An-Najah National University
Faculty of Engineering and IT

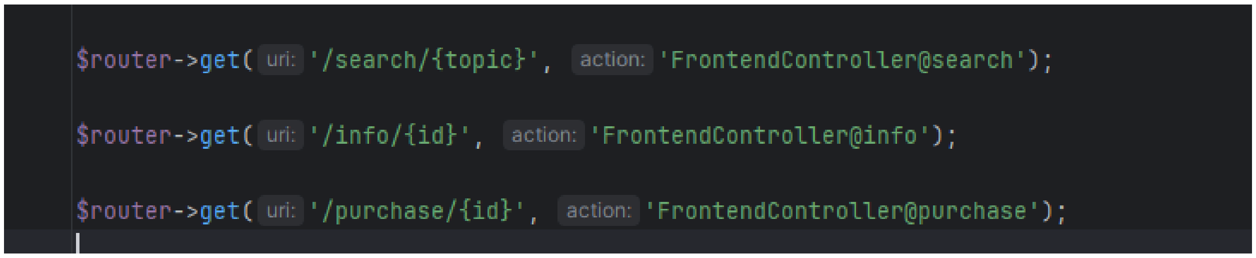
جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا المعلومات

Computer Engineering Department
Distributed operating systems
Report Lab 2: Bazar.com Microservices

Instructor Name: Dr.Samer Arandi	**Academic Year: 2023/2024**
Semester: 1st	**Student Name:** Majed Kawa
	Student ID: 11923626

Front-End Interaction:

1. Users interact with the front-end microservice through HTTP REST calls.



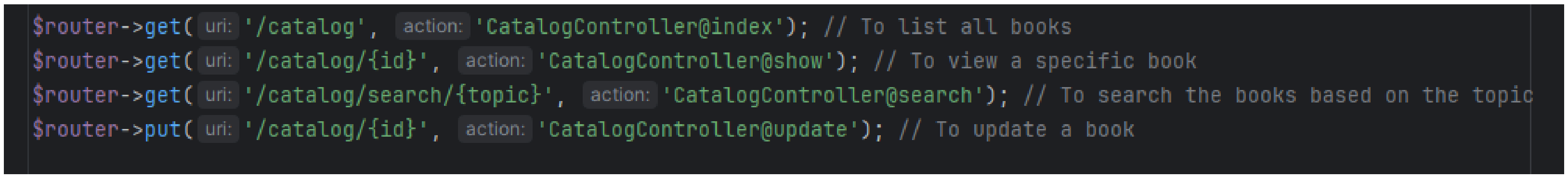
```
$router->get( uri: '/search/{topic}',  action: 'FrontendController@search');

$router->get( uri: '/info/{id}',  action: 'FrontendController@info');

$router->get( uri: '/purchase/{id}',  action: 'FrontendController@purchase');
```

2. Search and info operations trigger queries to the catalog microservice.
3. Purchase operation triggers a request to the orders microservice.

   In this Part, I used simple round robin to balance the load to the microservices

Catalog Microservice:

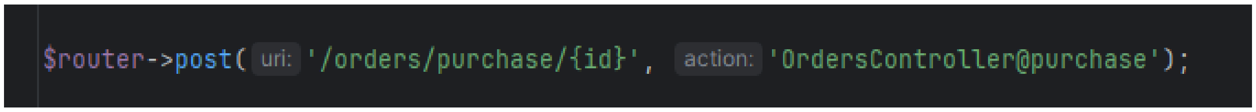
1. Handles queries and updates to the book catalog.
2. Maintains persistent data in a CSV file.
3. Exposes REST endpoints for search, show (view) and update operations.



```
$router->get( uri: '/catalog',  action: 'CatalogController@index'); // To list all books
$router->get( uri: '/catalog/{id}',  action: 'CatalogController@show'); // To view a specific book
$router->get( uri: '/catalog/search/{topic}',  action: 'CatalogController@search'); // To search the books based on the topic
$router->put( uri: '/catalog/{id}',  action: 'CatalogController@update'); // To update a book
```

Orders Microservice:

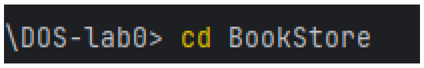
1. Verifies item availability through the catalog microservice before processing a purchase.
2. Updates the catalog microservice with the new item quantity upon successful purchase.



```
$router->post( uri: '/orders/purchase/{id}',  action: 'OrdersController@purchase');
```

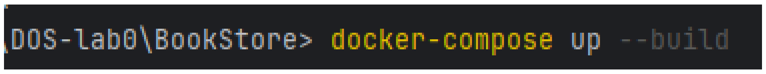
Running the Program:

1) We need to open docker desktop app
2) Go to the BookStore dir.



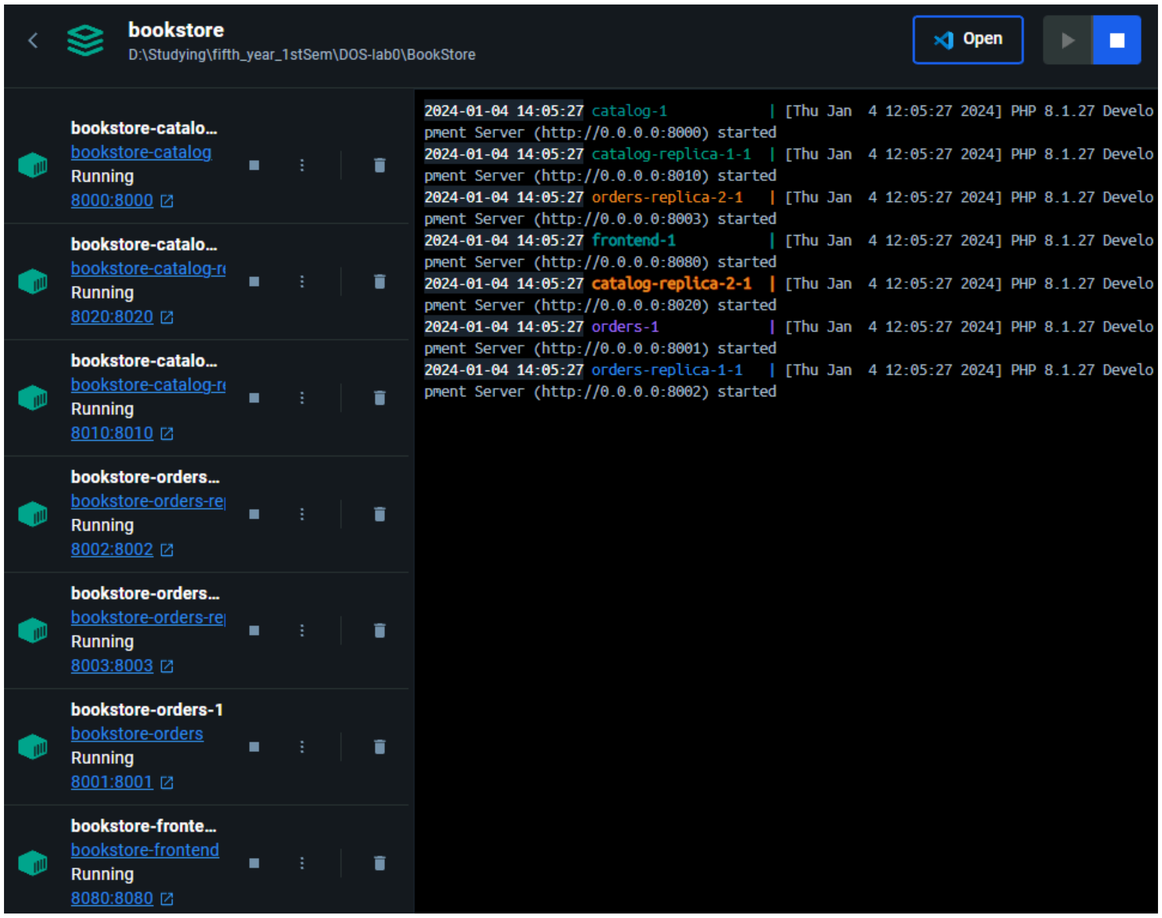
```
\DOS-lab0> cd BookStore
```

3) We need to run this command in the IDE terminal : docker-compose up --build



```
DOS-lab0\BookStore> docker-compose up --build
```

After it finishes,we can see every microservice runs in a separate container. And now we can Interact with the microservices,



bookstore
D:\Studying\fifth_year_1stSem\DOS-lab0\BookStore

bookstore-catalo...
bookstore-catalog
Running
8000:8000

bookstore-catalo...
bookstore-catalog-r
Running
8020:8020

bookstore-catalo...
bookstore-catalog-r
Running
8010:8010

bookstore-orders...
bookstore-orders-re
Running
8002:8002

bookstore-orders...
bookstore-orders-re
Running
8003:8003

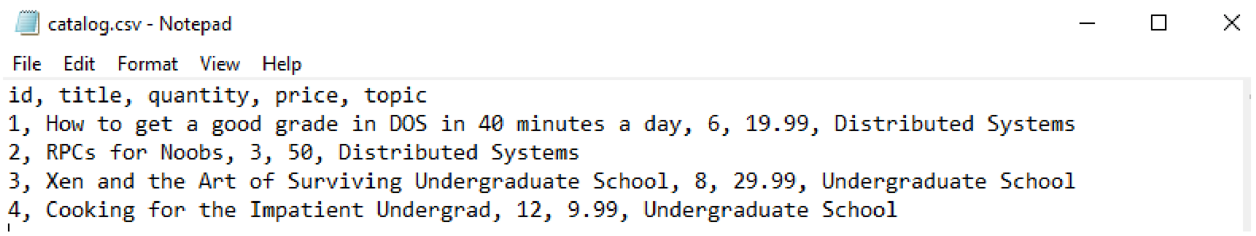
bookstore-orders-1
bookstore-orders
Running
8001:8001

bookstore-fronte...
bookstore-frontend
Running
8080:8080

```
2024-01-04 14:05:27 catalog-1           | [Thu Jan  4 12:05:27 2024] PHP 8.1.27 Develo
pment Server (http://0.0.0.0:8000) started
2024-01-04 14:05:27 catalog-replica-1-1  | [Thu Jan  4 12:05:27 2024] PHP 8.1.27 Develo
pment Server (http://0.0.0.0:8010) started
2024-01-04 14:05:27 orders-replica-2-1   | [Thu Jan  4 12:05:27 2024] PHP 8.1.27 Develo
pment Server (http://0.0.0.0:8003) started
2024-01-04 14:05:27 frontend-1           | [Thu Jan  4 12:05:27 2024] PHP 8.1.27 Develo
pment Server (http://0.0.0.0:8080) started
2024-01-04 14:05:27 catalog-replica-2-1  | [Thu Jan  4 12:05:27 2024] PHP 8.1.27 Develo
pment Server (http://0.0.0.0:8020) started
2024-01-04 14:05:27 orders-1             | [Thu Jan  4 12:05:27 2024] PHP 8.1.27 Develo
pment Server (http://0.0.0.0:8001) started
2024-01-04 14:05:27 orders-replica-1-1   | [Thu Jan  4 12:05:27 2024] PHP 8.1.27 Develo
pment Server (http://0.0.0.0:8002) started
```

Testing the endpoints from the front end microservice:

*here is the csv file structure, it's path is Dos-lab0\catalog-microservice\storage\app:


catalog.csv - Notepad

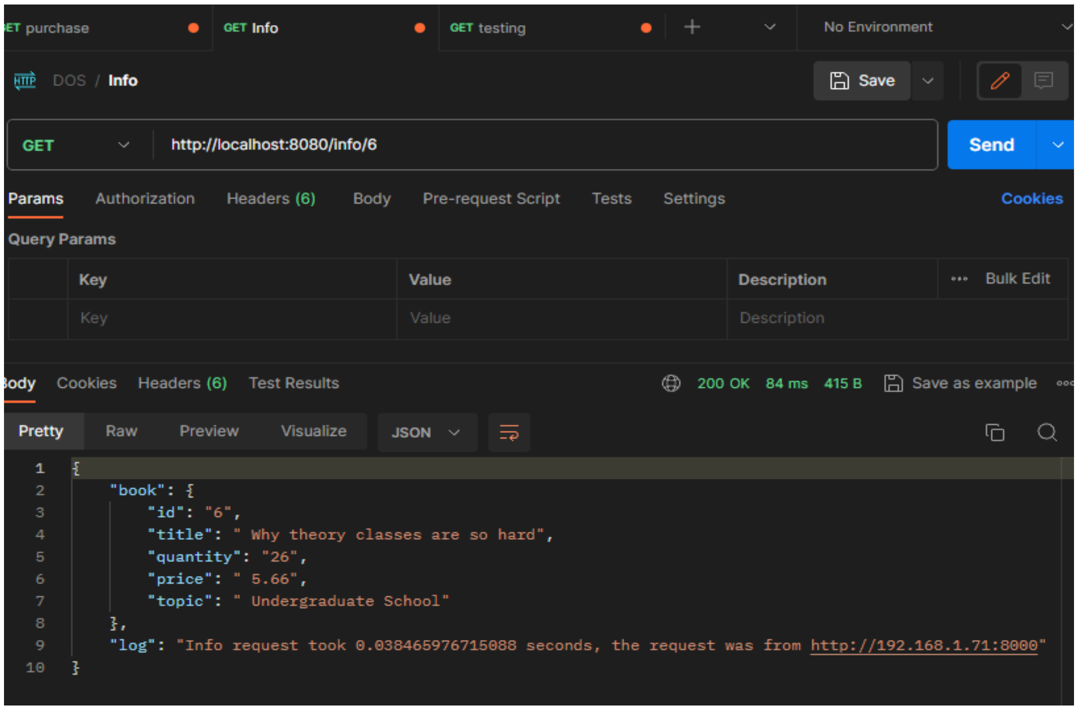
```
id, title, quantity, price, topic
1, How to get a good grade in DOS in 40 minutes a day, 6, 19.99, Distributed Systems
2, RPCs for Noobs, 3, 50, Distributed Systems
3, Xen and the Art of Surviving Undergraduate School, 8, 29.99, Undergraduate School
4, Cooking for the Impatient Undergrad, 12, 9.99, Undergraduate School
```

I) Info endpoint

This endpoint is used to get the info of a specific book using its id.

Using Postman, after entering the following URL to get the book with id = 2 from the catalog microservice:

http://localhost:8080/info/6


GET http://localhost:8080/info/6

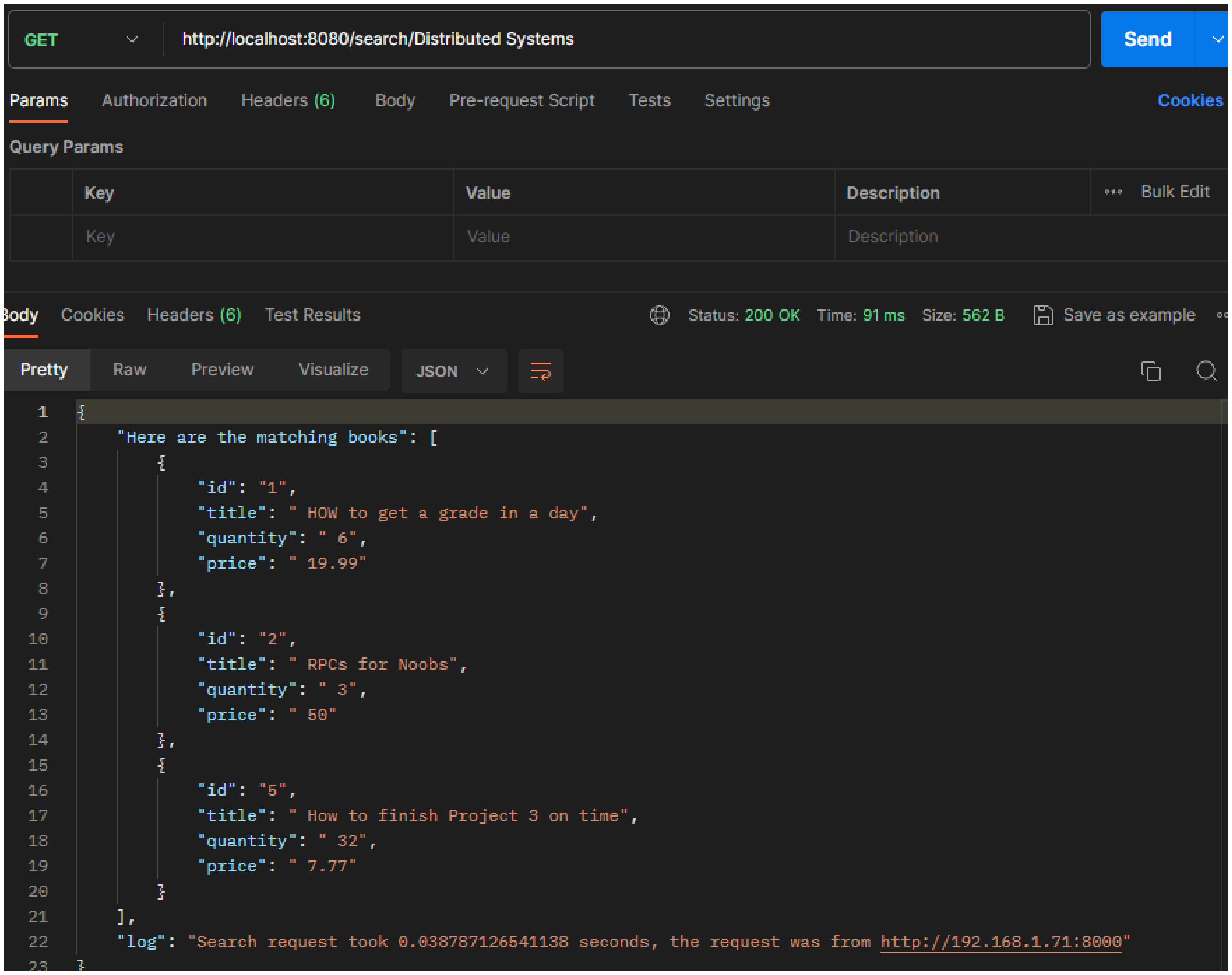
200 OK 84 ms 415 B

```
{
    "book": {
        "id": "6",
        "title": " Why theory classes are so hard",
        "quantity": "26",
        "price": " 5.66",
        "topic": " Undergraduate School"
    },
    "log": "Info request took 0.038465976715088 seconds, the request was from http://192.168.1.71:8000"
}
```

II) Search Endpoint:

This endpoint is used to allow the user to specify a topic and returns all entries belonging to that topic.

Using Postman, after entering the following URL, with the desired topic :http://localhost:8080/search/Distributed Systems

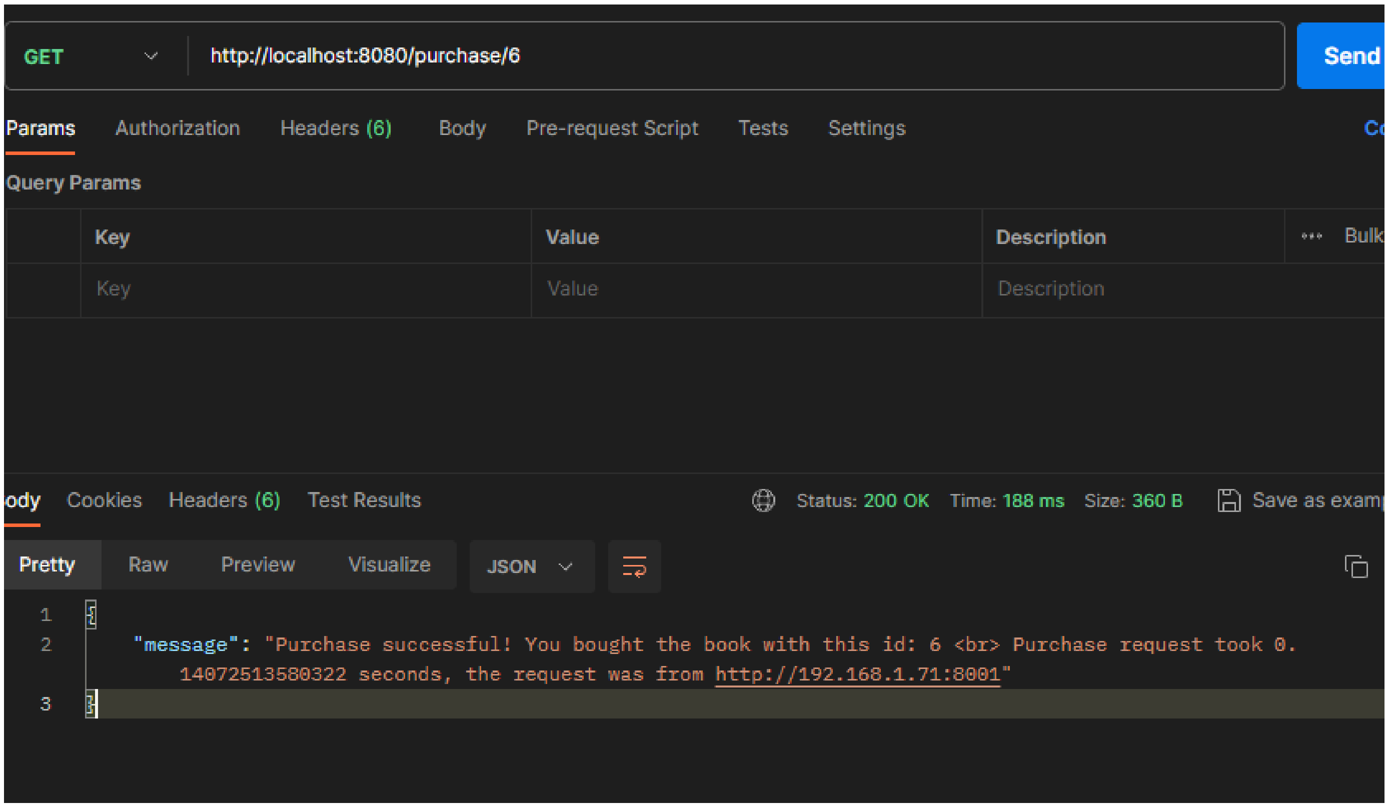


```
{
    "Here are the matching books": [
        {
            "id": "1",
            "title": " HOW to get a grade in a day",
            "quantity": " 6",
            "price": " 19.99"
        },
        {
            "id": "2",
            "title": " RPCs for Noobs",
            "quantity": " 3",
            "price": " 50"
        },
        {
            "id": "5",
            "title": " How to finish Project 3 on time",
            "quantity": " 32",
            "price": " 7.77"
        }
    ],
    "log": "Search request took 0.038787126541138 seconds, the request was from http://192.168.1.71:8000"
}
```

III) Purchase Endpoint

This endpoint is used to allow the user to enter the id of the book that he wants to purchase.

Using Postman, after entering the following URL, with the desired id number of the book :http://localhost:8080/purchase/6

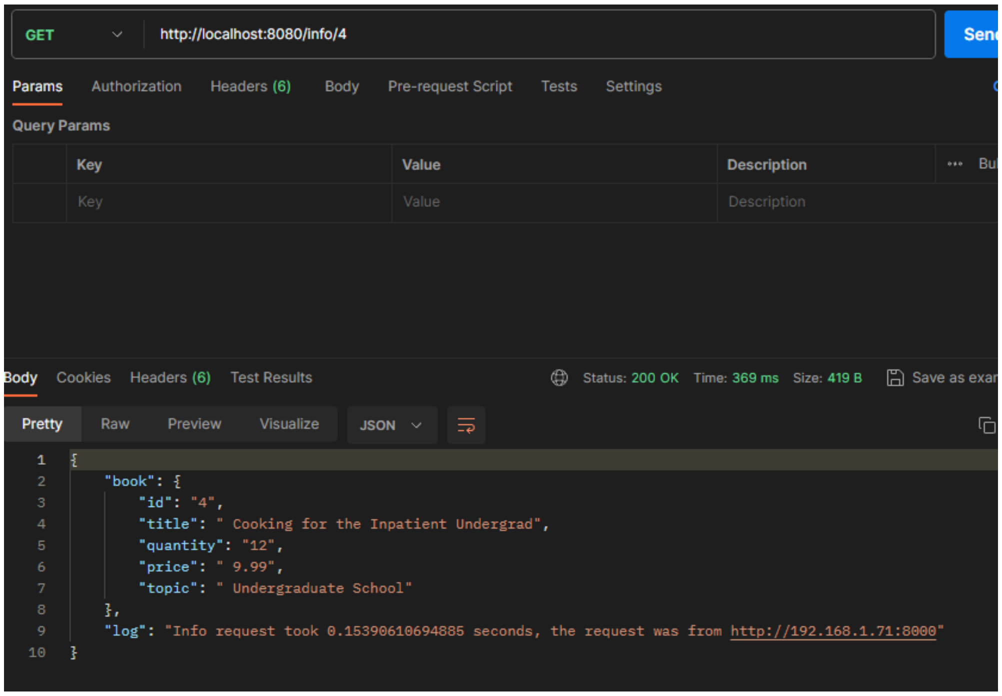


```
{
    "message": "Purchase successful! You bought the book with this id: 6 <br> Purchase request took 0.
        14072513580322 seconds, the request was from http://192.168.1.71:8001"
}
```

Experimental Evaluation and Measurements:

1. The average response time query without caching:

   0.0416926146 seconds, I took avg of 10 requests, here is an example of one:

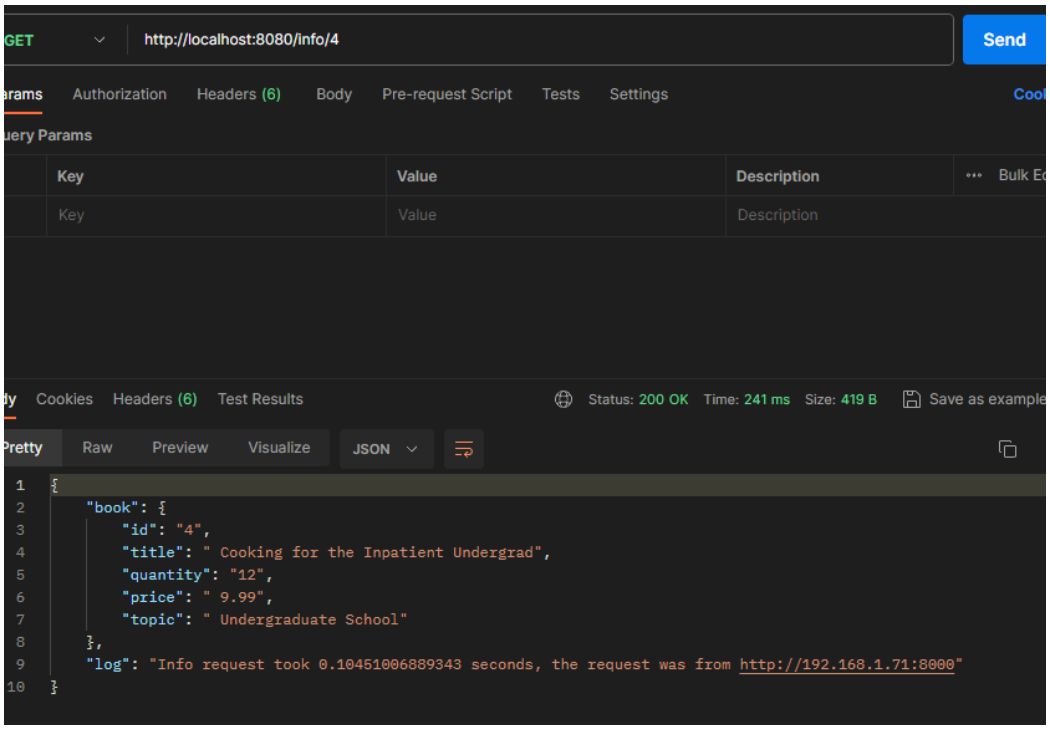


http://localhost:8080/info/4

Status: 200 OK Time: 369 ms Size: 419 B

```
{
    "book": {
        "id": "4",
        "title": " Cooking for the Inpatient Undergrad",
        "quantity": "12",
        "price": " 9.99",
        "topic": " Undergraduate School"
    },
    "log": "Info request took 0.15390610694885 seconds, the request was from http://192.168.1.71:8000"
}
```

The average response time query with caching: 0.0374510288 seconds,



```
GET http://localhost:8080/info/4
```

Key	Value	Description
Key	Value	Description

Status: 200 OK Time: 241 ms Size: 419 B

```
{
    "book": {
        "id": "4",
        "title": " Cooking for the Inpatient Undergrad",
        "quantity": "12",
        "price": " 9.99",
        "topic": " Undergraduate School"
    },
    "log": "Info request took 0.10451006889343 seconds, the request was from http://192.168.1.71:8000"
}
```

This represents a reduction of approximately 10.2% in response time

Design Tradeoffs

- Data Storage:

  Tradeoff: The decision to use a simple text file (CSV) for data storage.

  Rationale: While not as robust as a database, it simplifies the implementation

Possible Improvements and Extensions

- Database Integration:

  Integrating a lightweight database like SQLite for improved data management.

- Security Enhancements:

Implementing HTTPS for secure communication.

Adding authentication and authorization mechanisms for user access.