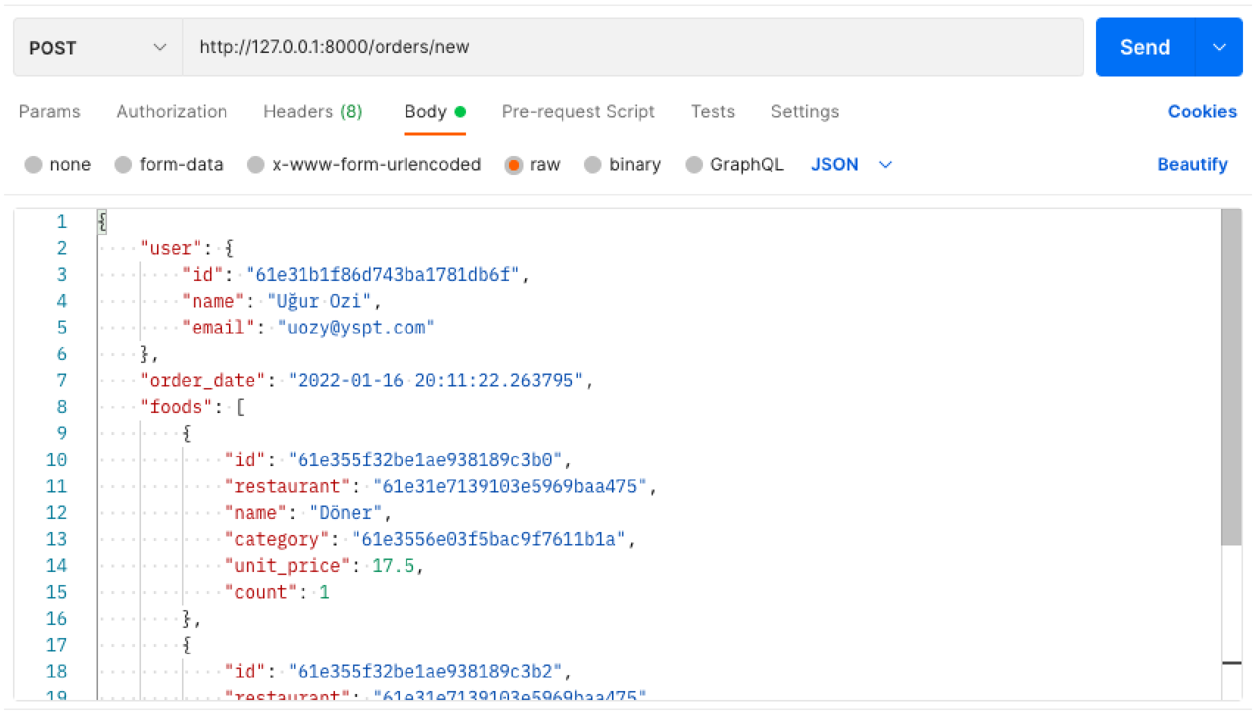
Endpoint	Method	Consumes	Produces	Info
/orders?t=<order_type>	GET	-	JSON	This endpoint is using for getting orders in the system. “t” is the parameter for order type: 0 is all orders 1 is completed orders -1 is incomplete orders It returns the orders in the system as JSON.
/orders/new	POST	JSON	JSON	This endpoint is using for creating new orders in the system. It accepts Order-type object JSON and it returns the Amazon SQS’s insert message.
/orders/complete	POST	-	JSON	This endpoint is using for completing orders (i.e removes an order from the queue and inserts to the orders table) It returns simple success message.

Simple Scenario

Create a new order in the system.

In order to create a new order in the system, we are using /orders/new endpoint with POST method.

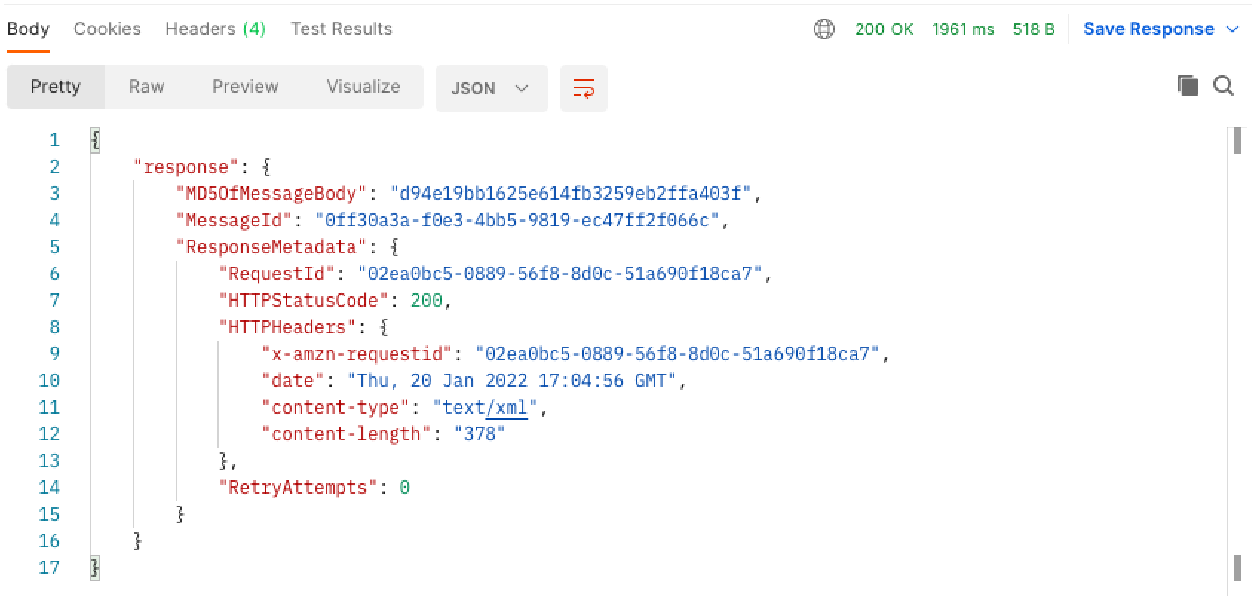
Request



```
http://127.0.0.1:8000/orders/new
```

```
{
    "user": {
        "id": "61e31b1f86d743ba1781db6f",
        "name": "Uğur Ozi",
        "email": "uozy@yspt.com"
    },
    "order_date": "2022-01-16 20:11:22.263795",
    "foods": [
        {
            "id": "61e355f32be1ae938189c3b0",
            "restaurant": "61e31e7139103e5969baa475",
            "name": "Döner",
            "category": "61e3556e03f5bac9f7611b1a",
            "unit_price": 17.5,
            "count": 1
        },
        {
            "id": "61e355f32be1ae938189c3b2",
            "restaurant": "61e31e7139103e5969baa475"
```

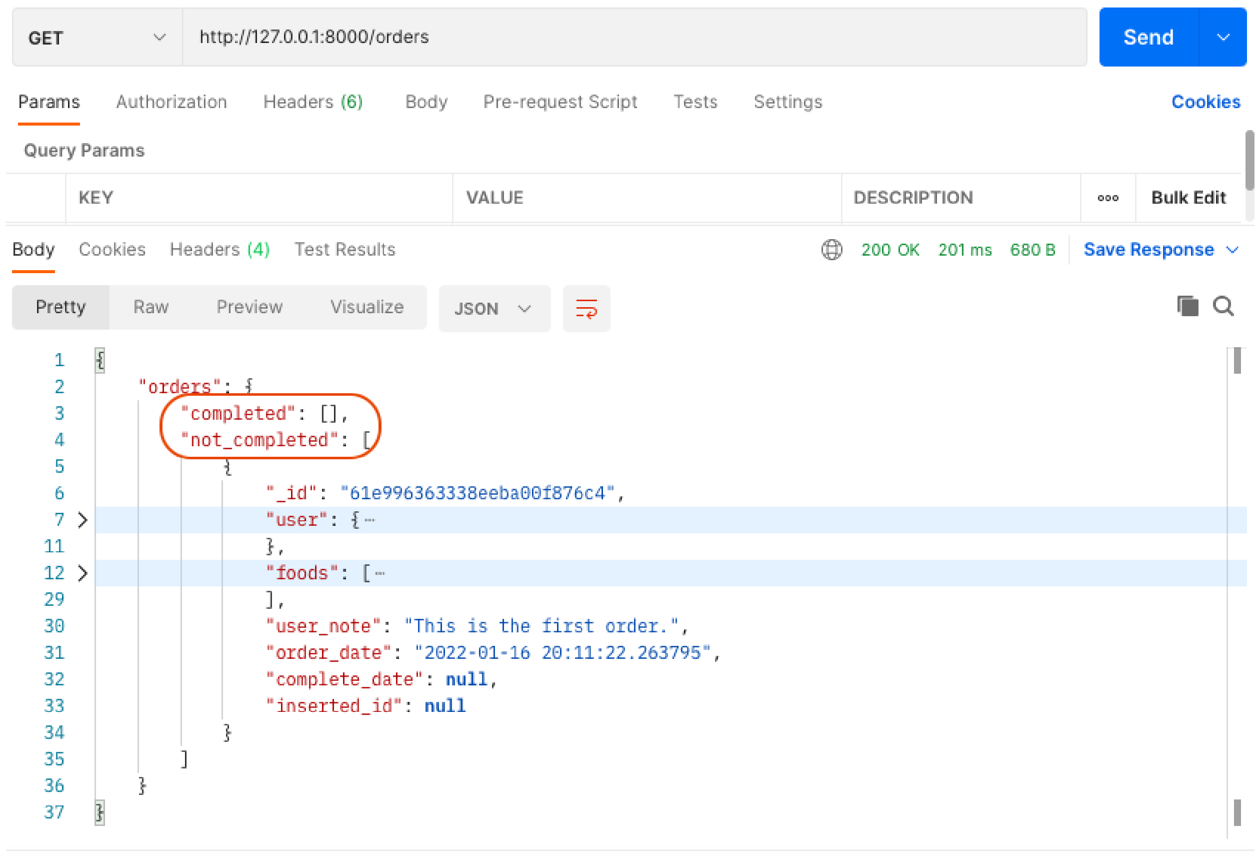
Response



```
{
    "response": {
        "MD5OfMessageBody": "d94e19bb1625e614fb3259eb2ffa403f",
        "MessageId": "0ff30a3a-f0e3-4bb5-9819-ec47ff2f066c",
        "ResponseMetadata": {
            "RequestId": "02ea0bc5-0889-56f8-8d0c-51a690f18ca7",
            "HTTPStatusCode": 200,
            "HTTPHeaders": {
                "x-amzn-requestid": "02ea0bc5-0889-56f8-8d0c-51a690f18ca7",
                "date": "Thu, 20 Jan 2022 17:04:56 GMT",
                "content-type": "text/xml",
                "content-length": "378"
            },
            "RetryAttempts": 0
        }
    }
}
```

List all orders.

Now, let’s see whether the latter order is placed in the system or not.



```
http://127.0.0.1:8000/orders
```

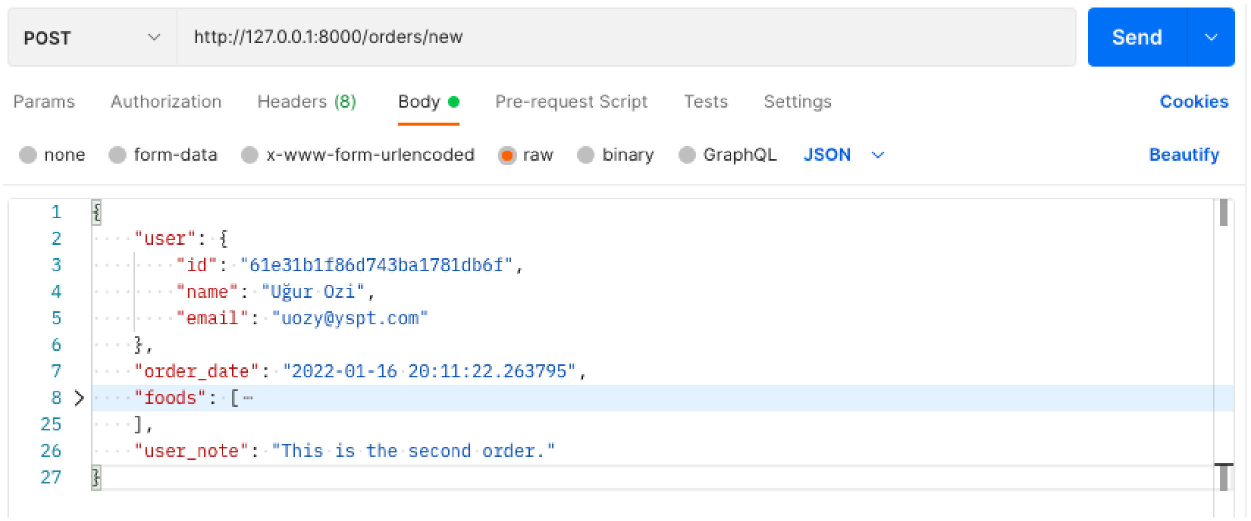
```
{
    "orders": {
        "completed": [],
        "not_completed": [
            {
                "_id": "61e996363338eeba00f876c4",
                "user": {⋯
                },
                "foods": [⋯
                ],
                "user_note": "This is the first order.",
                "order_date": "2022-01-16 20:11:22.263795",
                "complete_date": null,
                "inserted_id": null
            }
        ]
    }
}
```

As we can see from the image above; the first order placed in the orders; under the “not_completed” title. After the completion, we expect to see under the “completed” title.

Create another order.

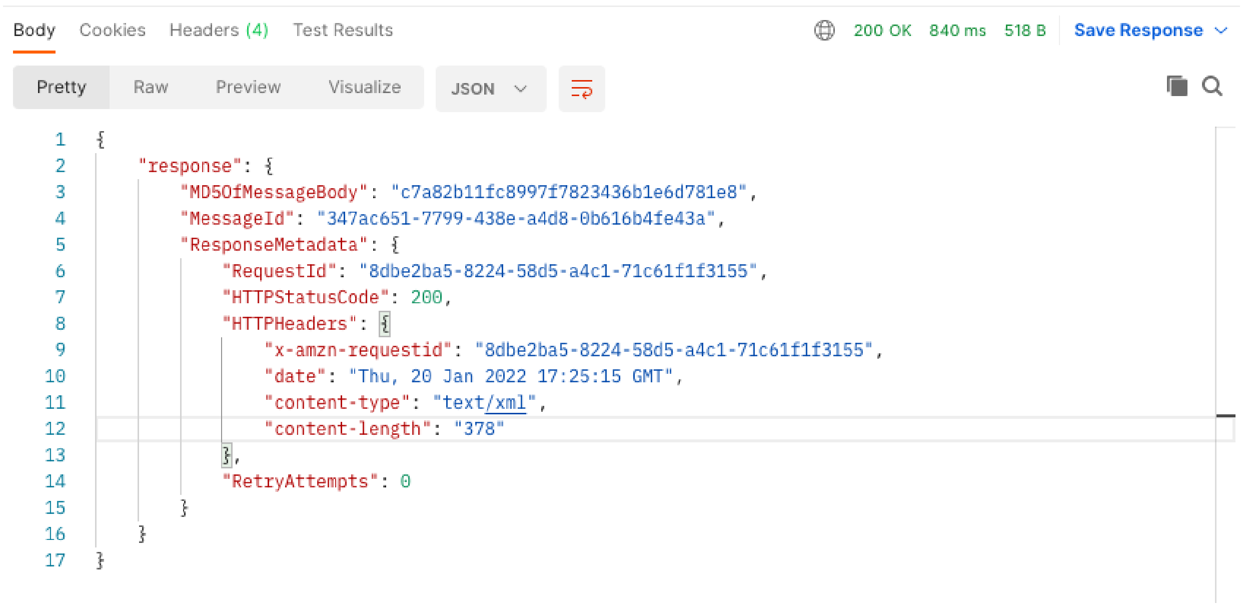
Let’s create another order. The system should that order in the system before the first one is completed.

Request


POST http://127.0.0.1:8000/orders/new

```
{
    "user": {
        "id": "61e31b1f86d743ba1781db6f",
        "name": "Uğur Ozi",
        "email": "uozy@yspt.com"
    },
    "order_date": "2022-01-16 20:11:22.263795",
    "foods": [⋯
    ],
    "user_note": "This is the second order."
}
```

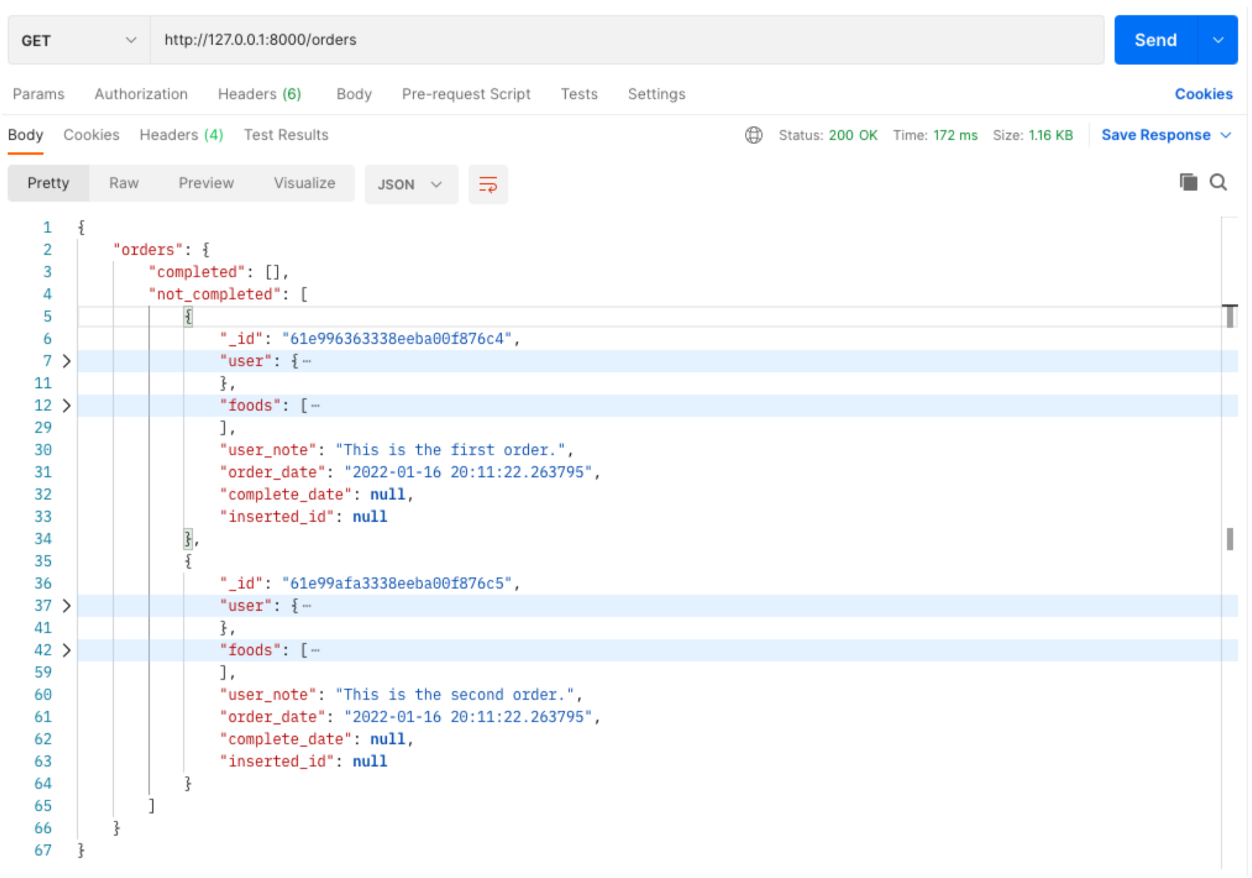
Response


200 OK 840 ms 518 B

```
{
    "response": {
        "MD5OfMessageBody": "c7a82b11fc8997f7823436b1e6d781e8",
        "MessageId": "347ac651-7799-438e-a4d8-0b616b4fe43a",
        "ResponseMetadata": {
            "RequestId": "8dbe2ba5-8224-58d5-a4c1-71c61f1f3155",
            "HTTPStatusCode": 200,
            "HTTPHeaders": {
                "x-amzn-requestid": "8dbe2ba5-8224-58d5-a4c1-71c61f1f3155",
                "date": "Thu, 20 Jan 2022 17:25:15 GMT",
                "content-type": "text/xml",
                "content-length": "378"
            },
            "RetryAttempts": 0
        }
    }
}
```

List all orders.

Now, let’s see the latest state of the orders in the system.



http://127.0.0.1:8000/orders

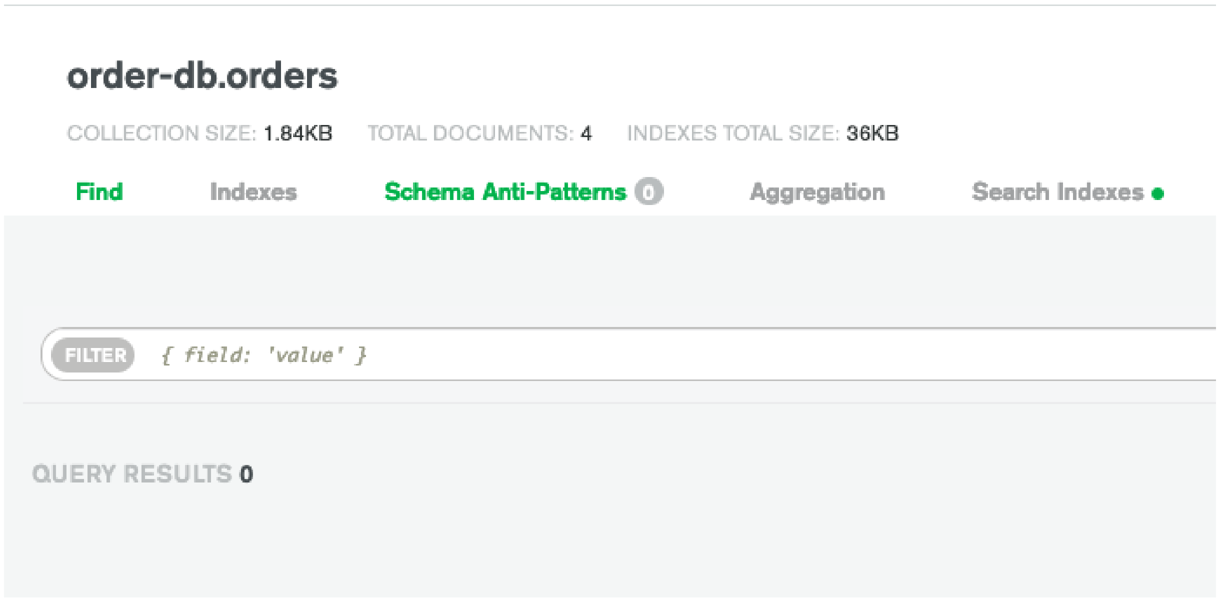
Status: 200 OK Time: 172 ms Size: 1.16 KB

```
{
    "orders": {
        "completed": [],
        "not_completed": [
            {
                "_id": "61e996363338eeba00f876c4",
                "user": {⋯
                },
                "foods": [⋯
                ],
                "user_note": "This is the first order.",
                "order_date": "2022-01-16 20:11:22.263795",
                "complete_date": null,
                "inserted_id": null
            },
            {
                "_id": "61e99afa3338eeba00f876c5",
                "user": {⋯
                },
                "foods": [⋯
                ],
                "user_note": "This is the second order.",
                "order_date": "2022-01-16 20:11:22.263795",
                "complete_date": null,
                "inserted_id": null
            }
        ]
    }
}
```

As we can see, the second one is placed in “not_completed”, too.

After the second order, the latest status in the collections in order database is as following:

- Order-db.orders

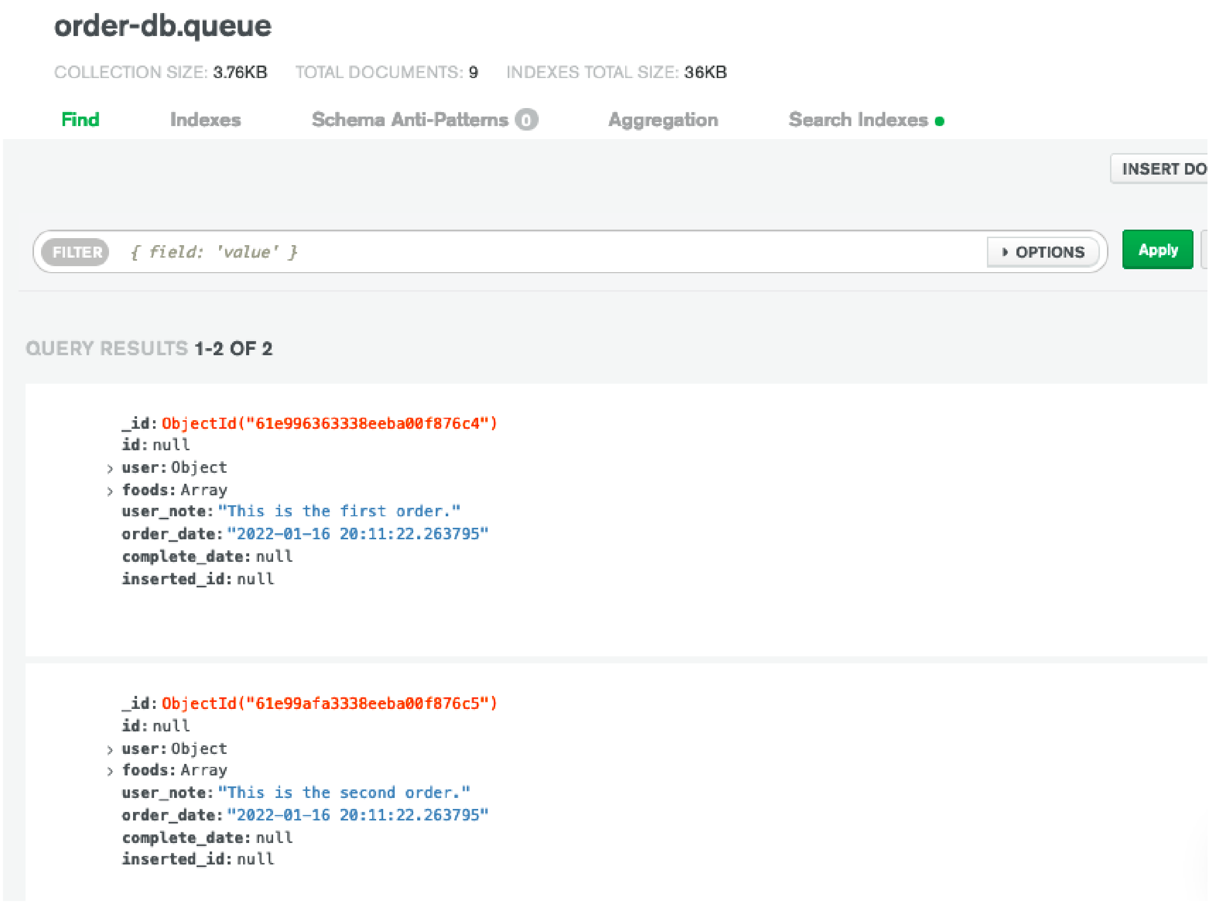


order-db.orders

COLLECTION SIZE: 1.84KB TOTAL DOCUMENTS: 4 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS 0

- Order-db.queue



order-db.queue

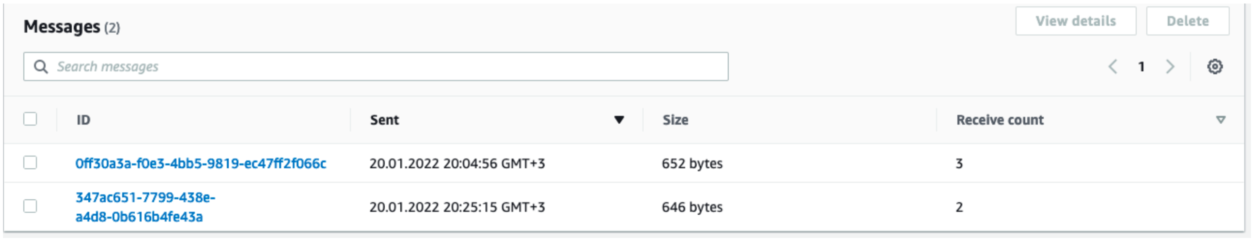
COLLECTION SIZE: 3.76KB TOTAL DOCUMENTS: 9 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS 1-2 OF 2

```
_id: ObjectId("61e996363338eeba00f876c4")
id: null
> user: Object
> foods: Array
user_note: "This is the first order."
order_date: "2022-01-16 20:11:22.263795"
complete_date: null
inserted_id: null
```

```
_id: ObjectId("61e99afa3338eeba00f876c5")
id: null
> user: Object
> foods: Array
user_note: "This is the second order."
order_date: "2022-01-16 20:11:22.263795"
complete_date: null
inserted_id: null
```

And our queue (Amazon SQS) is looking like:



Messages (2)

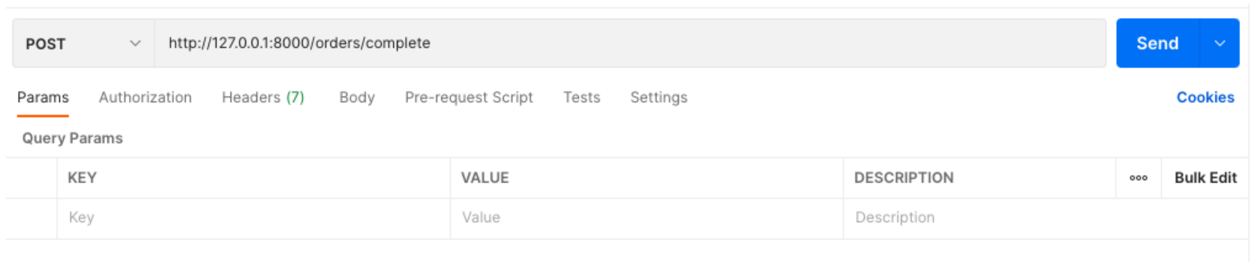
ID	Sent	Size	Receive count
0ff30a3a-f0e3-4bb5-9819-ec47ff2f066c	20.01.2022 20:04:56 GMT+3	652 bytes	3
347ac651-7799-438e-a4d8-0b616b4fe43a	20.01.2022 20:25:15 GMT+3	646 bytes	2

Now, we will try to complete an order. Our queue is designed like FIFO (first-in first-out) logic, so we are expecting that the first one (i.e user_note = “This is the first order.”) will be completed.

Complete an order.

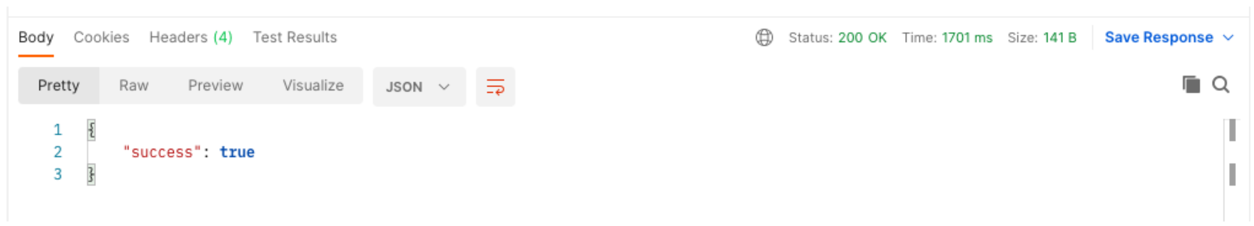
In order to complete an order, we are using /orders/complete endpoint with POST method.

Request



```
http://127.0.0.1:8000/orders/complete
```

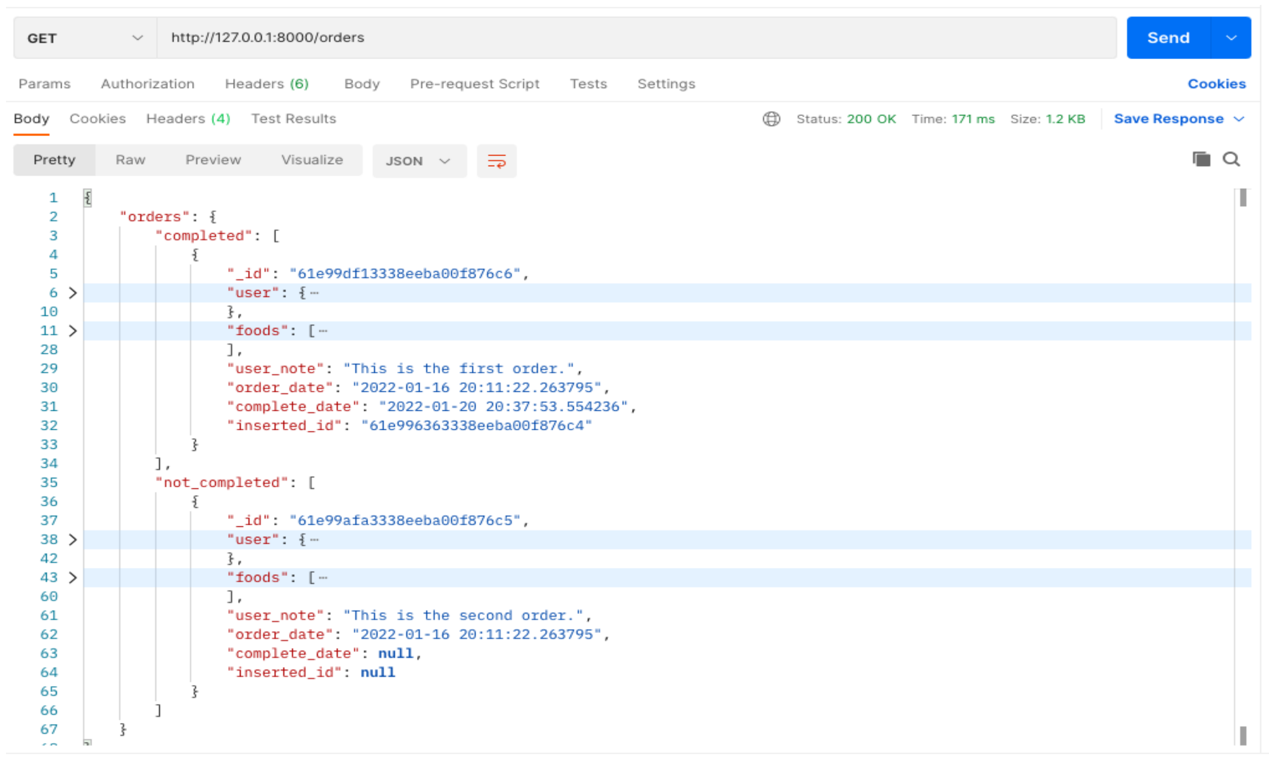
Response



Status: 200 OK Time: 1701 ms Size: 141 B

```
{
    "success": true
}
```

List all orders



```
http://127.0.0.1:8000/orders
```

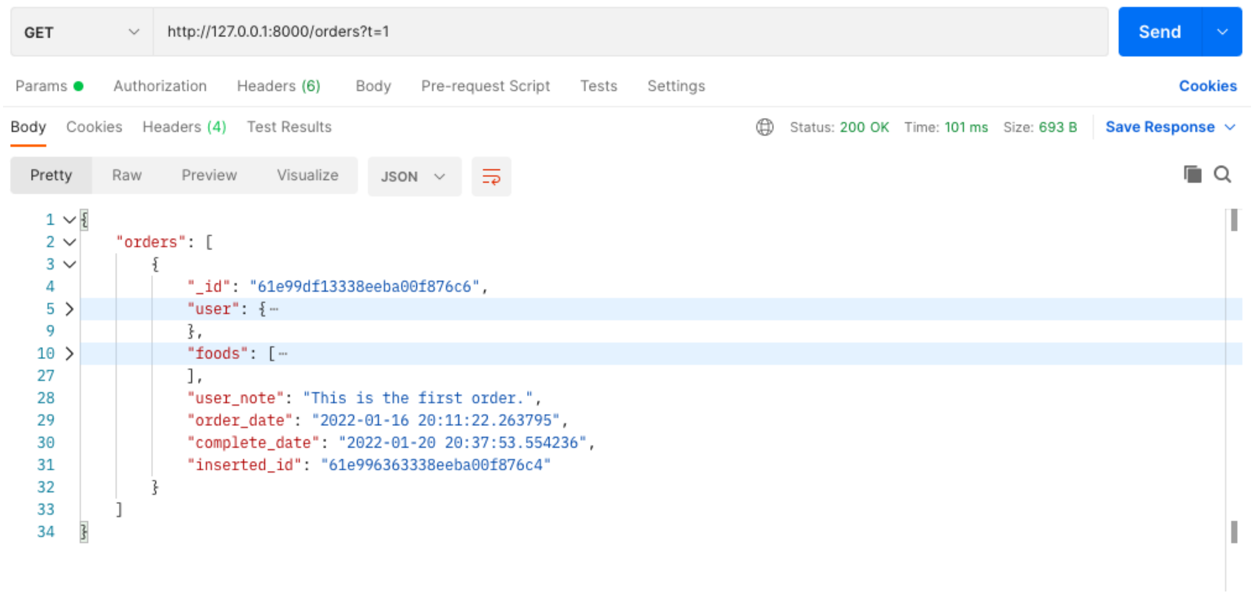
Status: 200 OK Time: 171 ms Size: 1.2 KB

```
{
    "orders": {
        "completed": [
            {
                "_id": "61e99df13338eeba00f876c6",
                "user": {⋯
                },
                "foods": [⋯
                ],
                "user_note": "This is the first order.",
                "order_date": "2022-01-16 20:11:22.263795",
                "complete_date": "2022-01-20 20:37:53.554236",
                "inserted_id": "61e996363338eeba00f876c4"
            }
        ],
        "not_completed": [
            {
                "_id": "61e99afa3338eeba00f876c5",
                "user": {⋯
                },
                "foods": [⋯
                ],
                "user_note": "This is the second order.",
                "order_date": "2022-01-16 20:11:22.263795",
                "complete_date": null,
                "inserted_id": null
            }
        ]
    }
```

So, as we expected, one of the orders in the queue is placed in “completed” title.

List completed orders

Let's test our GET methods' query parameter. For listing completed orders, we can use /orders?t=1 endpoint with GET method.

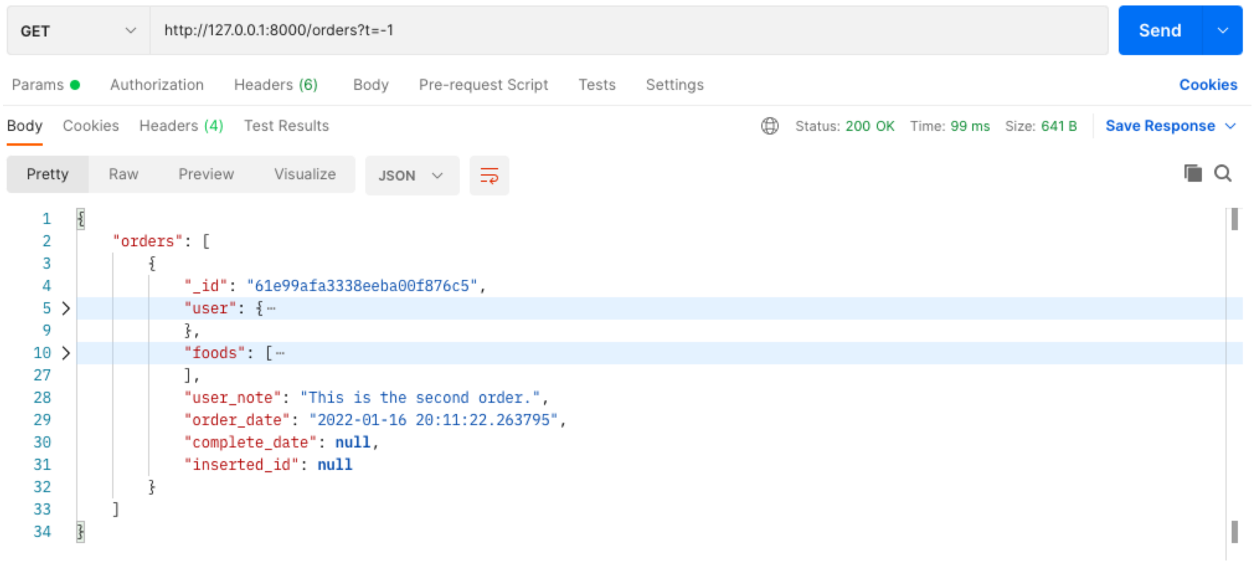


```
{
    "orders": [
        {
            "_id": "61e99df13338eeba00f876c6",
            "user": {⋯
            },
            "foods": [⋯
            ],
            "user_note": "This is the first order.",
            "order_date": "2022-01-16 20:11:22.263795",
            "complete_date": "2022-01-20 20:37:53.554236",
            "inserted_id": "61e996363338eeba00f876c4"
        }
    ]
}
```

Yes, the first one was completed.

List waiting orders

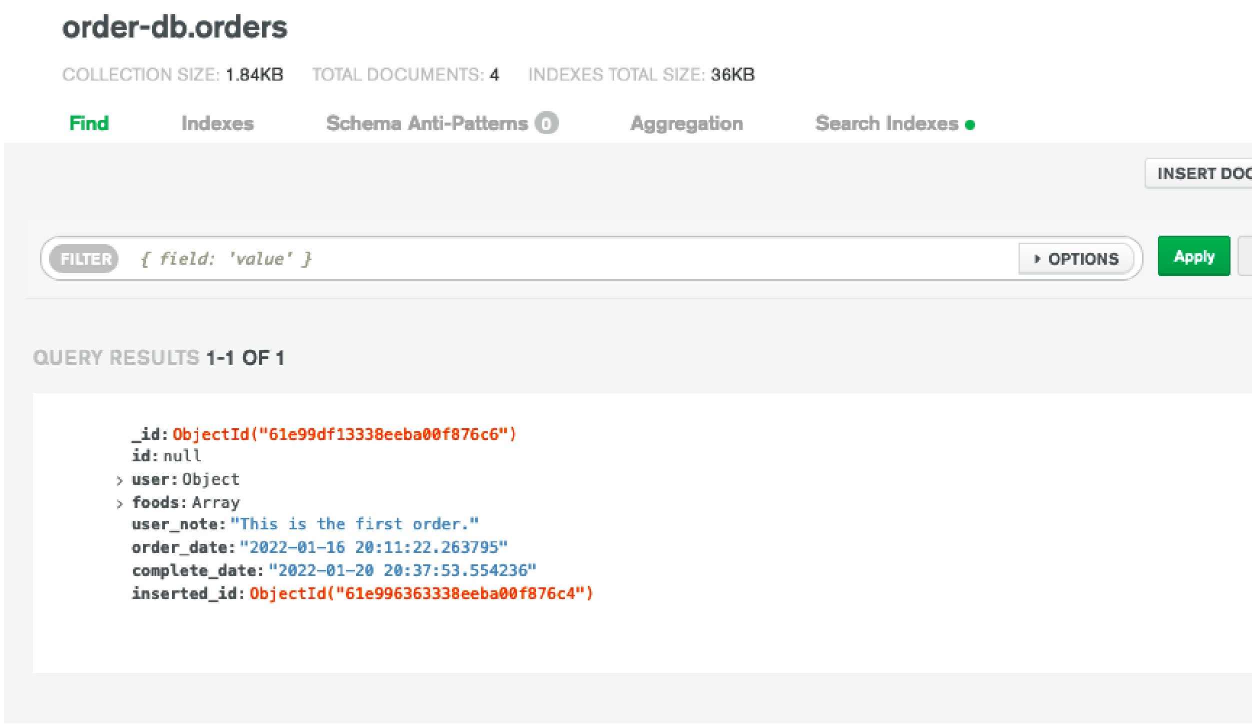
And finally, let's list completed orders. For that operation, we can use /orders?t=-1 endpoint with GET method.



```
{
    "orders": [
        {
            "_id": "61e99afa3338eeba00f876c5",
            "user": {⋯
            },
            "foods": [⋯
            ],
            "user_note": "This is the second order.",
            "order_date": "2022-01-16 20:11:22.263795",
            "complete_date": null,
            "inserted_id": null
        }
    ]
}
```

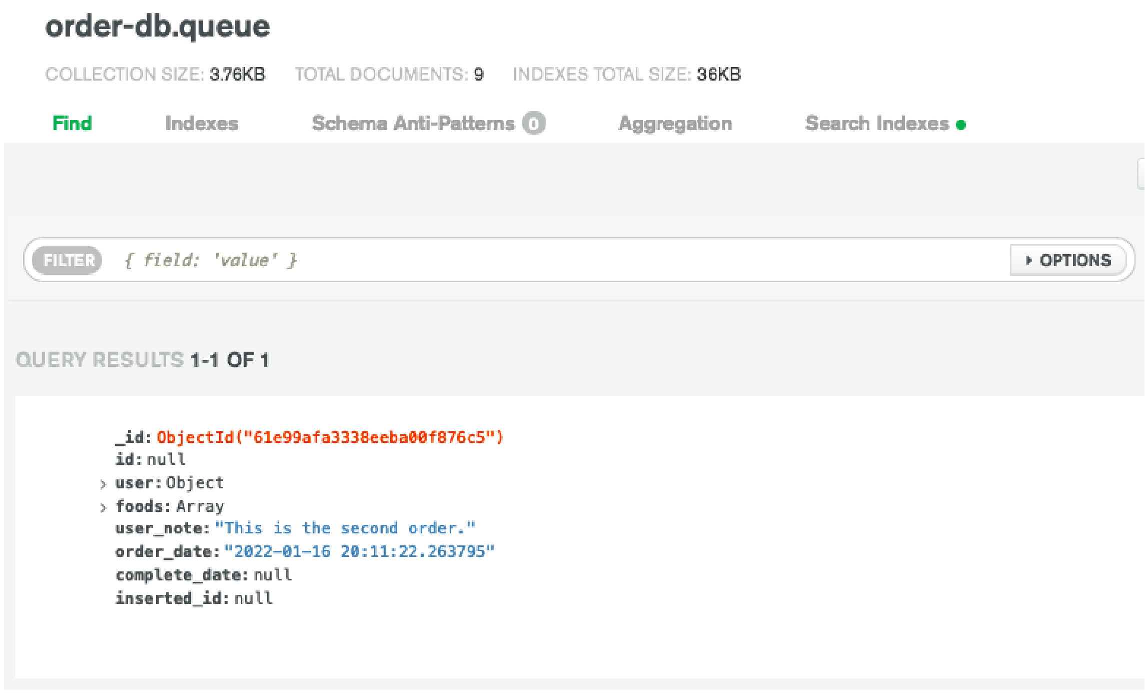
And, finally, let's see the database side:
The final status of the database is as following:

- Order-db.orders



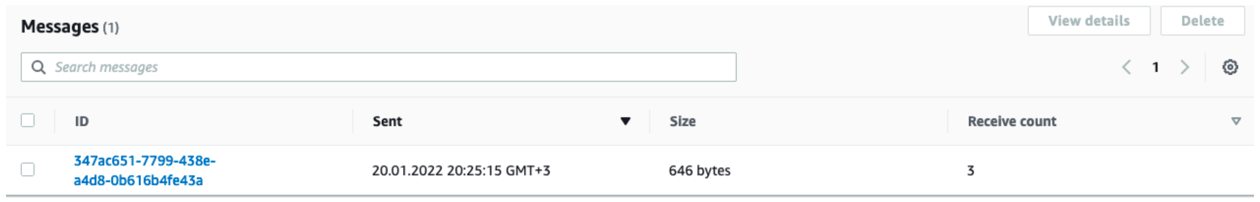
```
_id: ObjectId("61e99df13338eeba00f876c6")
id: null
> user: Object
> foods: Array
user_note: "This is the first order."
order_date: "2022-01-16 20:11:22.263795"
complete_date: "2022-01-20 20:37:53.554236"
inserted_id: ObjectId("61e996363338eeba00f876c4")
```

- Order-db.queue



```
_id: ObjectId("61e99afa3338eeba00f876c5")
id: null
> user: Object
> foods: Array
user_note: "This is the second order."
order_date: "2022-01-16 20:11:22.263795"
complete_date: null
inserted_id: null
```

And now, the queue is containing only one order:


Messages (1)

ID	Sent	Size	Receive count
347ac651-7799-438e-a4d8-0b616b4fe43a	20.01.2022 20:25:15 GMT+3	646 bytes	3

So, basically:

- The new order placed in the queue after the request to our API.
- When the user sends a request for completing an order, our service gets an order from the queue and completes the order.
- And we can list all orders, just completed ones and the incomplete ones.

For that project, we used following tools:

- Web service: Fast API
- Database: MongoDB Atlas
- Queue: Amazon Simple Queue Service (Amazon SQS)

You can access the full repo from this link.