

Осуществлена реализация 3-х микросервисов, а также реализация сервиса notification (rabbitmq).

Для каждого микросервиса подключается своя база данных:

- postgresql для productService, storeroomService;
- mongodb для orderService.



Также есть реализация брокера сообщений rabbitmq.



Есть связь по gRPC между orderService и storeroomService.



Каждый микросервис, а также база данных, брокер сообщений, gRPC работают на своих портах:

- productService 18080, postgresql 5432;
- orderService 18081, mongodb 27017, rabbitmq 5672, gRPC 50054;
- storeroomService 18082, postgresql 5433.

Для проверки работоспособности был выбран postman.



Для быстрого поднятия базы данных и брокера сообщений использовался docker.



Поднятие postgres для productService, storeroomService:

```
PS C:\Users\ThinkPadE470\Рабочий стол\выч_системы\student\productService> docker run --name=product -e POSTGRES_PASSWORD=querty -p 5432:5432 -d postgres 17c95f1c9d22e74a73353276dd436631a304fef19bf3a30d2551d913a2f559c2
```

```
PS C:\Users\ThinkPadE470\Рабочий стол\выч_системы\student> docker run --name=order -e POSTGRES_PASSWORD=querty -p 5433:5432 -d postgres
```

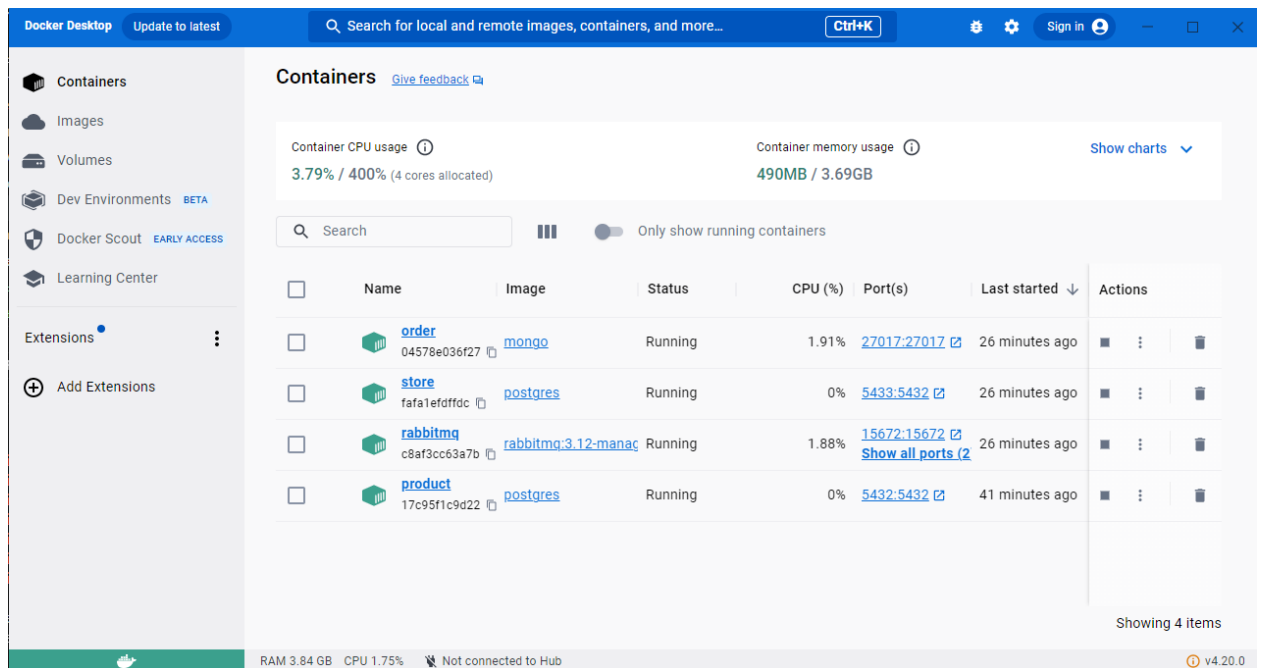
Поднятие mongodb для orderService:

```
PS C:\Users\ThinkPadE470\Рабочий стол\выч_системы\student> docker run -p 27017:27017 --name order mongo
```

Поднятие rabbitmq для orderService:

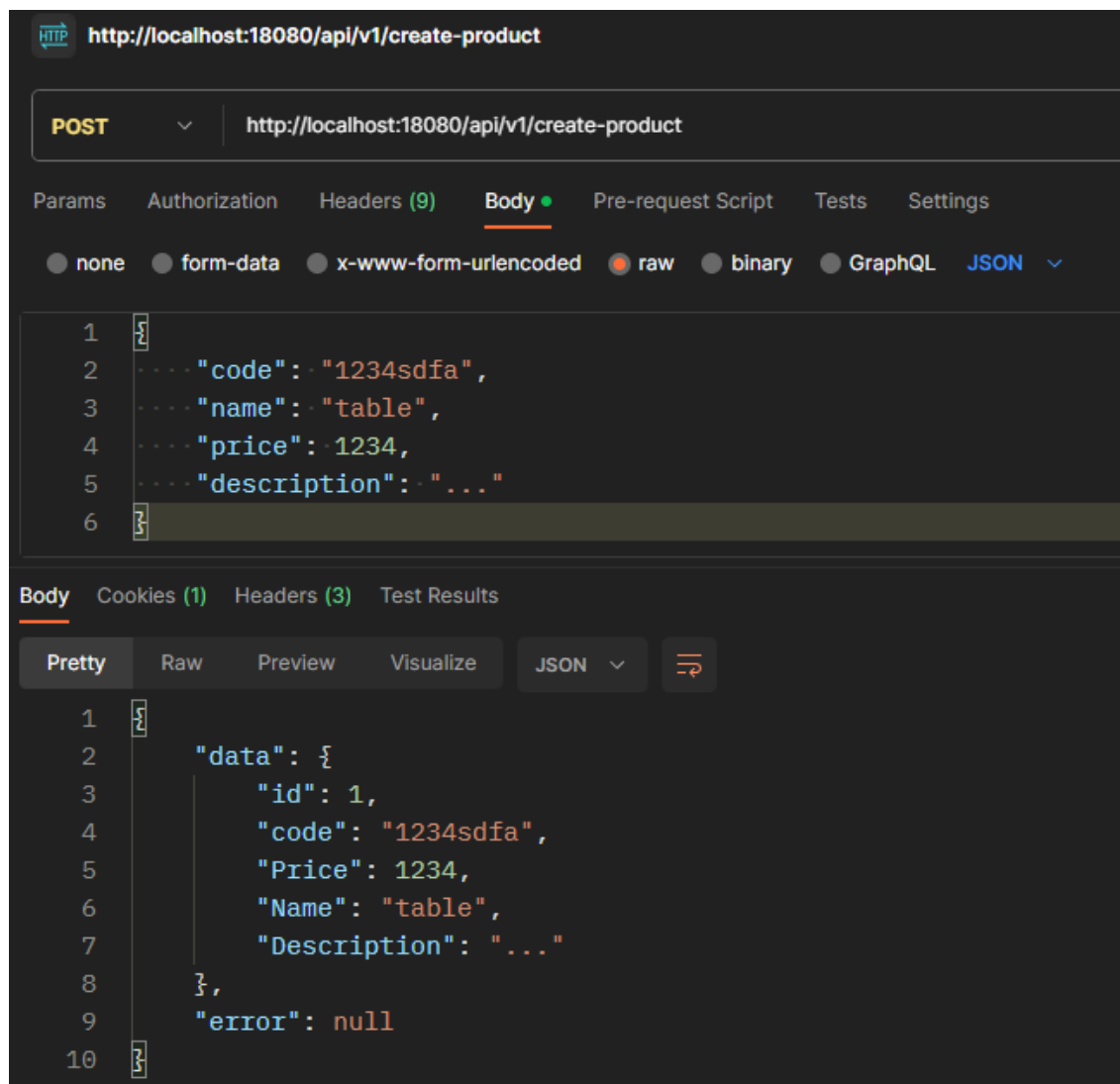
```
PS C:\Users\ThinkPadE470\Рабочий стол\выч_системы\student\orderService> docker run -it -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.12-management 88af3cc63a7b0b2cc0bff800cb136436f9e7b9e3a84b43b39eac0c3a352ba671
```

Как это выглядит в docker desktop:



Пример работы productService:

Создание продукта:



удаление продукта:

The screenshot displays a REST client interface for a DELETE request to `http://localhost:18080/api/v1/delete-product`. The request body is a JSON object with a `code` property set to `"1234sdfa"`. The response is also in JSON format, showing a `data` object with product details and an `error` set to `null`.

**Request:**

```
DELETE http://localhost:18080/api/v1/delete-product
```

**Body:**

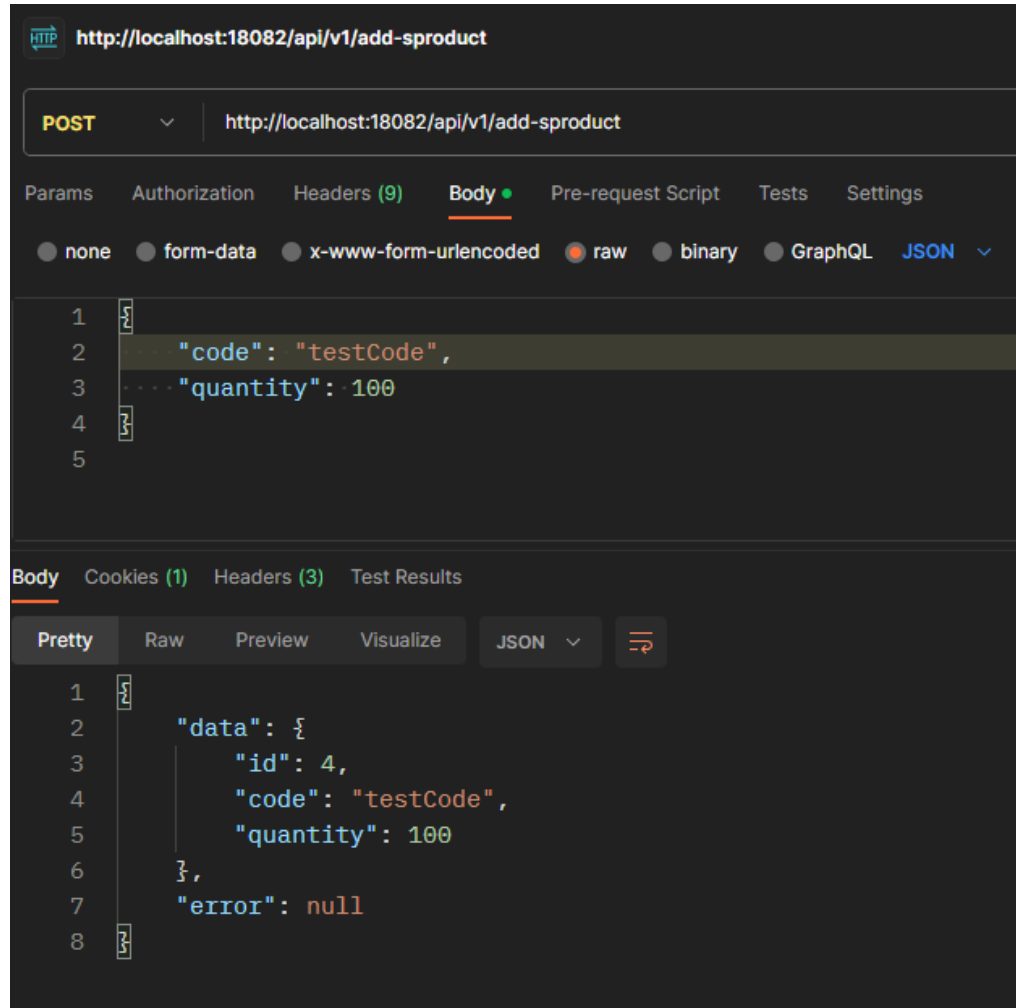
```
{
  "code": "1234sdfa"
}
```

**Response:**

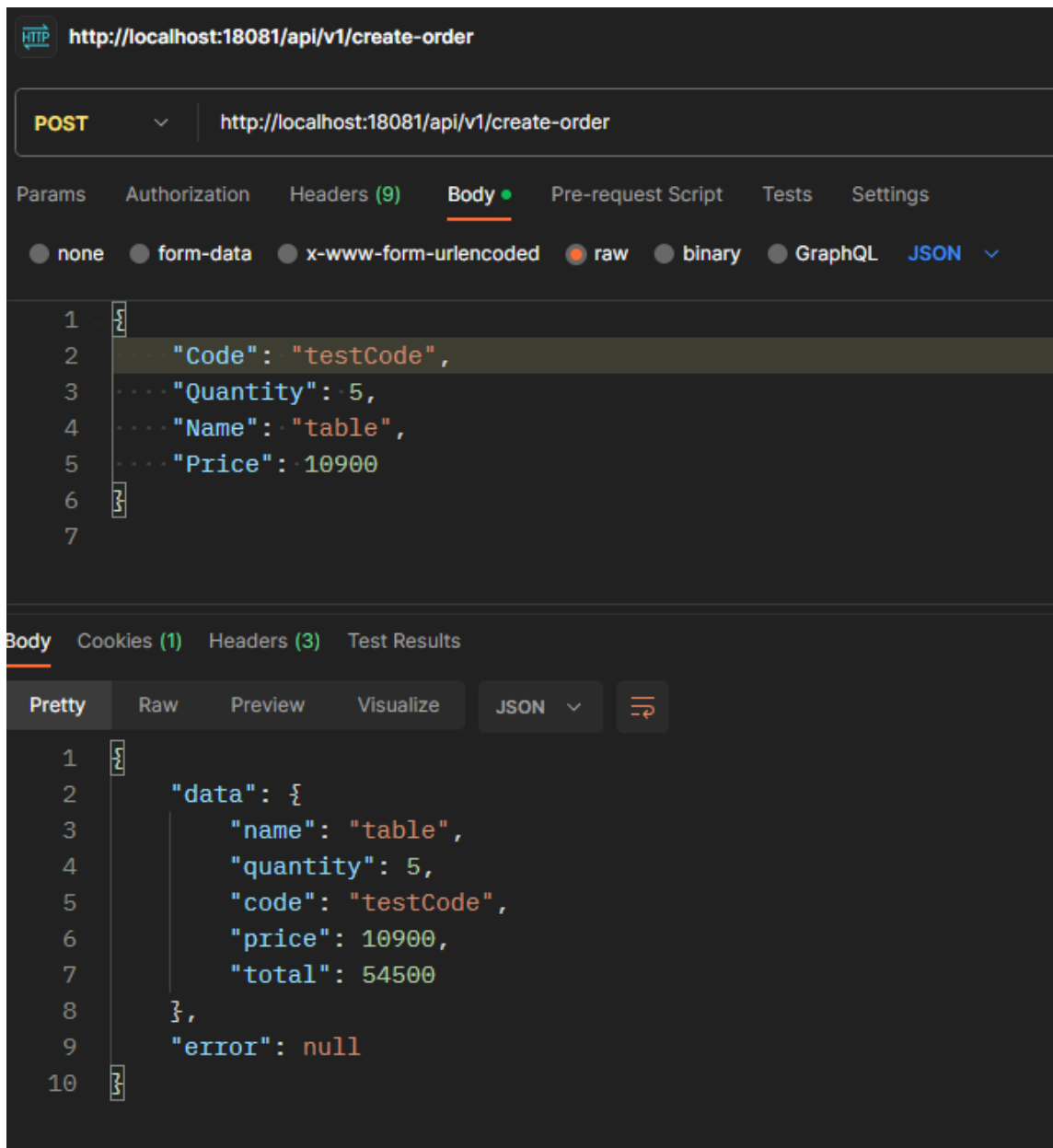
```
{
  "data": {
    "id": 1,
    "code": "1234sdfa",
    "Price": 1234,
    "Name": "table",
    "Description": "..."
  },
  "error": null
}
```

Пример работы orderService + storeroomService (общение через gRPC, отправка сообщение rabbitmq):

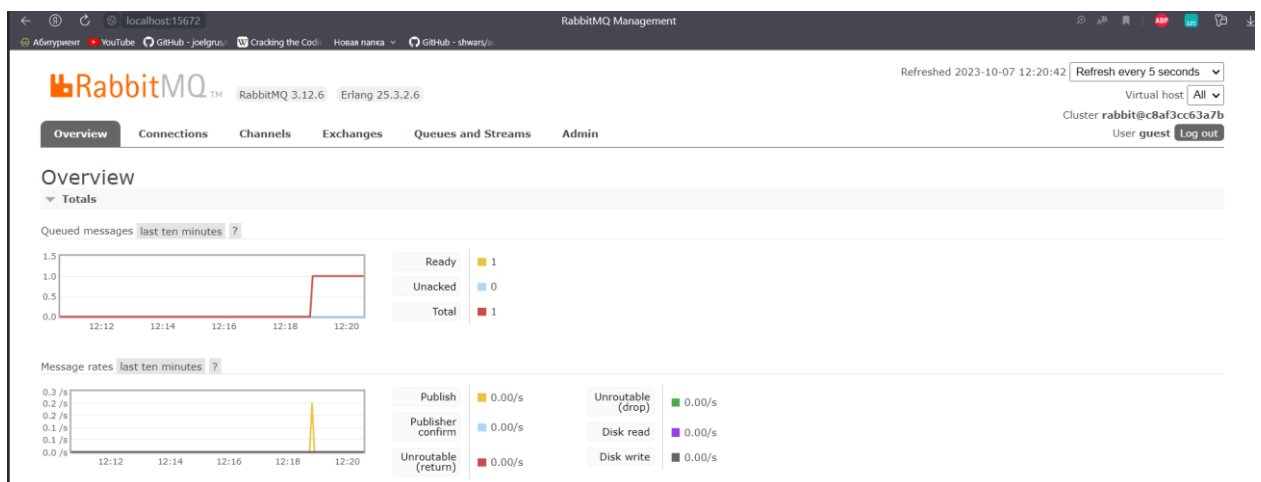
Добавление товара на склад:



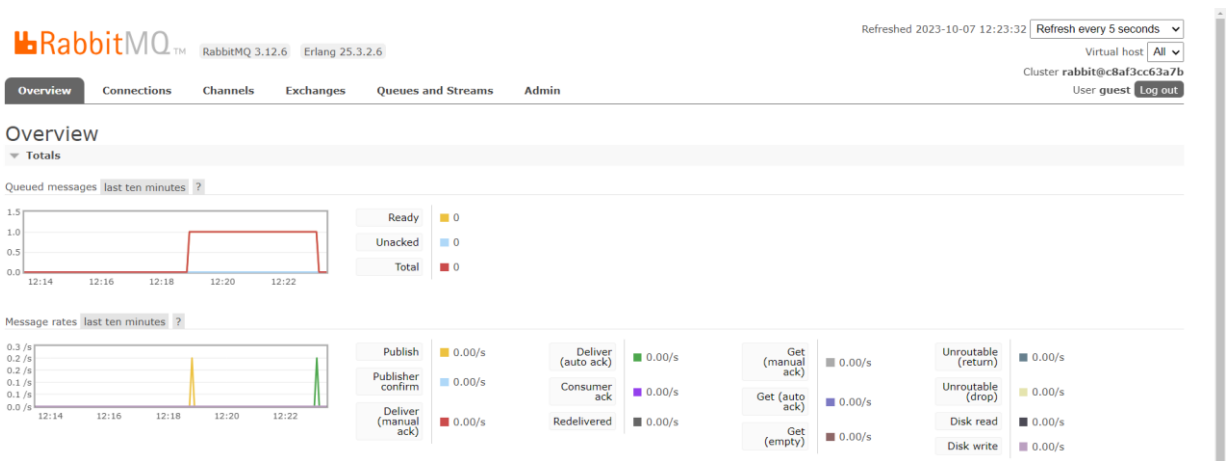
Создание заказа (идет проверка на складе через gRPC, отправка сообщение rabbitmq):



Просмотр отправленного сообщения rabbitmq:



Получатель сообщения из очереди rabbitmq:



При заказе больше, чем есть на складе заказ не создается:

HTTP **http://localhost:18081/api/v1/create-order**

**POST** **http://localhost:18081/api/v1/create-order**

Params Authorization Headers (9) **Body** Pre-request Script

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```
1 {
2   ... "Code": "testCode",
3   ... "Quantity": 102,
4   ... "Name": "table",
5   ... "Price": 10900
6 }
7
```

**Body** Cookies (1) Headers (3) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": null
3 }
```