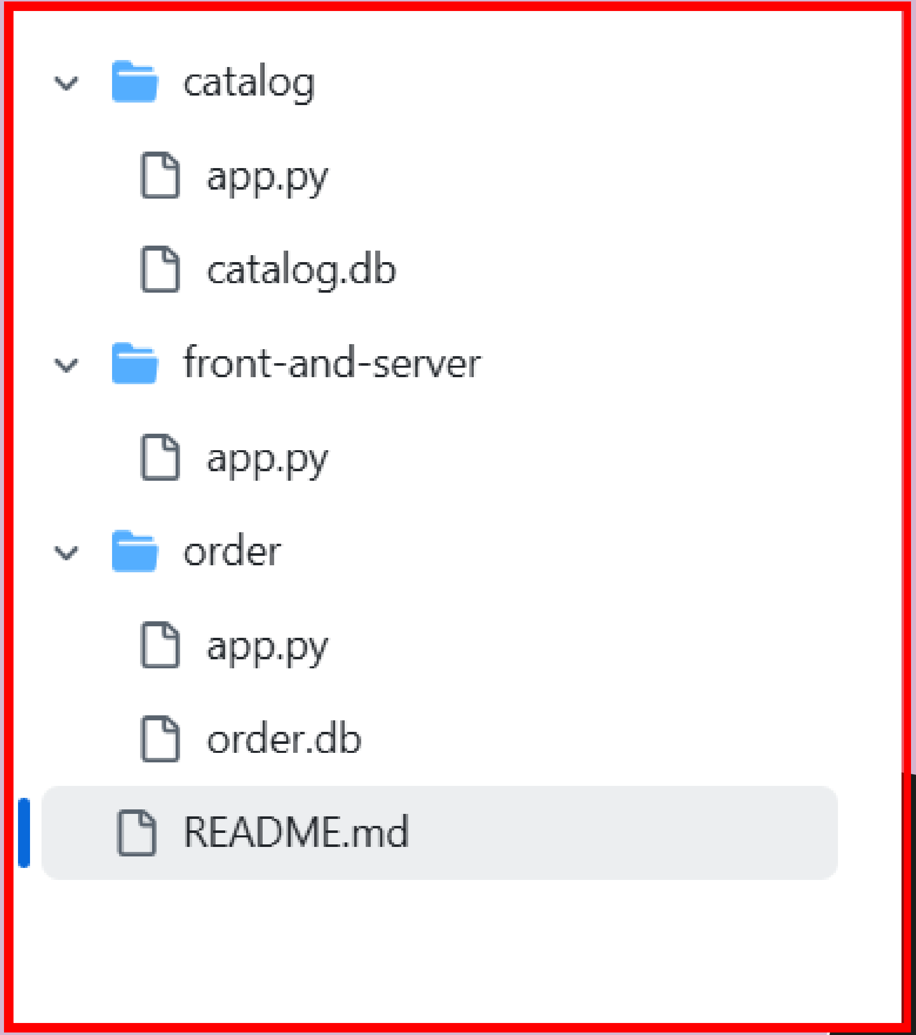
DOS

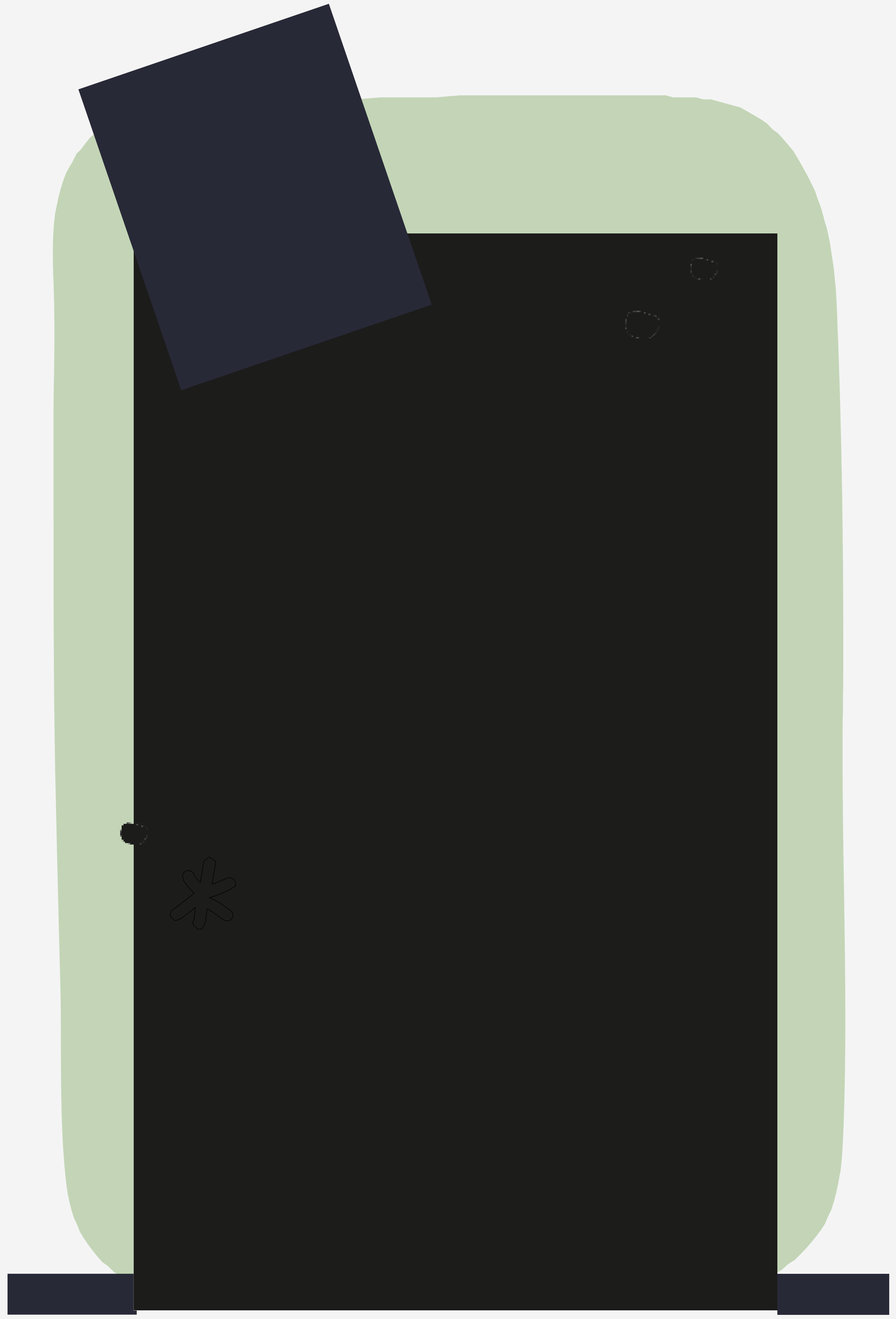
Fatima zahra Saidi
Lana Hassan

The project is an online book store that we are required to develop using three microservices

- Order Service: This is concerned with all user requests and checks if the book is available and how much stock we have of it
- Catalog Service: This service works directly with the database and we can search for book information and work on changing it
- Front-end Service: This is the interface for the user to be able to use the store, search and order and is linked to the catalog



```
catalog
    app.py
    catalog.db
front-and-server
    app.py
order
    app.py
    order.db
README.md
```



We developed it using Flask, where the user interface can be easily interacted with

- Catalog Service (5001): This service provides two dedicated endpoints for integration with other services: query and update.
- Query endpoint: Supports two types of GET requests: "query/item" and "query/topic". The "query/topic" request sends a search string to the URI that retrieves matching books in the database, returning a list of those books, or an empty list if no match is found. Regarding the "query/item" request, it takes the book's identifier from the URI and returns the book's details if it exists, while it returns a 404 NOT FOUND status if the book is not found.
- Update endpoint: PUT requests allow updating book information. A person can modify any part of the book information (except the identifier) by sending the modified data in JSON format, while preserving the information before the modification. The book number is sent via URI, if the book is found, it shows the modified data in JSON format, if the book is not found, it returns 404 NOT FOUND.

Service 5002 is only an endpoint and represents the purchase process. When we PUT with the title URL of the book, it first checks if it is identifier and then retrieves the information about the book from the catalog. After locating the book, it will check how many copies are left and whether there are any copies left.

The 5000 service is for the user to perform searches, through which the purchase process is made and he searches for the information he may need, so he uses GET with the URL address included, the request will reach the catalog,
The same method is for searches, but we use PUT to deliver the requests and transfer them to the point of purchase.

Database :

Catalog : include books table.

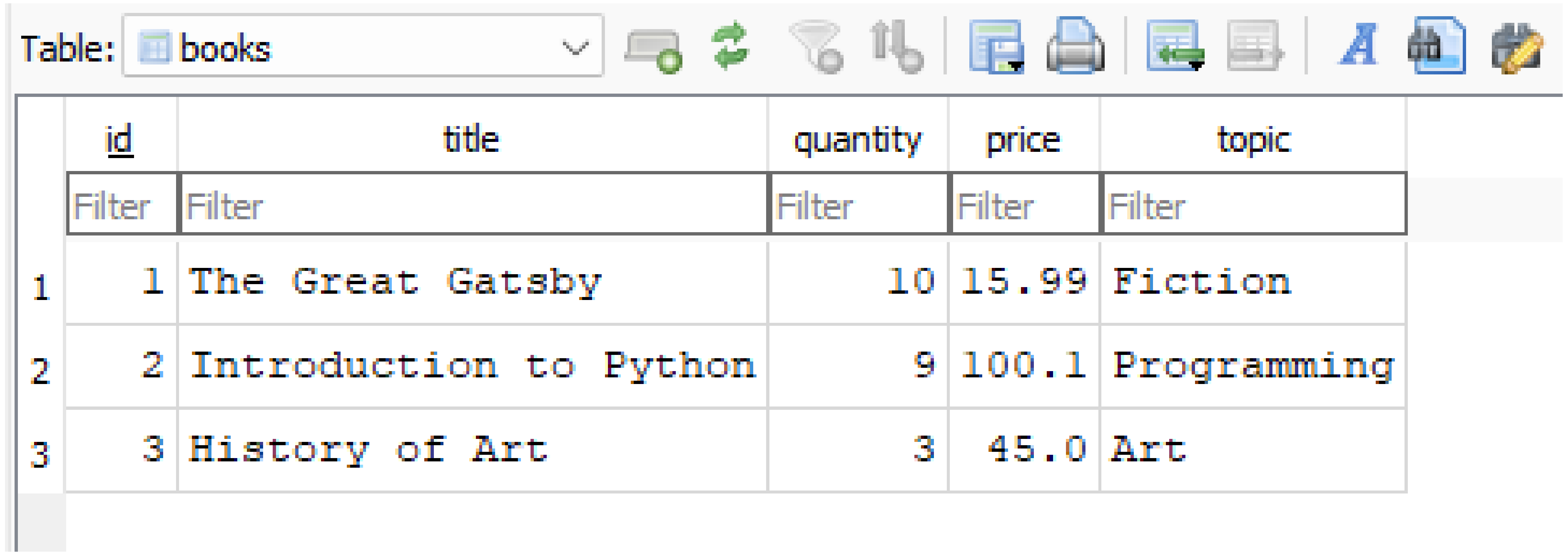


Table: books

	id	title	quantity	price	topic
	Filter	Filter	Filter	Filter	Filter
1	1	The Great Gatsby	10	15.99	Fiction
2	2	Introduction to Python	9	100.1	Programming
3	3	History of Art	3	45.0	Art

order : include orders table.

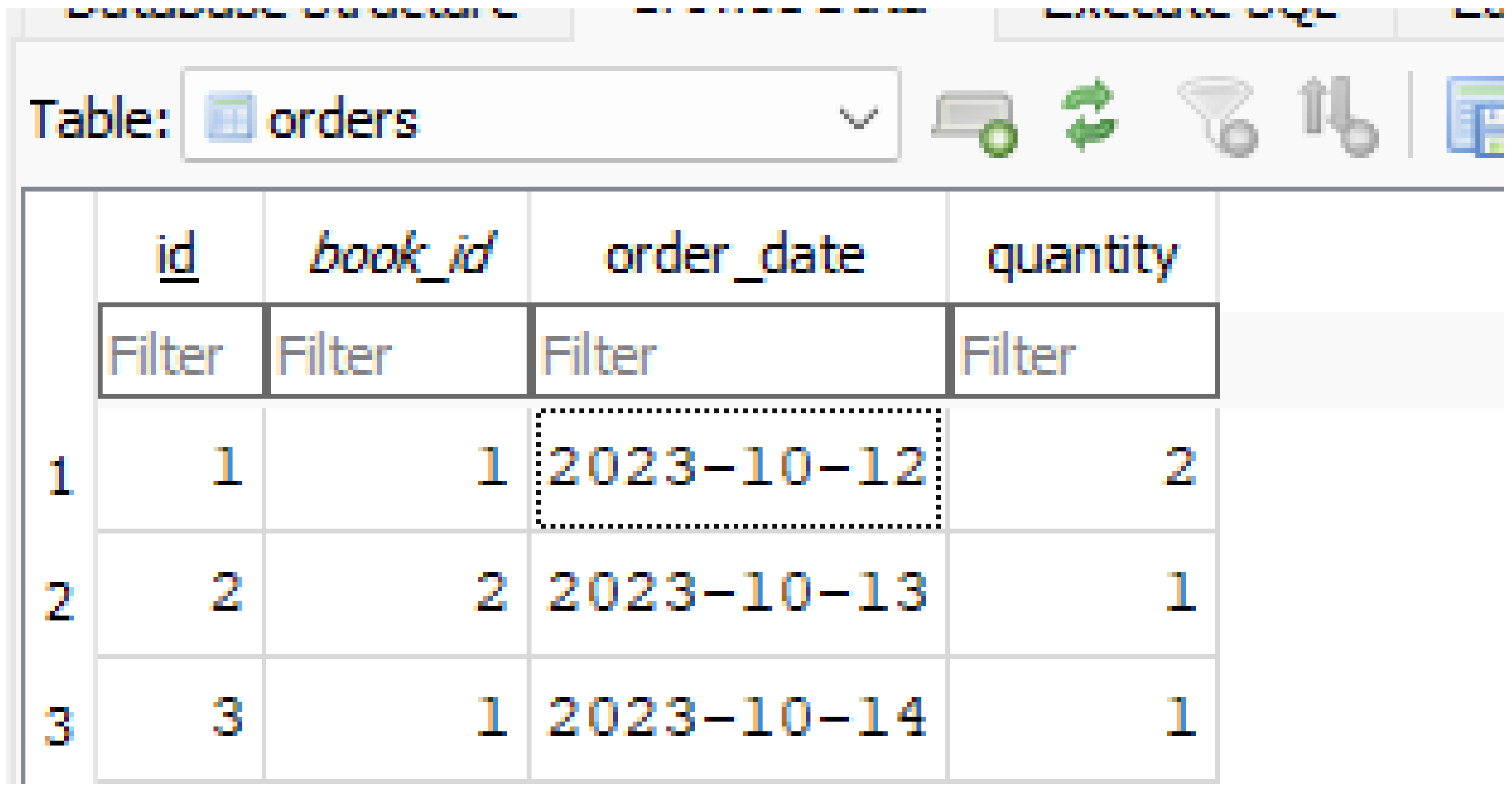
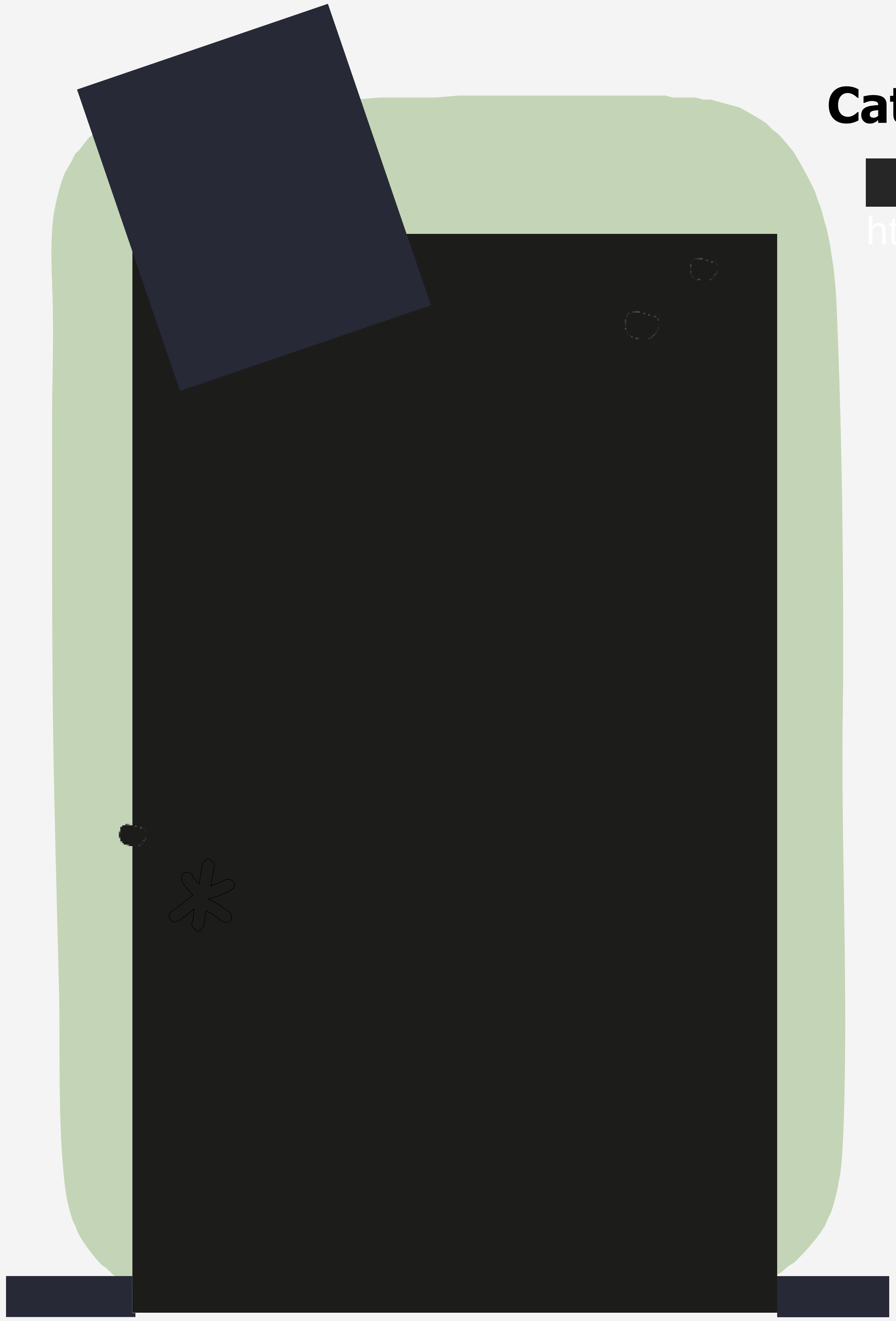


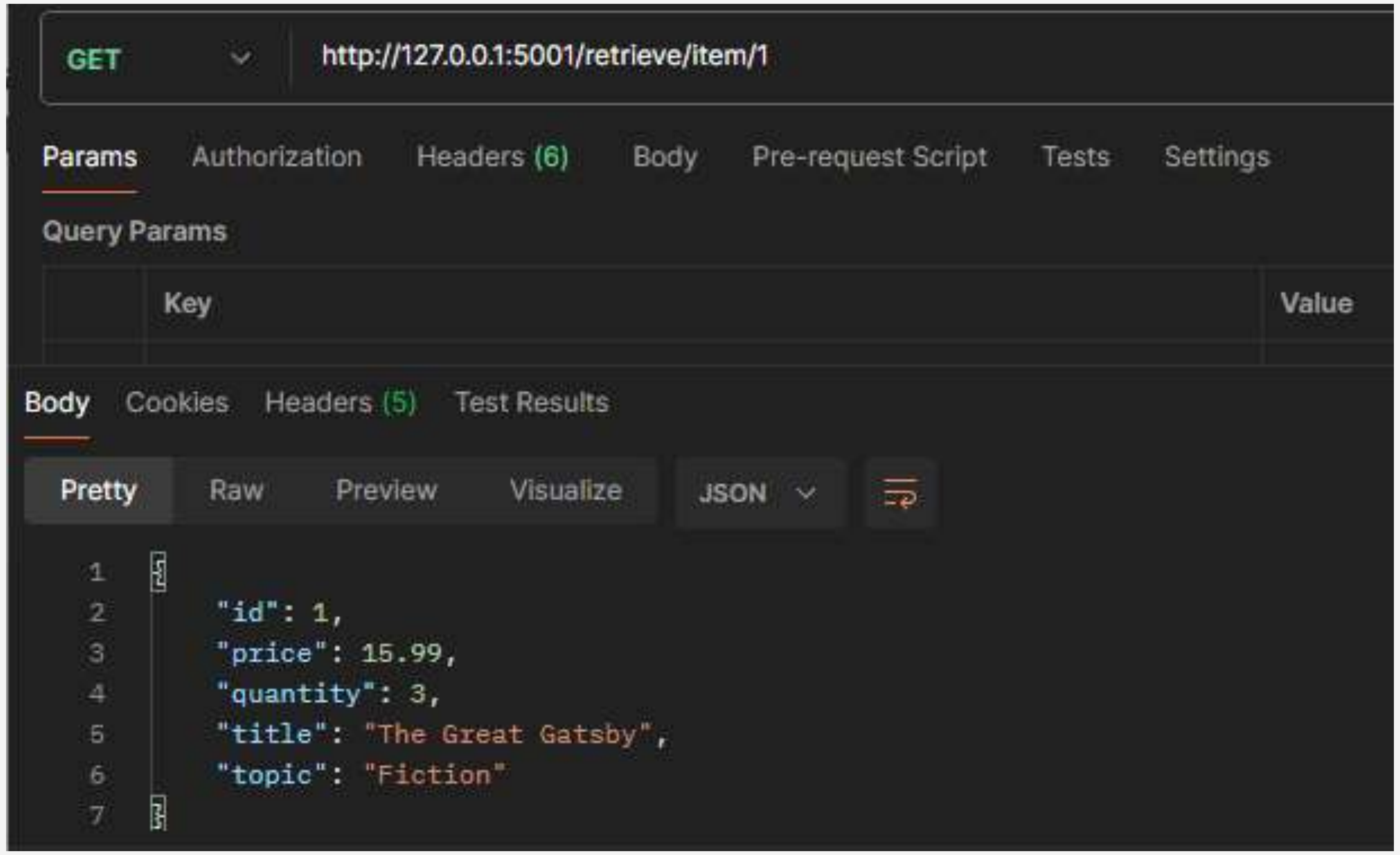
Table: orders

	id	*book_id*	order_date	quantity
	Filter	Filter	Filter	Filter
1	1	1	2023-10-12	2
2	2	2	2023-10-13	1
3	3	1	2023-10-14	1

Catalog server on port 5001

http://127.0.0.1:5001/retrieve/item/1



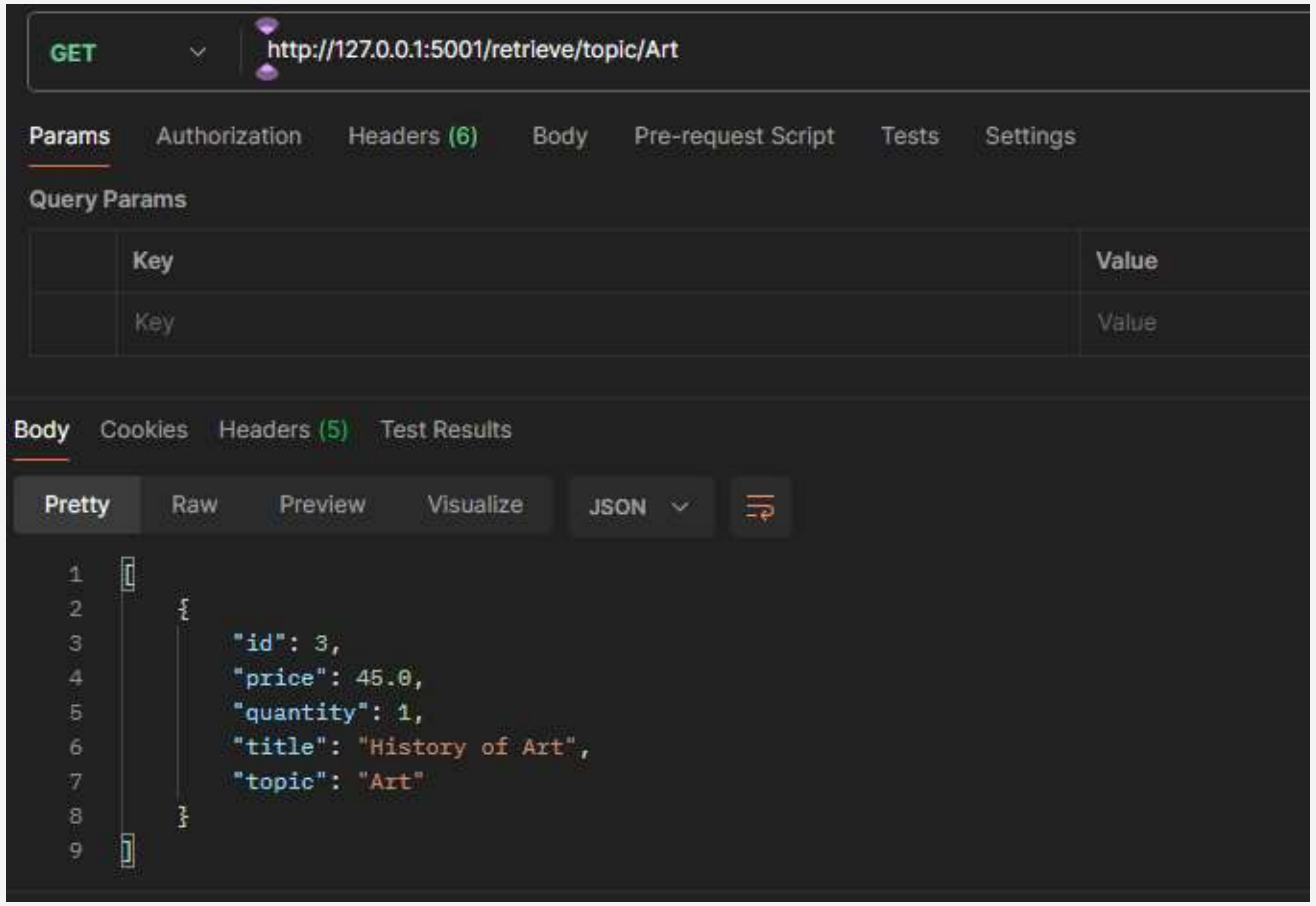


http://127.0.0.1:5001/retrieve/item/1

```
{
    "id": 1,
    "price": 15.99,
    "quantity": 3,
    "title": "The Great Gatsby",
    "topic": "Fiction"
}
```

Catalog server on port 5001

```
http://127.0.0.1:5001/retrieve/topic/Art
```

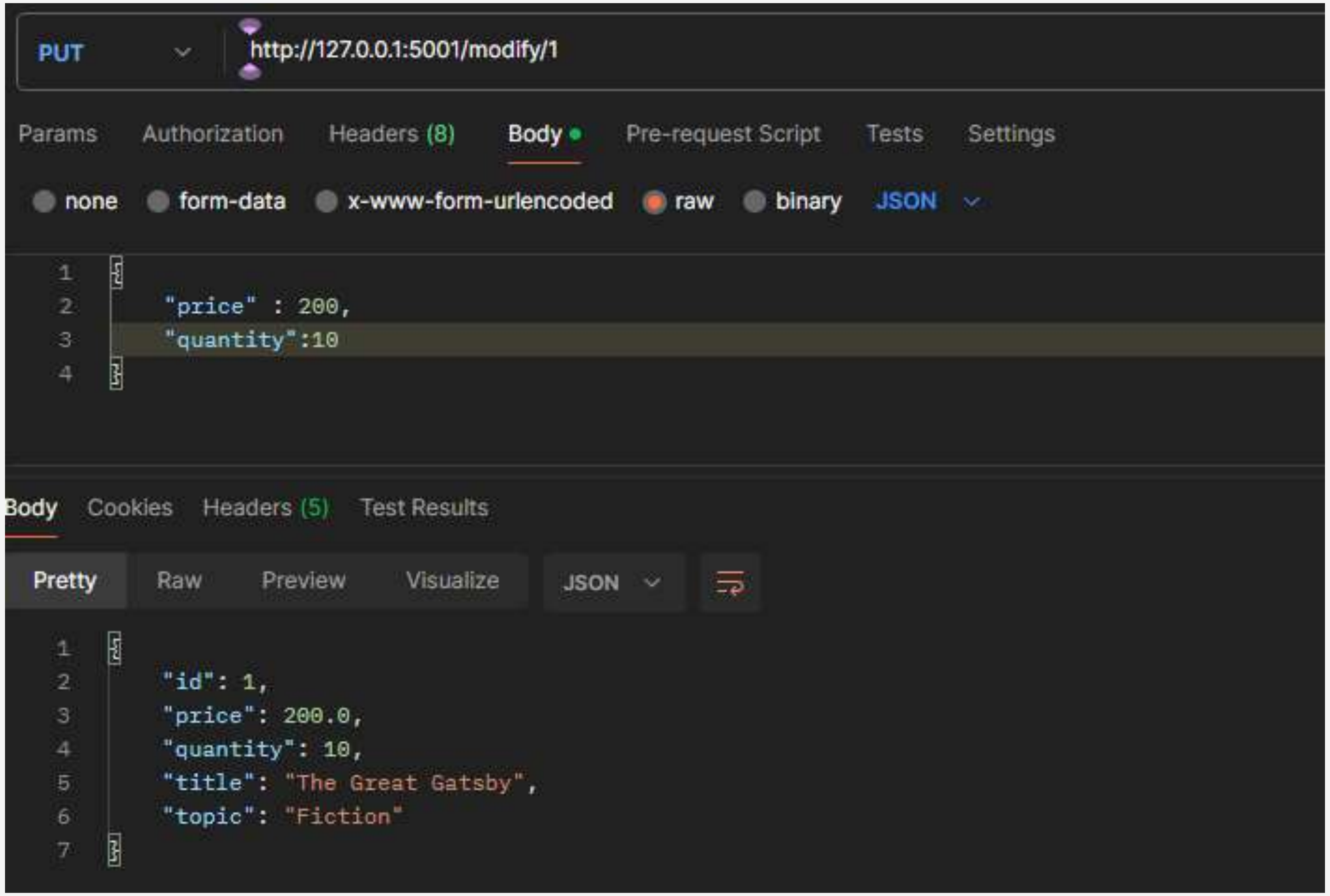


GET http://127.0.0.1:5001/retrieve/topic/Art

```
[
    {
        "id": 3,
        "price": 45.0,
        "quantity": 1,
        "title": "History of Art",
        "topic": "Art"
    }
]
```

Catalog server on port 5001

http://127.0.0.1:5001/modify/1



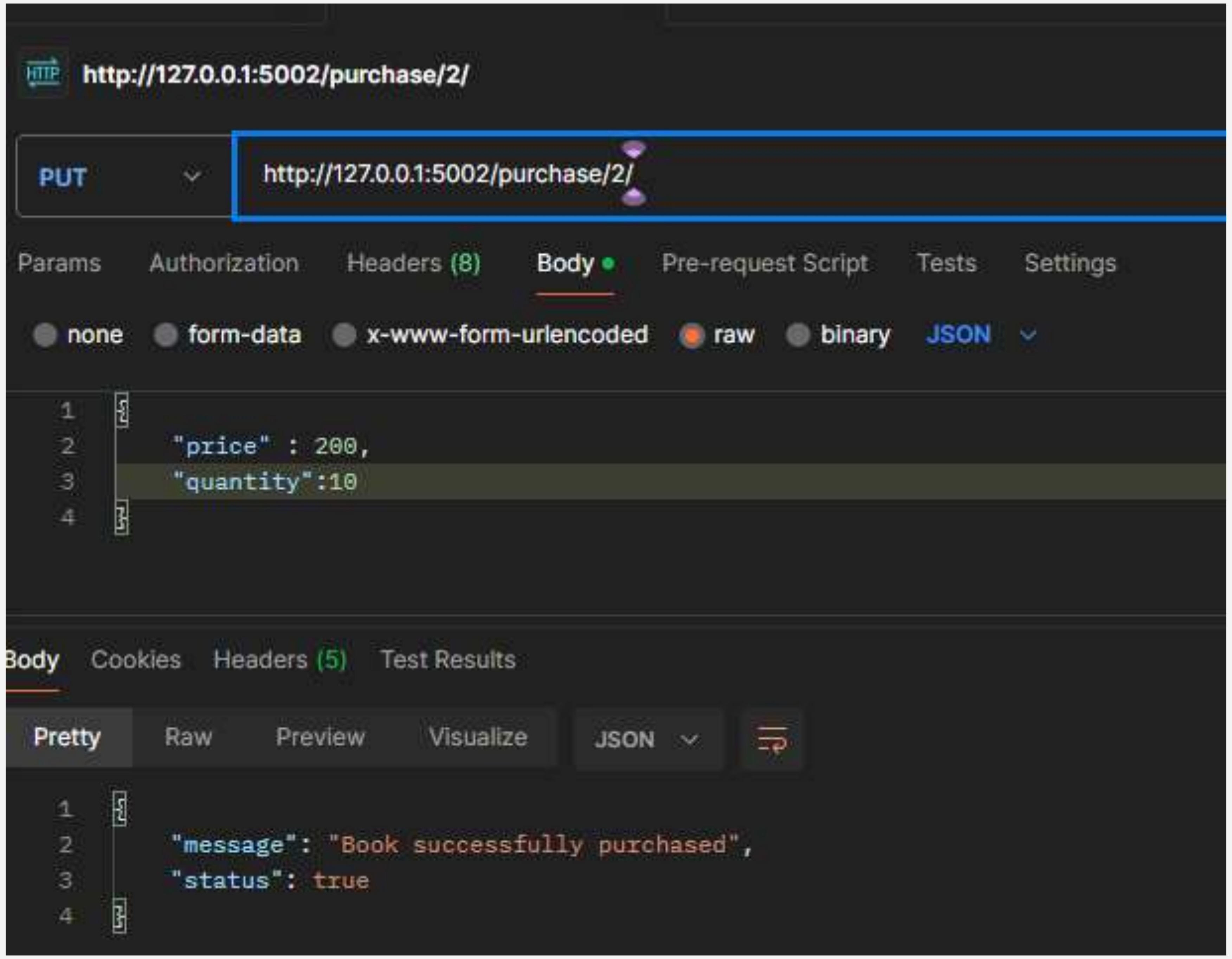
http://127.0.0.1:5001/modify/1

```
{
    "price" : 200,
    "quantity":10
}
```

```
{
    "id": 1,
    "price": 200.0,
    "quantity": 10,
    "title": "The Great Gatsby",
    "topic": "Fiction"
}
```

Order server on port 5002

http://127.0.0.1:5001/purchase/2/


http://127.0.0.1:5002/purchase/2/

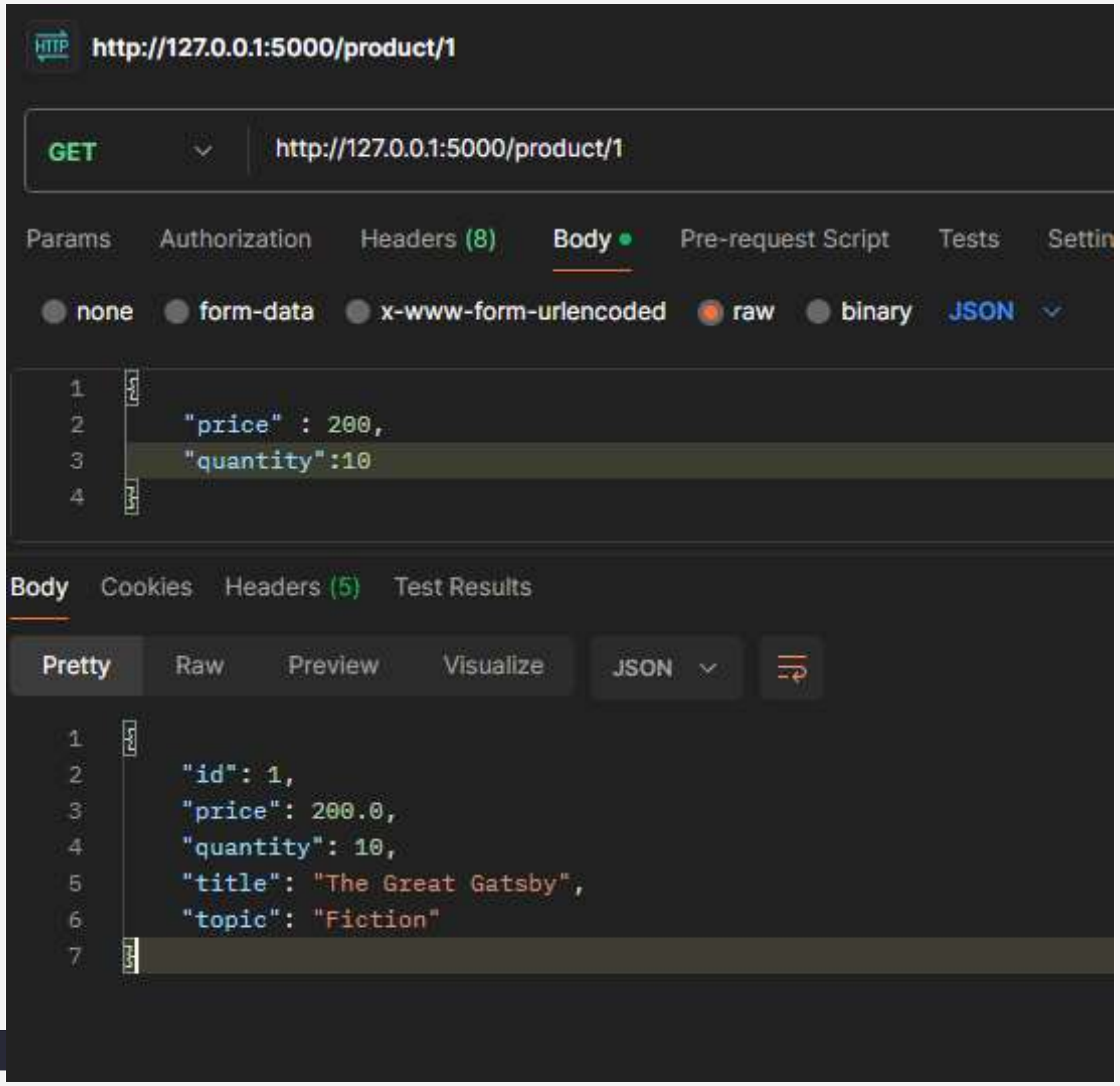
PUT http://127.0.0.1:5002/purchase/2/

```
{
    "price" : 200,
    "quantity":10
}
```

```
{
    "message": "Book successfully purchased",
    "status": true
}
```

Front-and-server server on port 5000

http://127.0.0.1:5000/product/1



http://127.0.0.1:5000/product/1

GET http://127.0.0.1:5000/product/1

Params Authorization Headers (8) Body Pre-request Script Tests Settin

none form-data x-www-form-urlencoded [✓ raw] binary JSON

```
{
    "price" : 200,
    "quantity":10
}
```

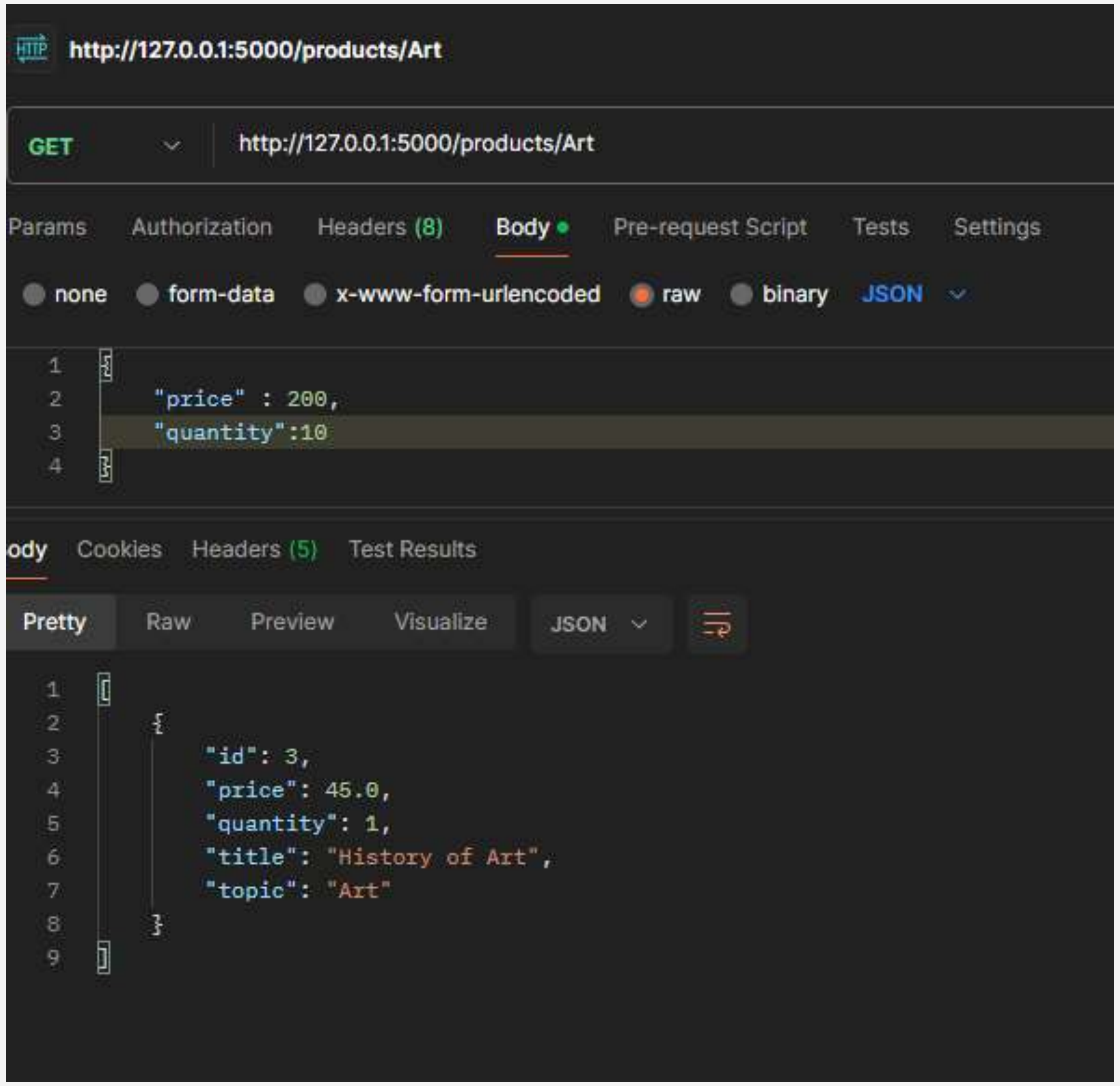
Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "id": 1,
    "price": 200.0,
    "quantity": 10,
    "title": "The Great Gatsby",
    "topic": "Fiction"
}
```

Front-and-server server on port 5000

http://127.0.0.1:5000/products/Art



http://127.0.0.1:5000/products/Art

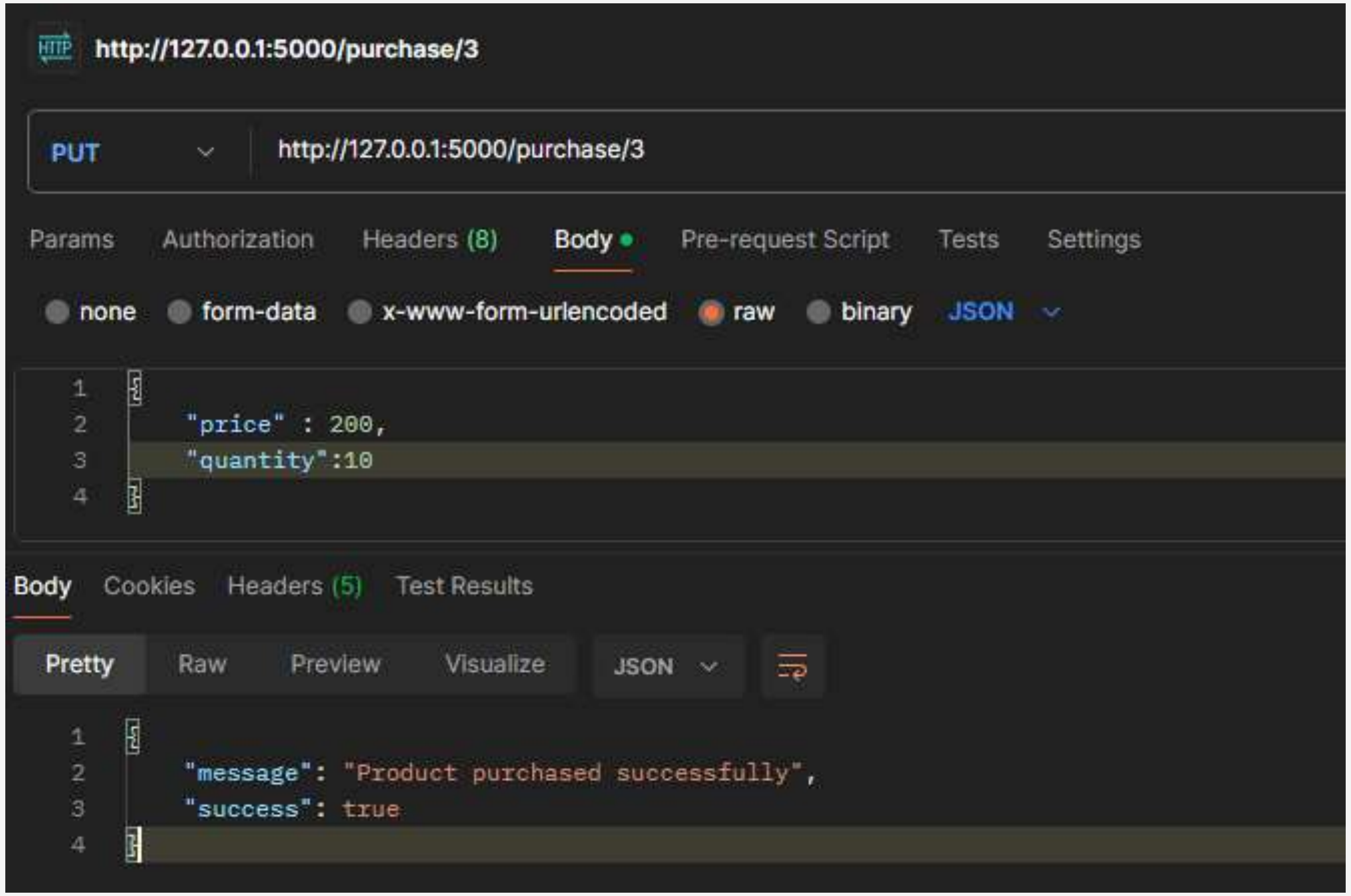
GET http://127.0.0.1:5000/products/Art

```
{
    "price" : 200,
    "quantity":10
}
```

```
[
    {
        "id": 3,
        "price": 45.0,
        "quantity": 1,
        "title": "History of Art",
        "topic": "Art"
    }
]
```

Front-and-server server on port 5000

http://127.0.0.1:5000/purchase/3



http://127.0.0.1:5000/purchase/3

PUT http://127.0.0.1:5000/purchase/3

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded [✓ raw] binary JSON

```
{
    "price" : 200,
    "quantity":10
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "Product purchased successfully",
    "success": true
}
```

There were several servers, each running on a separate machine. The first server was running Windows, while the other two were running Ubuntu. These servers interacted with each other over the network using the HTTP protocol.

We test the functionality of the three servers using Postman to send requests back and forth. The front-end server starts where the client interacts.

Each separate machine has several servers
The first one runs Windows
The second and third Ubuntu
They must interact with each other over the network using HTTP
We test the functionality of the three servers to send requests in both directions Sending and receiving starts from the user's server