

# DOS Project Report Part 2

Student name: Bisan mohammad Joudeh

12113771

## Part1 : Replication and Caching

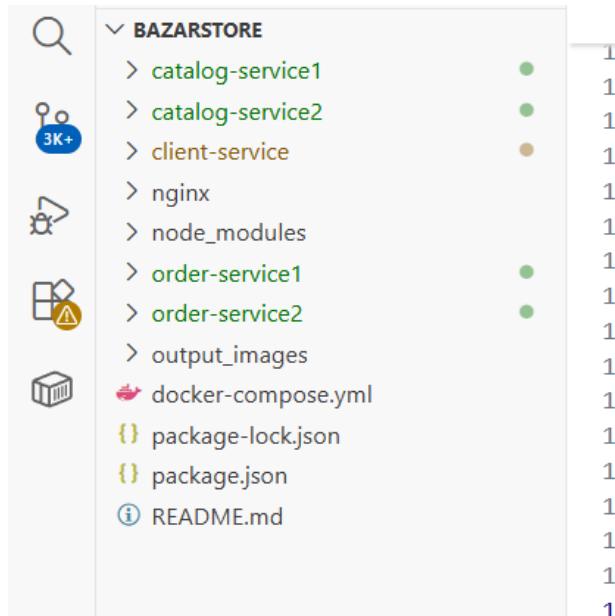
Replication :

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.



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

## Exempels:

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

The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists collections, environments, and flows. In the center, a 'Bazar Store / Front end service / Search by topic' collection is selected. A GET request is shown with the URL `http://localhost:4000/search/undergraduate school`. The 'Params' tab is active, showing a single parameter 'Key' with value 'Value'. The 'Body' tab displays a JSON response with an arrow pointing to the 'replica' field. The response body is:

```
1 {  
2   "replica": "catalog-1", ←  
3     "data": [  
4       {  
5         "book_id": 3,  
6         "book_title": "Xen and the Art of Surviving Undergraduate School"  
7       },  
8       {  
9         "book_id": 4,  
10        "book_title": "Cooking for the Impatient Undergrad"  
11      }  
12    ]  
13 }
```

The status bar at the bottom right indicates a 200 OK response with 113 ms latency and 412 B size. An arrow points upwards from the JSON response towards the status bar.

```
Node.js v22.15.0  
PS D:\BazarStore\client-service> node index.mjs  
Front-End Server running at http://localhost:4000  
[Cache Miss] Topic: undergraduate school ←  
[Catalog] Response from: http://localhost:5001
```

A red arrow points from the terminal output to the '[Cache Miss]' message.

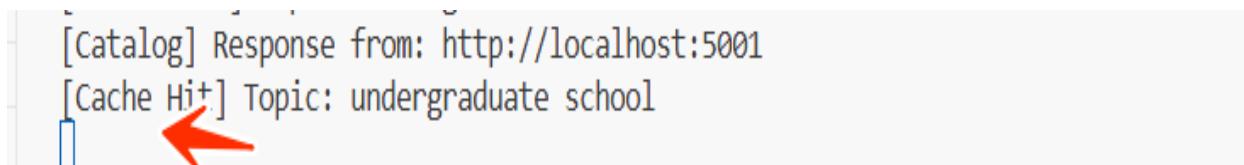
The response from the first catalog service (load distribution) was a cash miss because it was the first time sent request and the time was **113 milliseconds**

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

The screenshot shows the Postman interface. In the left sidebar, under 'My Workspace', there is a collection named 'Bazar Store' which contains a folder 'Front end service' with a 'GET Search by topic' request. The request URL is 'http://localhost:4000/search/undergraduate school'. The response status is '200 OK' with a time of '10 ms'. The response body is a JSON object:

```
{  
  "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



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

```

1 {
2   "status": "success",
3   "order_id": 28,
4   "catalog": {
5     "replica": "catalog_4",
6     "status": "ok",
7     "book": "RPCs for Noobs",
8     "remaining": 53
9   },
10  "message": "Ordered 1 of \"RPCs for Noobs\""
11 }

```

200 OK | 951 ms | 401 B | Save Response ↗

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.

The screenshot shows the Postman interface with a collection named 'Bazar Store' containing a 'Front end service' collection. Under 'Front end service', there is a 'GET Get info(item\_number)' request. The request URL is 'http://localhost:4000/info/3'. The response status is '200 OK' with a response time of 3 ms and a size of 416 B. The response body is a JSON object:

```

1  {
2   "replica": "catalog-1",
3   "data": [
4     {
5       "book_id": 3,
6       "book_title": "Xen and the Art of Surviving Undergraduate School",
7       "book_topic": "undergradu_3 school",
8       "book_quantity": 26,
9       "book_price": 88
10    }
11    {
12      "book_quantity": 63,
13      "book_price": 78
14    }
15  ]
16 }

```

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:

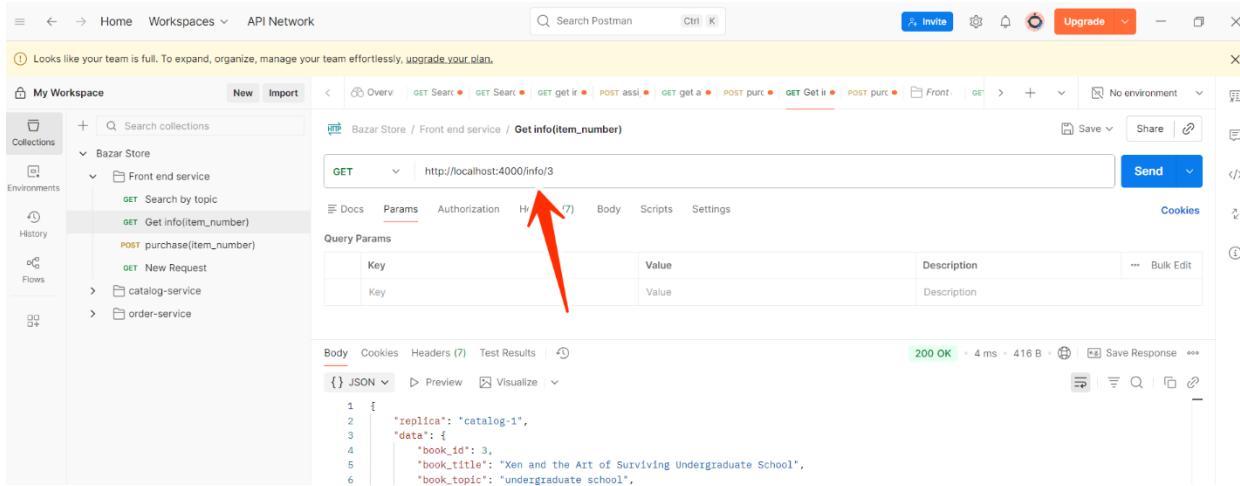
get info for book id = 3 with cache hit :

The screenshot shows the Postman interface. A POST request is made to `http://localhost:4000/purchase/3` with a body containing `{"quantity": 2}`. The response is a 200 OK status with a JSON body:

```
1 {  
2   "status": "success",  
3   "order_id": 31,  
4   "catalog": [  
5     {"replica": "catalog-1",  
6     "status": "ok",  
7     "book": "Xen and the Art of Surviving Undergraduate School",  
8     "remaining": 24  
9   ],  
10  "message": "Ordered 2 of \'Xen and the Art of Surviving Undergraduate School\'"  
11 }
```

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

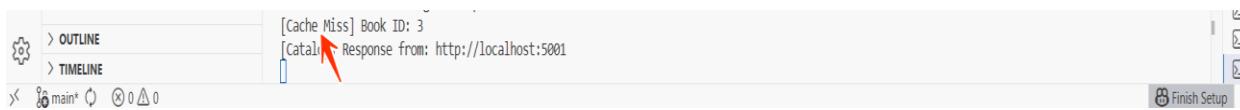
when a purchase operation is performed on a retrieved book, the operation is completed successfully. However, when the same book is requested again after the purchase, the request results in a cache miss because the corresponding entry was removed from the cache to maintain consistency.



The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists collections like 'Bazar Store' and environments like 'Front end service'. In the center, a request for 'GET /info/item\_number' is displayed. A red arrow points to the 'Headers' tab, which is currently selected. Below it, the 'Body' tab is selected, showing a JSON response:

```
1 {  
2   "replica": "catalog-1",  
3   "data": {  
4     "book_id": 3,  
5     "book_title": "Zen and the Art of Surviving Undergraduate School",  
6     "book_topic": "undergraduate school",  
7   }  
}
```

At the bottom, the status bar shows '200 OK' with a response time of '4 ms' and a size of '416 B'. On the right, there are tabs for Body, Cookies, Headers, and Test Results, along with a preview of the JSON response.

The screenshot shows the Catalog UI. A red arrow points to the status message '[Cache Miss] Book ID: 3'. Below it, another message '[Catalog] Response from: http://localhost:5001' is visible. The interface includes sections for OUTLINE and TIMELINE, and buttons for Finish Setup and Go Live.

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.

