

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	<p>This endpoint is using for getting orders in the system. "t" is the parameter for order type:</p> <p>0 is all orders 1 is completed orders -1 is incomplete orders</p> <p>It returns the orders in the system as JSON.</p>
/orders/new	POST	JSON	JSON	<p>This endpoint is using for creating new orders in the system.</p> <p>It accepts Order-type object JSON and it returns the Amazon SQS's insert message.</p>
/orders/complete	POST	-	JSON	<p>This endpoint is using for completing orders (i.e removes an order from the queue and inserts to the orders table)</p> <p>It returns simple success message.</p>

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

POST

http://127.0.0.1:8000/orders/new

Send

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettingsCookies

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

Beautify

```
1  {
2    "user": {
3      "id": "61e31b1f86d743ba1781db6f",
4      "name": "Uğur Ozi",
5      "email": "uozy@spt.com"
6    },
7    "order_date": "2022-01-16 20:11:22.263795",
8    "foods": [
9      {
10       "id": "61e355f32be1ae938189c3b0",
11       "restaurant": "61e31e7139103e5969baa475",
12       "name": "Döner",
13       "category": "61e3556e03f5bac9f7611b1a",
14       "unit_price": 17.5,
15       "count": 1
16     },
17     {
18       "id": "61e355f32be1ae938189c3b2",
19       "restaurant": "61e31e7139103e5969baa475"
```

Response

BodyCookiesHeaders (4)Test Results200 OK1961 ms518 BSave Response

PrettyRawPreviewVisualizeJSON

```
1  {
2    "response": {
3      "MD5OfMessageBody": "d94e19bb1625e614fb3259eb2ffa403f",
4      "MessageId": "0ff30a3a-f0e3-4bb5-9819-ec47ff2f066c",
5      "ResponseMetadata": {
6        "RequestId": "02ea0bc5-0889-56f8-8d0c-51a690f18ca7",
7        "HTTPStatusCode": 200,
8        "HTTPHeaders": {
9          "x-amzn-requestid": "02ea0bc5-0889-56f8-8d0c-51a690f18ca7",
10         "date": "Thu, 20 Jan 2022 17:04:56 GMT",
11         "content-type": "text/xml",
12         "content-length": "378"
13       },
14       "RetryAttempts": 0
15     }
16   }
17 }
```

List all orders.

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

GET http://127.0.0.1:8000/orders

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
-----	-------	-------------	-----	-----------

Body Cookies Headers (4) Test Results 200 OK 201 ms 680 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "orders": {
3     "completed": [],
4     "not_completed": [
5       {
6         "_id": "61e996363338eeba00f876c4",
7         "user": { ...
11      },
12      "foods": [ ...
29    ],
30    "user_note": "This is the first order.",
31    "order_date": "2022-01-16 20:11:22.263795",
32    "complete_date": null,
33    "inserted_id": null
34      }
35    ]
36  }
37 }
```

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

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {
2   ... "user": {
3     ... "id": "61e31b1f86d743ba1781db6f",
4     ... "name": "Uğur Ozi",
5     ... "email": "uozy@yspt.com"
6   },
7   ... "order_date": "2022-01-16 20:11:22.263795",
8 >  ... "foods": [ ...
25  ... ],
26  ... "user_note": "This is the second order."
27 }
```

Response

Body

Cookies

Headers (4)

Test Results

200 OK 840 ms 518 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "response": {
3     "MD5OfMessageBody": "c7a82b11fc8997f7823436b1e6d781e8",
4     "MessageId": "347ac651-7799-438e-a4d8-0b616b4fe43a",
5     "ResponseMetadata": {
6       "RequestId": "8dbe2ba5-8224-58d5-a4c1-71c61f1f3155",
7       "HTTPStatusCode": 200,
8       "HTTPHeaders": {
9         "x-amzn-requestid": "8dbe2ba5-8224-58d5-a4c1-71c61f1f3155",
10        "date": "Thu, 20 Jan 2022 17:25:15 GMT",
11        "content-type": "text/xml",
12        "content-length": "378"
13      },
14      "RetryAttempts": 0
15    }
16  }
17 }
```

List all orders.

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

GET http://127.0.0.1:8000/orders Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (4) Test Results Status: 200 OK Time: 172 ms Size: 1.16 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "orders": {
3     "completed": [],
4     "not_completed": [
5       {
6         "_id": "61e996363338eeba00f876c4",
7         "user": {
11        },
12        "foods": [
29        ],
30        "user_note": "This is the first order.",
31        "order_date": "2022-01-16 20:11:22.263795",
32        "complete_date": null,
33        "inserted_id": null
34      },
35      {
36        "_id": "61e99afa3338eeba00f876c5",
37        "user": {
41        },
42        "foods": [
59        ],
60        "user_note": "This is the second order.",
61        "order_date": "2022-01-16 20:11:22.263795",
62        "complete_date": null,
63        "inserted_id": null
64      }
65    ]
66  }
67 }
```

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

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes ●

FILTER { field: 'value' }

QUERY RESULTS 0

- Order-db.queue

order-db.queue

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

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes ●

INSERT DO

FILTER { field: 'value' } OPTIONS Apply

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)					View details	Delete
<input type="text" value="Search messages"/>					< 1 >	⚙
<input type="checkbox"/>	ID	Sent	Size	Receive count		
<input type="checkbox"/>	0ff30a3a-f0e3-4bb5-9819-ec47ff2f066c	20.01.2022 20:04:56 GMT+3	652 bytes	3		
<input type="checkbox"/>	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

POST	http://127.0.0.1:8000/orders/complete	Send					
Params	Authorization	Headers (7)	Body	Pre-request Script	Tests	Settings	Cookies
Query Params							
	KEY	VALUE	DESCRIPTION	...	Bulk Edit		
	Key	Value	Description				

Response

Body	Cookies	Headers (4)	Test Results	Status: 200 OK	Time: 1701 ms	Size: 141 B	Save Response
Pretty	Raw	Preview	Visualize	JSON			
<pre>1 { 2 "success": true 3 }</pre>							

List all orders

GET	http://127.0.0.1:8000/orders	Send					
Params	Authorization	Headers (6)	Body	Pre-request Script	Tests	Settings	Cookies
Body	Cookies	Headers (4)	Test Results	Status: 200 OK	Time: 171 ms	Size: 1.2 KB	Save Response
Pretty	Raw	Preview	Visualize	JSON			
<pre>1 { 2 "orders": { 3 "completed": [4 { 5 "_id": "61e99df13338eeba00f876c6", 6 "user": { 10 }, 11 "foods": [28], 29 "user_note": "This is the first order.", 30 "order_date": "2022-01-16 20:11:22.263795", 31 "complete_date": "2022-01-20 20:37:53.554236", 32 "inserted_id": "61e996363338eeba00f876c4" 33 }, 34 "not_completed": [35 { 36 "_id": "61e99afa3338eeba00f876c5", 37 "user": { 38 }, 42 }, 43 "foods": [60], 61 "user_note": "This is the second order.", 62 "order_date": "2022-01-16 20:11:22.263795", 63 "complete_date": null, 64 "inserted_id": null 65 } 66] 67 } 68 }</pre>							

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.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/orders?t=1
- Status:** 200 OK
- Time:** 101 ms
- Size:** 693 B

The response body is displayed in JSON format:

```
{
  "orders": [
    {
      "_id": "61e99df13338eeba00f876c6",
      "user": {
        "name": "John Doe",
        "email": "john.doe@example.com",
        "password": "12345678"
      },
      "foods": [
        {
          "name": "Burger",
          "quantity": 1
        },
        {
          "name": "Fries",
          "quantity": 1
        }
      ],
      "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.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/orders?t=-1
- Status:** 200 OK
- Time:** 99 ms
- Size:** 641 B

The response body is displayed in JSON format:

```
{
  "orders": [
    {
      "_id": "61e99afa3338eeba00f876c5",
      "user": {
        "name": "John Doe",
        "email": "john.doe@example.com",
        "password": "12345678"
      },
      "foods": [
        {
          "name": "Burger",
          "quantity": 1
        },
        {
          "name": "Fries",
          "quantity": 1
        }
      ],
      "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

order-db.orders

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

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes ●

INSERT DOCUMENT

FILTER { field: 'value' } **OPTIONS** **Apply**

QUERY RESULTS 1-1 OF 1

```
_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

order-db.queue

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

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes ●

FILTER { field: 'value' } **OPTIONS**

QUERY RESULTS 1-1 OF 1

```
_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)					View details	Delete
<input type="text" value="Search messages"/>					< 1 > ⚙	
<input type="checkbox"/>	ID	Sent	Size	Receive count		
<input type="checkbox"/>	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](#).