DOS Project Report Part 2

Student name: Bisan mohammad Joudeh

12113771

Part1 : Replication and Caching

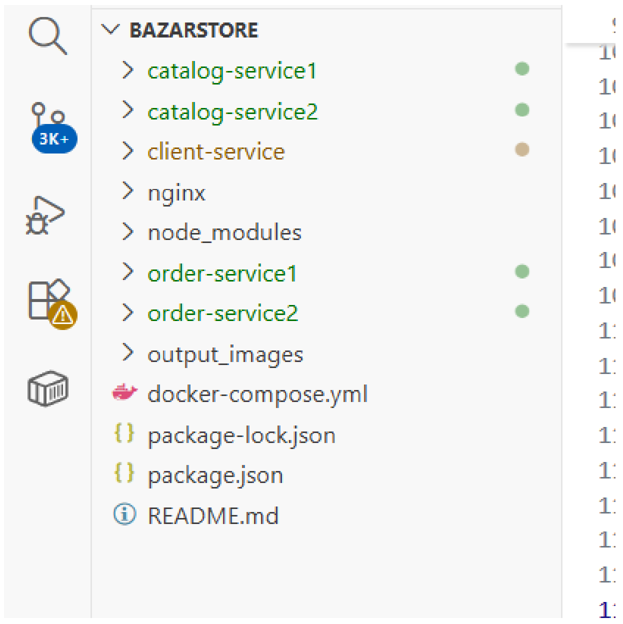
Replication :

I created multiple instances of the Catalog and Order services instead of just one. And each one has its own database.

To distribute the load (Load Balancing).

If one instance fails, the other continues the work (High Availability).

Improved response time when the number of requests increases.



```
BAZARSTORE
  catalog-service1
  catalog-service2
  client-service
  nginx
  node_modules
  order-service1
  order-service2
  output_images
  docker-compose.yml
  package-lock.json
  package.json
  README.md
```

The frontend distributes requests between them using the Round Robin algorithm. Each response returns the name of the version that responded (response from catalog-1).

Caching

I used a front-end cache to store the results of read requests instead of having to return to the server or database each time.

I set the cache size to 50 items. If the cache is full and a new request arrives, I have to delete an old item and replace it with the new one using the LRU (Least Recently Used) algorithm:

If the request is in the cache → Cache Hit

If the request is not in the cache → Cache Miss

Caching was used in:

Read requests (queries) such as:

Fetching a book

Fetching product information

It was not used in write operations (Buy/Update)

Is caching important?

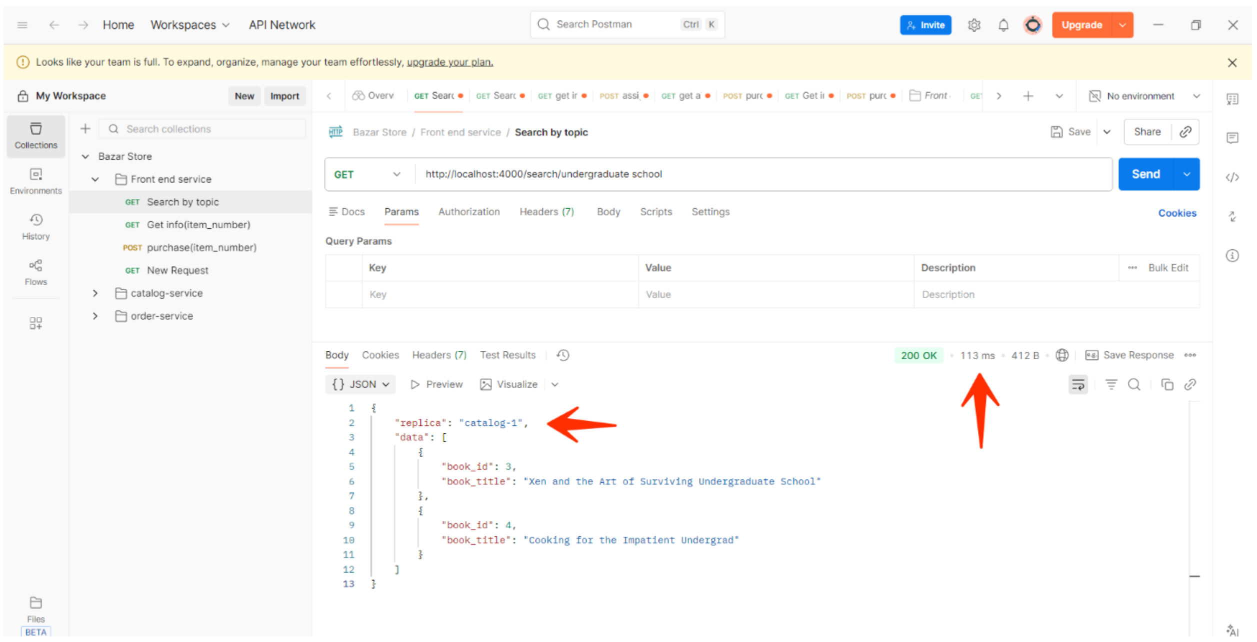
Reducing response time

Reducing the load on the database

Improving overall performance

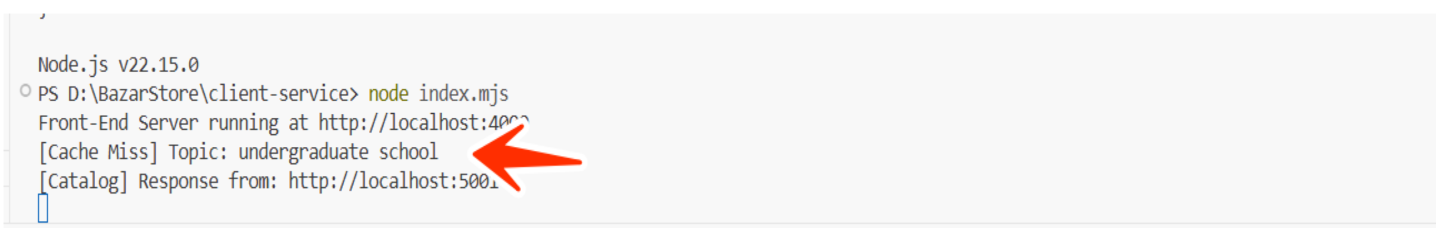
Exampels:

Search book by topic By sending the request to the frontend service:



GET http://localhost:4000/search/undergraduate school

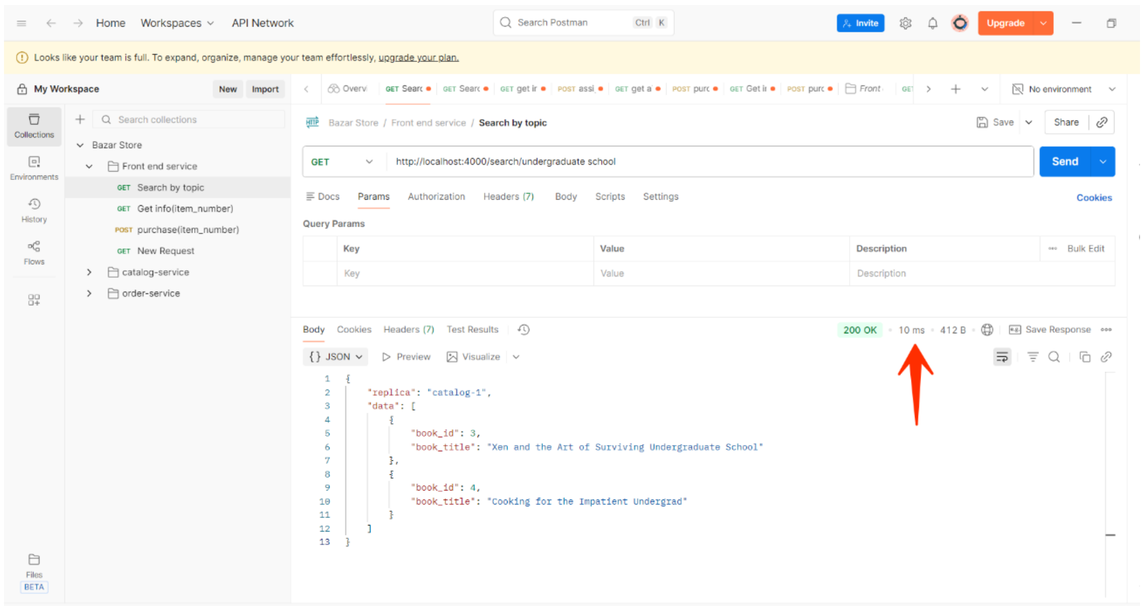
```
{
    "replica": "catalog-1",
    "data": [
        {
            "book_id": 3,
            "book_title": "Xen and the Art of Surviving Undergraduate School"
        },
        {
            "book_id": 4,
            "book_title": "Cooking for the Impatient Undergrad"
        }
    ]
}
```



```
Node.js v22.15.0
PS D:\BazarStore\client-service> node index.mjs
Front-End Server running at http://localhost:40
[Cache Miss] Topic: undergraduate school
[Catalog] Response from: http://localhost:500
```

The response from the first catalog service (load distribution) was a cash miss because it was the first time sent request and the time was 113 milliseconds

The second time the request was sent, there is cache hit and time were 10 milliseconds. As shown:

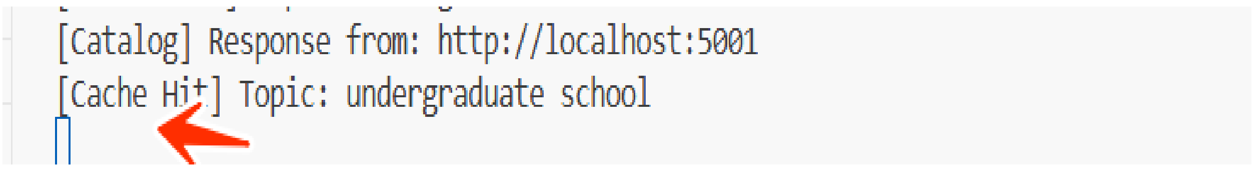


GET http://localhost:4000/search/undergraduate school

200 OK · 10 ms · 412 B

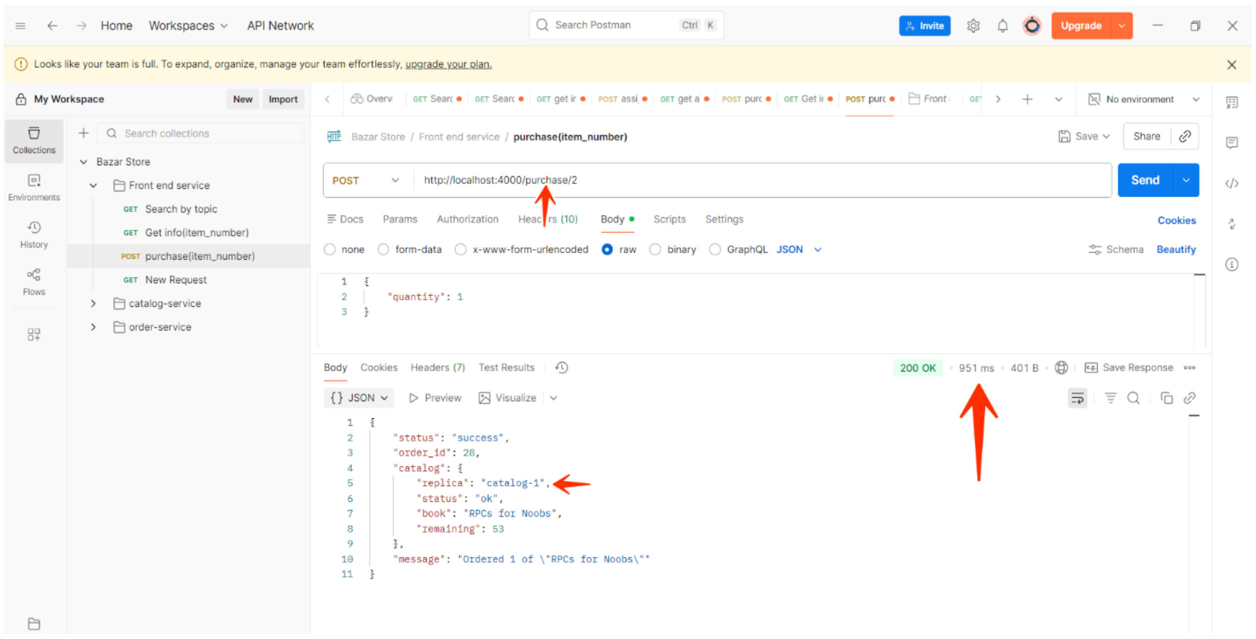
```
{
    "replica": "catalog-1",
    "data": [
        {
            "book_id": 3,
            "book_title": "Xen and the Art of Surviving Undergraduate School"
        },
        {
            "book_id": 4,
            "book_title": "Cooking for the Impatient Undergrad"
        }
    ]
}
```

and the cache is hit



```
[Catalog] Response from: http://localhost:5001
[Cache Hit] Topic: undergraduate school
```

In the purchase order operation, there is no caching mechanism, and therefore the response takes a longer time to be processed, as shown in the results. In addition, the response is served by the primary Order Service replica, not the backup replica.



POST http://localhost:4000/purchase/2

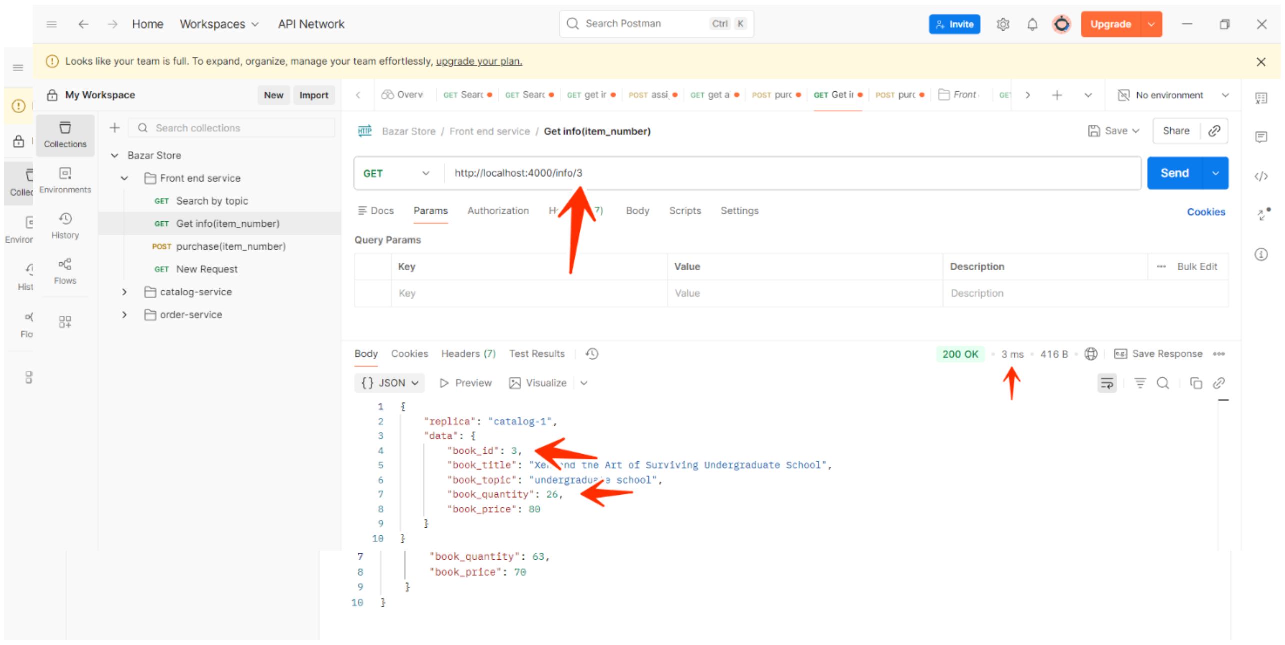
```
{
    "quantity": 1
}
```

200 OK · 951 ms · 401 B

```
{
    "status": "success",
    "order_id": 28,
    "catalog": {
        "replica": "catalog-1",
        "status": "ok",
        "book": "RPCs for Noobs",
        "remaining": 53
    },
    "message": "Ordered 1 of \"RPCs for Noobs\""
}
```

Request	Cache	Response Time (ms)
Query Book	Miss	113
Query Book	Hit	10
Buy Book	No Cache	951

If the new request is sent again, the response will be served by the second catalog replica due to load balancing.



GET http://localhost:4000/info/3 Send

200 OK · 3 ms · 416 B

```
{
    "replica": "catalog-1",
    "data": {
        "book_id": 3,
        "book_title": "Xen and the Art of Surviving Undergraduate School",
        "book_topic": "undergraduate school",
        "book_quantity": 26,
        "book_price": 80
    }
}
        "book_quantity": 63,
        "book_price": 70
    }
}
```

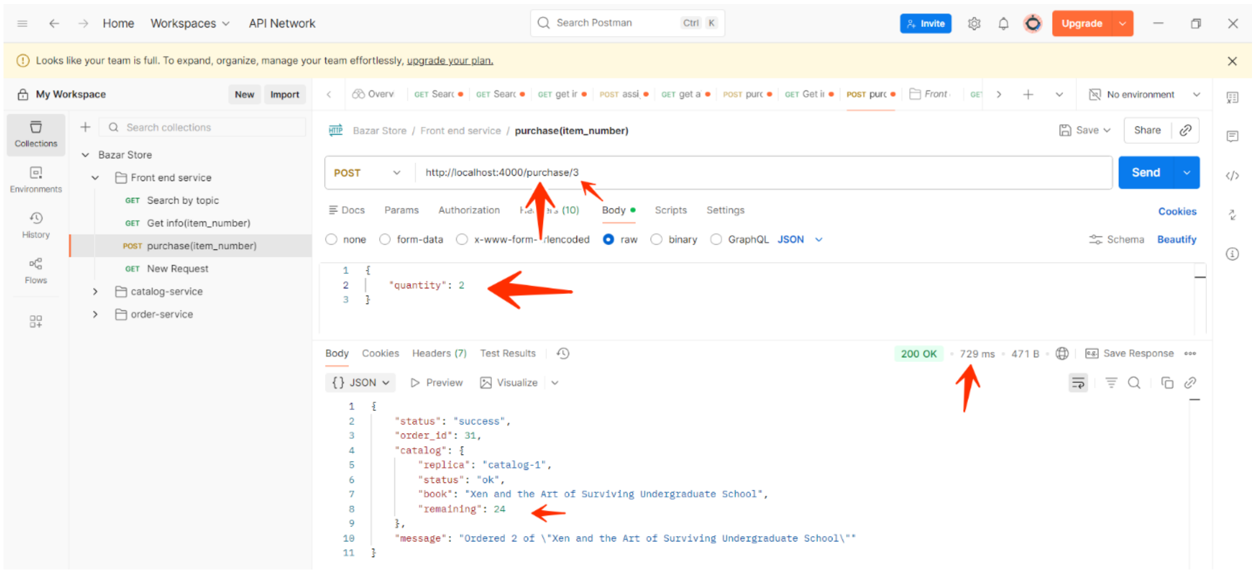
Similarly, for the Order Service, when a request is sent from the user interface to retrieve all orders for the first time, the response is served by the first replica. When the request is sent again, the response is served by the second replica.

Cache Consistency:

The cache is stored in the Front-End, and when a writing operation takes place (buying a book / modifying a quantity / updating a catalog), the old data is deleted from the cache so that the next request brings new data from the real service.

Example:

get info for book id = 3 with cache hit :

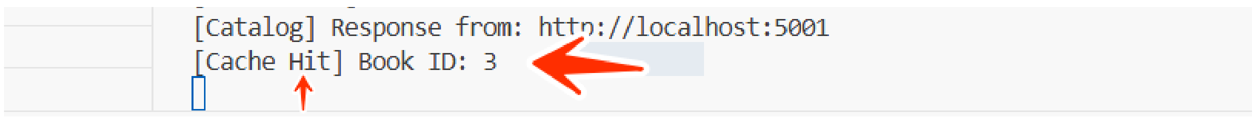


POST http://localhost:4000/purchase/3

```
{
    "quantity": 2
}
```

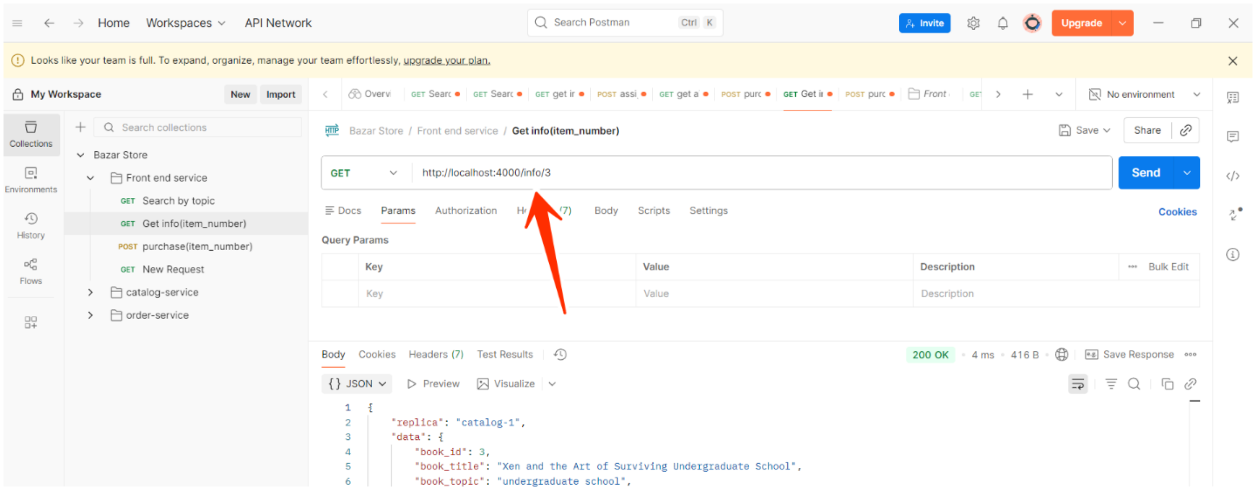
200 OK · 729 ms · 471 B

```
{
    "status": "success",
    "order_id": 31,
    "catalog": {
        "replica": "catalog-1",
        "status": "ok",
        "book": "Xen and the Art of Surviving Undergraduate School",
        "remaining": 24
    },
    "message": "Ordered 2 of \"Xen and the Art of Surviving Undergraduate School\""
}
```



```
[Catalog] Response from: http://localhost:5001
[Cache Hit] Book ID: 3
```

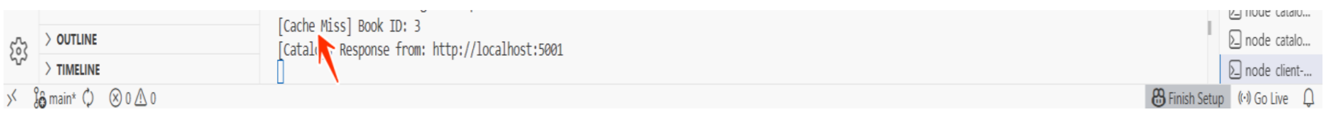
when a purchase operation is performed on a retrieved book, the operation is completed successfully. However, when the same book is requested again after the purchase, the request results in a cache miss because the corresponding entry was removed from the cache to maintain consistency.



```
GET http://localhost:4000/info/3
```

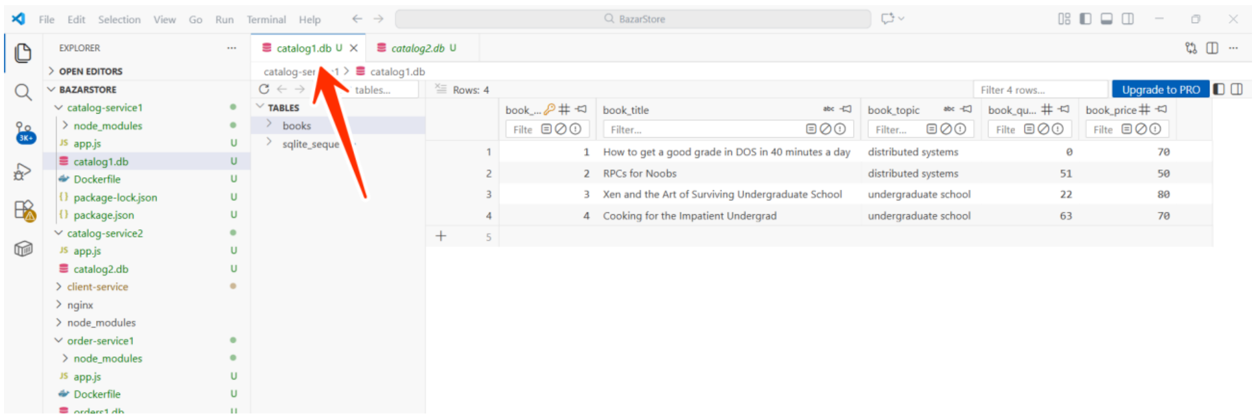
200 OK · 4 ms · 416 B

```
{
    "replica": "catalog-1",
    "data": {
        "book_id": 3,
        "book_title": "Xen and the Art of Surviving Undergraduate School",
        "book_topic": "undergraduate school",
```



```
[Cache Miss] Book ID: 3
[Catal  Response from: http://localhost:5001
```

Data stored in the database is maintained consistently between the catalog service and order service versions. When any update occurs, such as a purchase or order creation, the change is first applied to the primary database and then generalized to the secondary version.



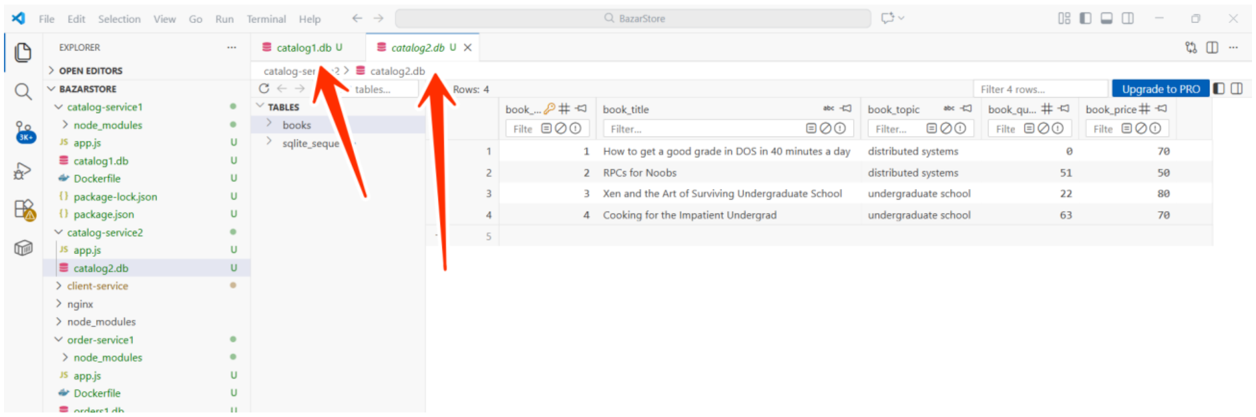
catalog-service1
- node_modules
- app.js U
- catalog1.db U
- Dockerfile U
- package-lock.json U
- package.json U

catalog-service2
- app.js U
- catalog2.db U

client-service
nginx
node_modules
order-service1
- node_modules
- app.js U
- Dockerfile U

Rows: 4

	book_...	book_title	book_topic	book_qu...	book_price
1	1	How to get a good grade in DOS in 40 minutes a day	distributed systems	0	70
2	2	RPCs for Noobs	distributed systems	51	50
3	3	Xen and the Art of Surviving Undergraduate School	undergraduate school	22	80
4	4	Cooking for the Impatient Undergrad	undergraduate school	63	70
5					



catalog-service1
- node_modules
- app.js U
- catalog1.db U
- Dockerfile U
- package-lock.json U
- package.json U

catalog-service2
- app.js U
- catalog2.db U

client-service
nginx
node_modules
order-service1
- node_modules
- app.js U
- Dockerfile U

Rows: 4

	book_...	book_title	book_topic	book_qu...	book_price
1	1	How to get a good grade in DOS in 40 minutes a day	distributed systems	0	70
2	2	RPCs for Noobs	distributed systems	51	50
3	3	Xen and the Art of Surviving Undergraduate School	undergraduate school	22	80
4	4	Cooking for the Impatient Undergrad	undergraduate school	63	70
5					