Bazar.com

Part2

SUBMITTED TO: Dr.Samer Arandi

Reem AbdAl Raheem Hasan << 12112527>>

Farah Faisal Ahmad << 12112820>>

Github link:

https://github.com/ReemHasan31/my-Book-app

Introduciton

In this lab, I redesigned the online bookstore **Bazar.com**, originally developed in Lab 1, with the aim of enhancing request processing time. The system had begun to struggle under the increasing load caused by growing customer demand.
To address these challenges, I improved the system's scalability and efficiency by introducing **replication**, **caching techniques**, and several additional architectural enhancements.
The educational objectives of this project focus on helping students understand **multi-tier web architecture**, **microservices principles**, and **containerization using Docker**.
This report outlines the modifications applied to the system to support replication, caching, and consistency mechanisms. It also explains the technologies that contribute to reducing response time while increasing overall system capacity.

Updated Architecture with Replication and Caching

To enhance performance, I implemented a multi-tier architecture that includes the following components:

- ✓ Front-End Server with In-Memory Cache: The front-end now includes an in memory cache to store frequently accessed catalog data and recent orders.

- ✓ Replicated Catalog and Order Services: I added multiple replicas of the catalog and order services to improve redundancy and decrease response times by distributing the load.

Implementation Details

In-Memory Caching

To improve response time, an in-memory caching mechanism was added to the front-end server. This cache keeps recently accessed data, reducing the need to repeatedly query the database for identical requests.

- **Cache Design:**
  The cache is structured as a key-value store, where frequently accessed items and recent orders are stored directly in memory.

```
54  // Cache helpers
55  function getFromCache(key) {
56    return cache[key] ? cache[key].data : null;
57  }
58
59  function setCache(key, data) {
60    cache[key] = { data };
61  }
62
63  function invalidateCache(key) {
64    if (cache[key]) delete cache[key];
65    console.log(chalk.yellowBright(`Cache cleared successfully for "${key}"`));
66
67  }
```

Data Replication

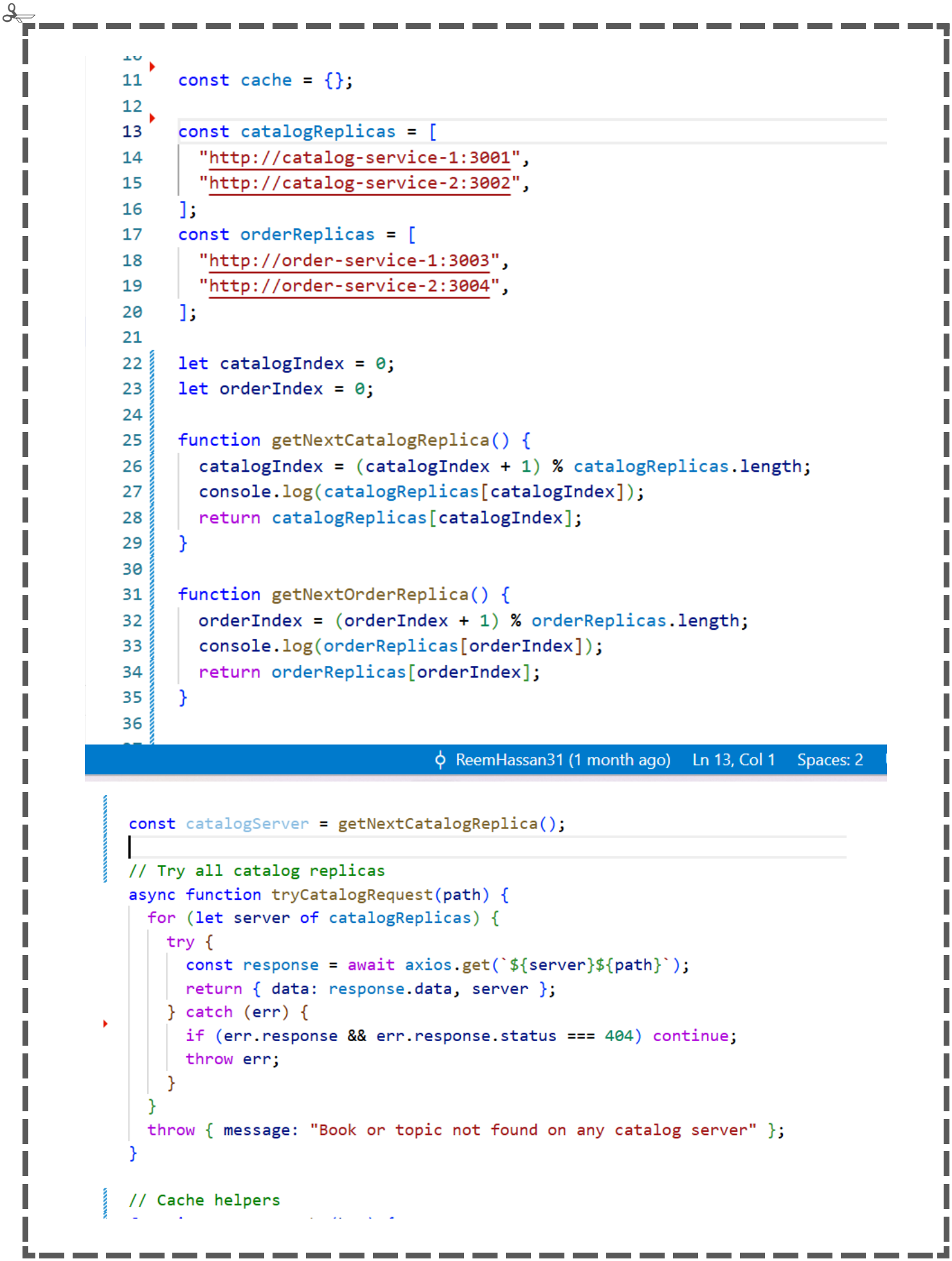
To enhance system availability and reduce latency, data replication was introduced. Multiple instances of both the catalog service and the order service were deployed, allowing the front-end server to distribute incoming requests across these replicated components.

- **Replication Strategy:**
  A round-robin approach was adopted to direct requests evenly among the available replicas.

- **Load Balancing:**
  The front-end server forwards requests to the replicated catalog and order services, ensuring that the workload is distributed efficiently.

- **High Availability and Fault Tolerance:**
  By combining replication and caching, the system maintains high availability. Users can continue accessing catalog or order services even if one or more replicas fail, and cached data helps reduce the impact of temporary outages.

- **Scalability:**
  The replication and load balancing design allows the system to scale horizontally. New replicas can be added to handle increasing traffic, and the round-robin mechanism automatically incorporates them into the request distribution process without major code changes.



```
const cache = {};

const catalogReplicas = [
  "http://catalog-service-1:3001",
  "http://catalog-service-2:3002",
];
const orderReplicas = [
  "http://order-service-1:3003",
  "http://order-service-2:3004",
];

let catalogIndex = 0;
let orderIndex = 0;

function getNextCatalogReplica() {
  catalogIndex = (catalogIndex + 1) % catalogReplicas.length;
  console.log(catalogReplicas[catalogIndex]);
  return catalogReplicas[catalogIndex];
}

function getNextOrderReplica() {
  orderIndex = (orderIndex + 1) % orderReplicas.length;
  console.log(orderReplicas[orderIndex]);
  return orderReplicas[orderIndex];
}
```

ReemHassan31 (1 month ago) Ln 13, Col 1 Spaces: 2

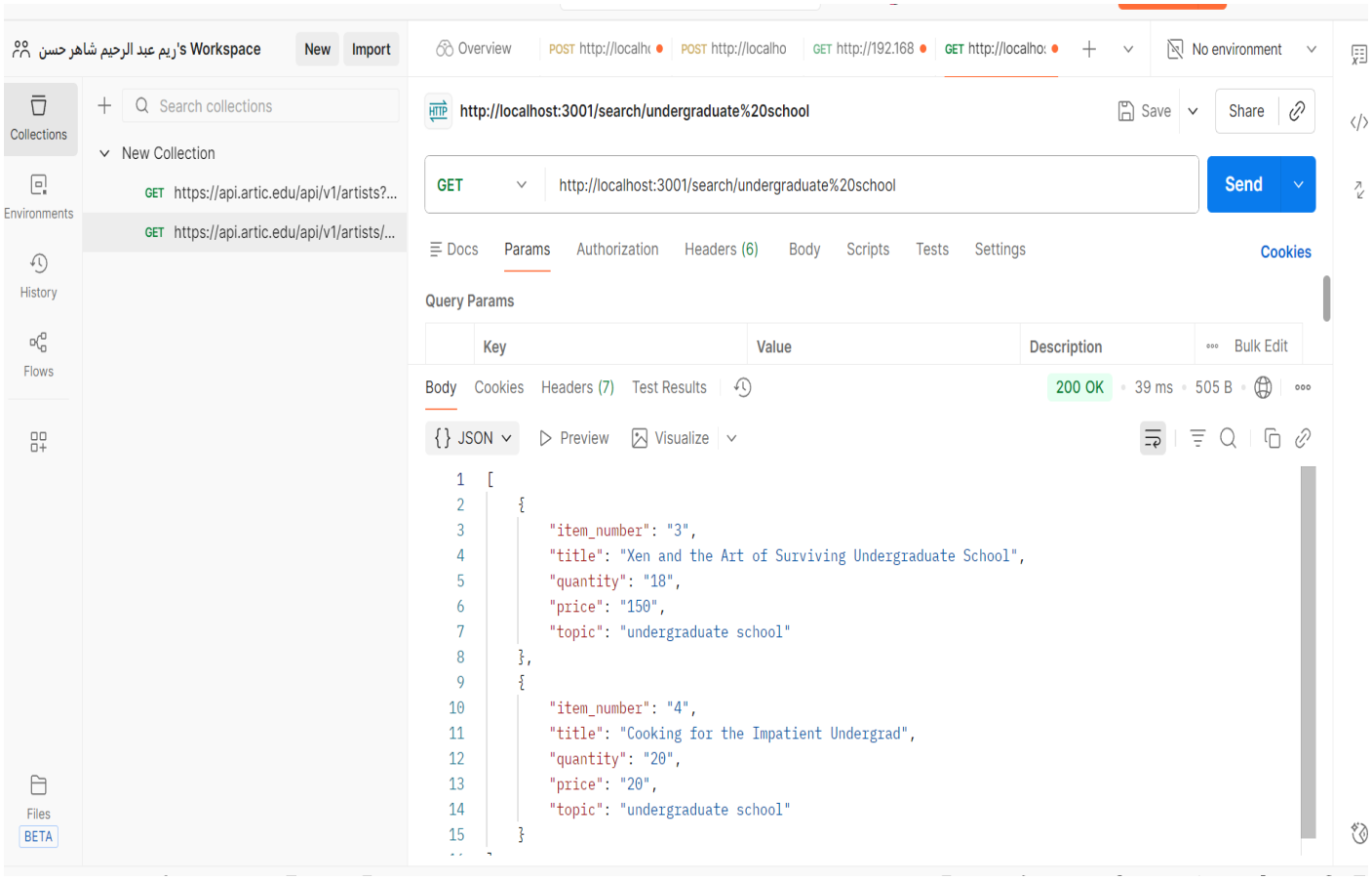
```
const catalogServer = getNextCatalogReplica();

// Try all catalog replicas
async function tryCatalogRequest(path) {
  for (let server of catalogReplicas) {
    try {
      const response = await axios.get(`${server}${path}`);
      return { data: response.data, server };
    } catch (err) {
      if (err.response && err.response.status === 404) continue;
      throw err;
    }
  }
  throw { message: "Book or topic not found on any catalog server" };
}

// Cache helpers
```

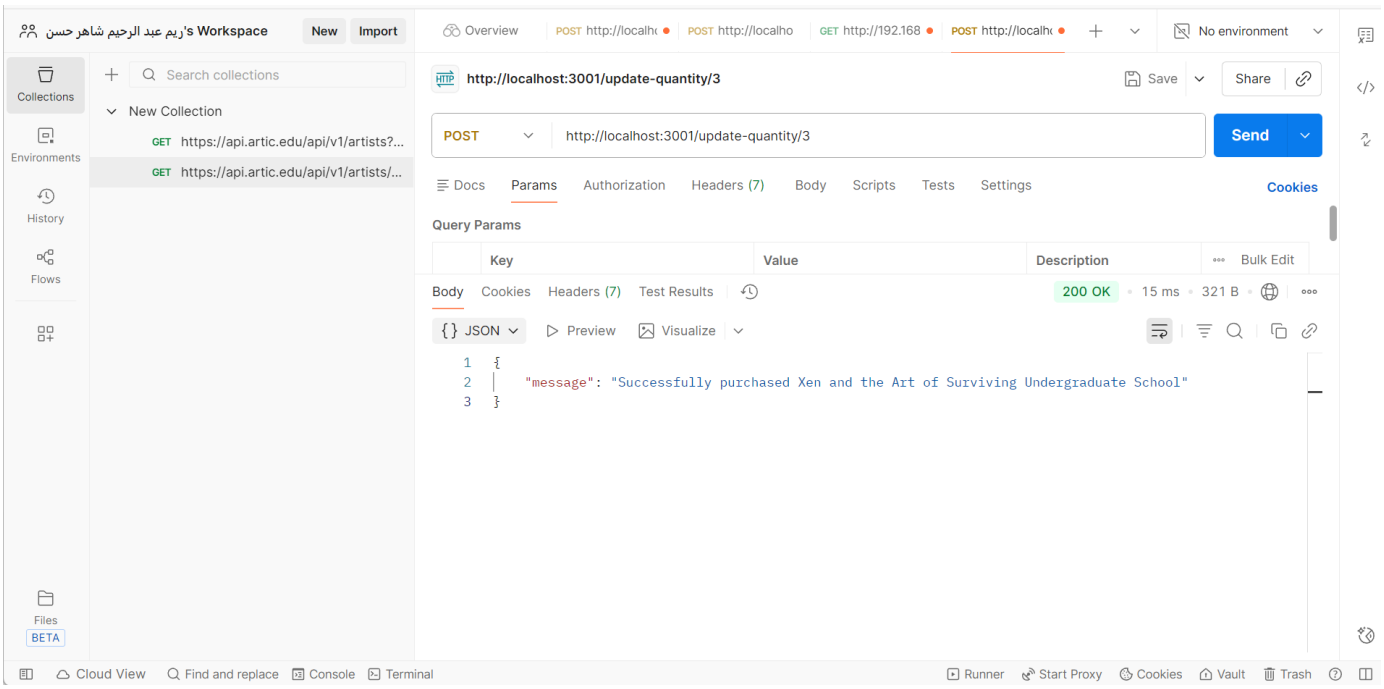
Testing and Results(Performance Analysis)

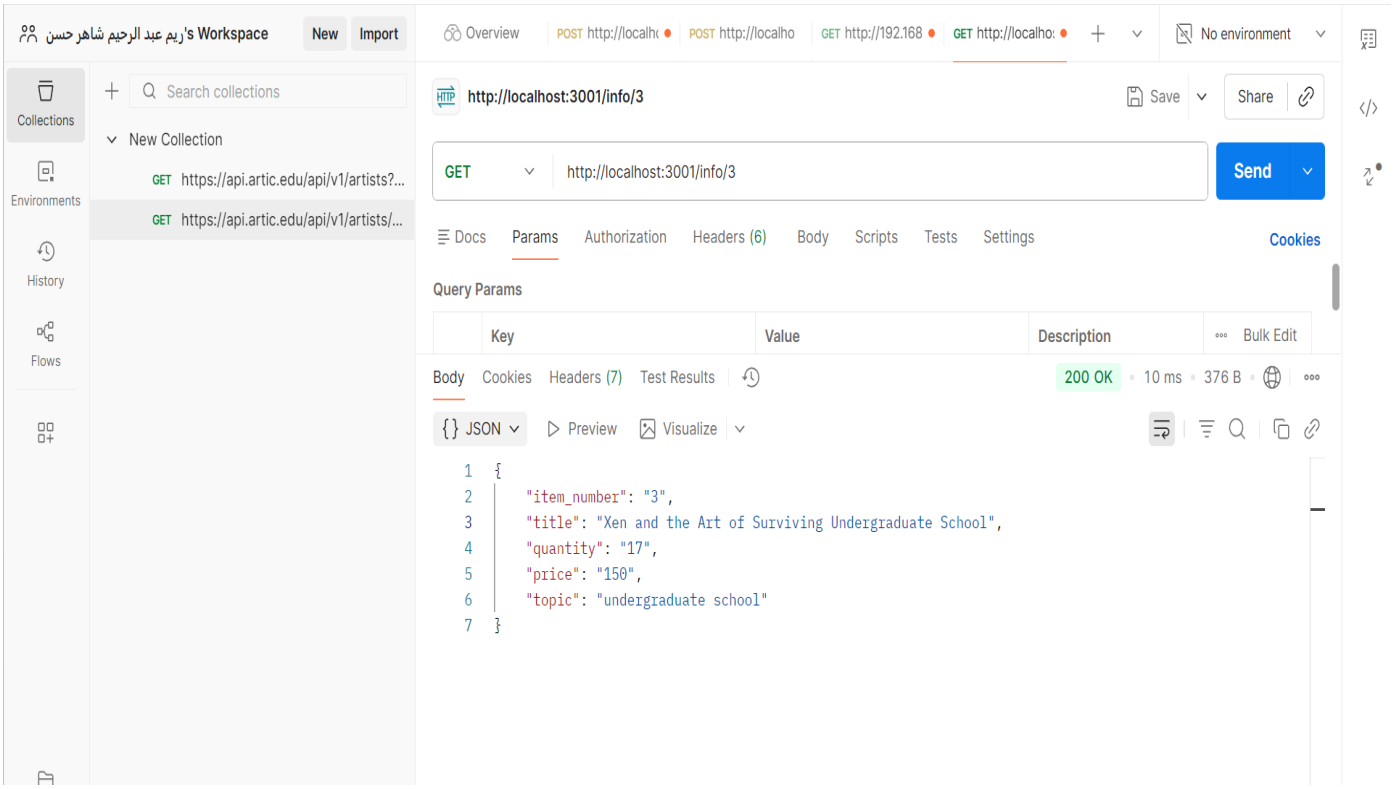
➢ Latency Improvement

The system's response time was evaluated both before and after introducing caching and replication. The experimental results show a clear and significant reduction in average response time after these optimizations were applied. This improvement demonstrates the effectiveness of caching and replication in reducing request latency and enhancing overall system performance.

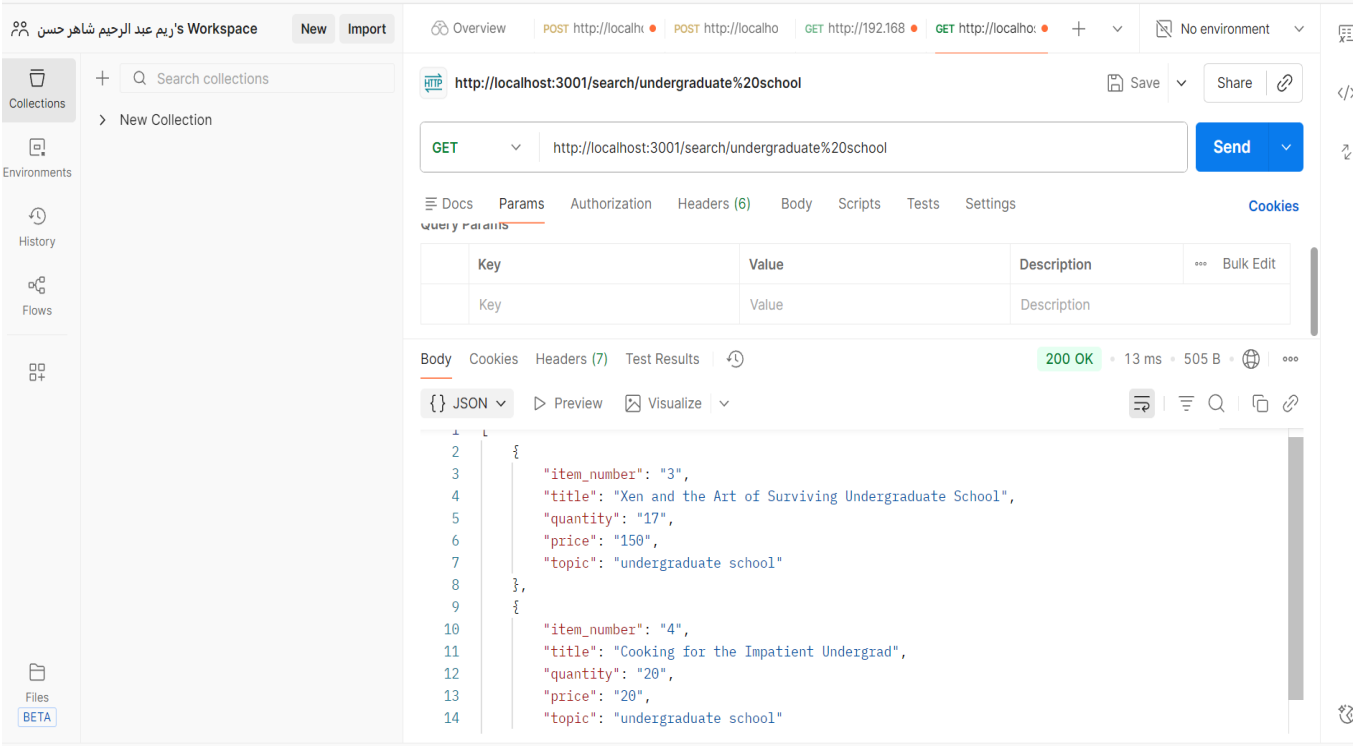
Without using cache:

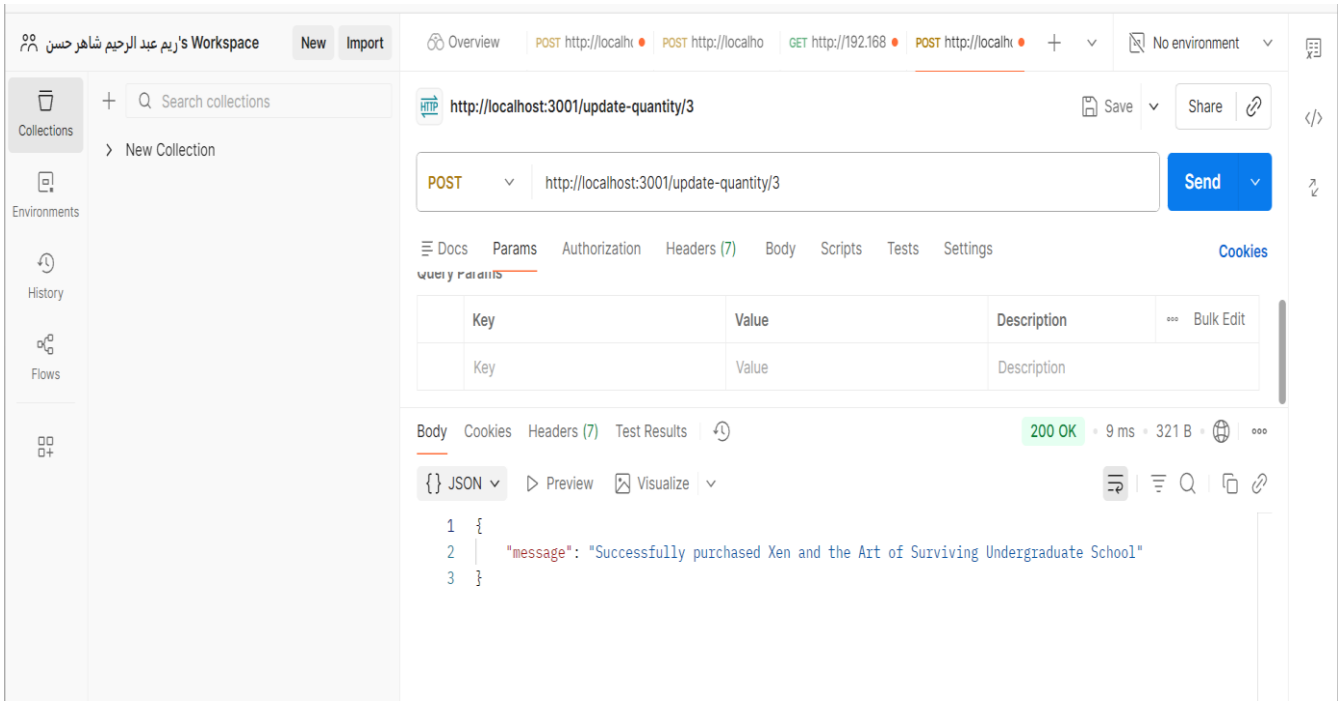


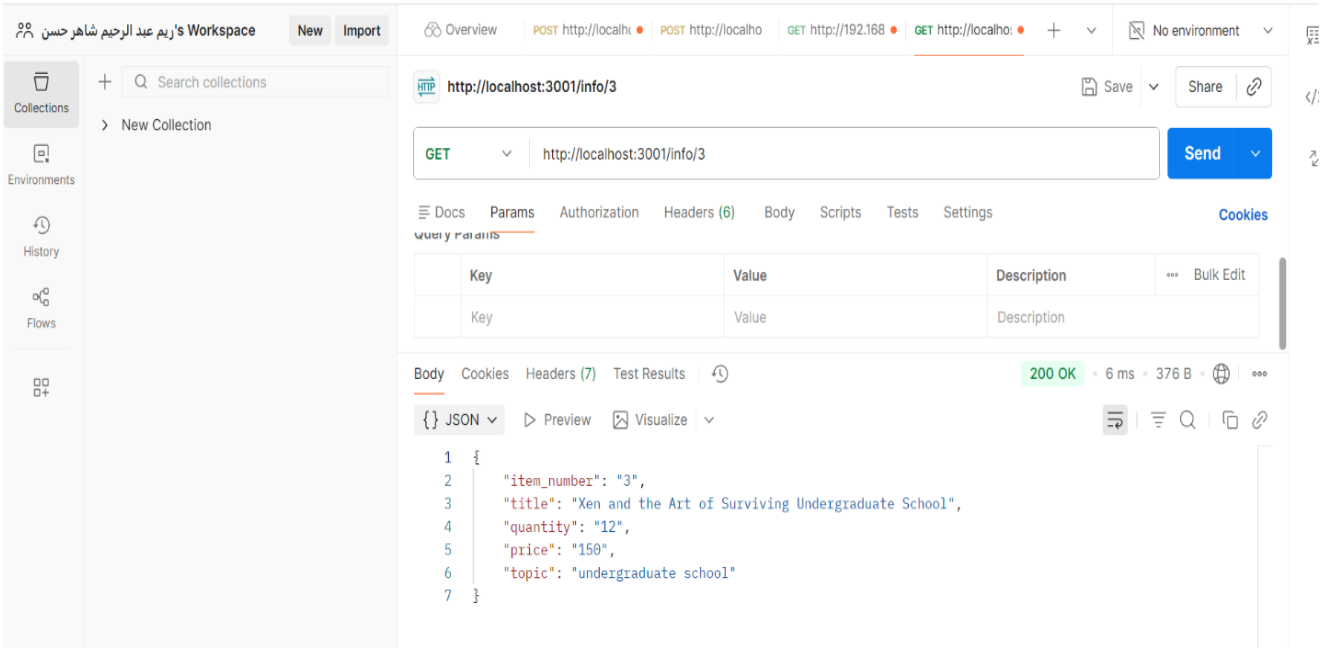

ريم عبد الرحيم شاهر حسن's Workspace
New
Import
Collections
Environments
History
Flows
Files
BETA
Search collections
New Collection
GET https://api.artic.edu/api/v1/artists?...
GET https://api.artic.edu/api/v1/artists/...
Overview
POST http://localh
POST http://localho
GET http://192.168
POST http://localh
No environment
http://localhost:3001/update-quantity/3
Save
Share
POST
http://localhost:3001/update-quantity/3
Send
Docs
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
15 ms
321 B
{} JSON
Preview
Visualize
1 {
2 "message": "Successfully purchased Xen and the Art of Surviving Undergraduate School"
3 }
Cloud View
Find and replace
Console
Terminal
Runner
Start Proxy
Cookies
Vault
Trash


ريم عبد الرحيم شاهر حسن's Workspace
New
Import
Collections
Environments
History
Flows
Search collections
New Collection
GET https://api.artic.edu/api/v1/artists?...
GET https://api.artic.edu/api/v1/artists/...
Overview
POST http://localh
POST http://localho
GET http://192.168
GET http://localho:
No environment
http://localhost:3001/info/3
Save
Share
GET
http://localhost:3001/info/3
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
10 ms
376 B
{} JSON
Preview
Visualize
1 {
2 "item_number": "3",
3 "title": "Xen and the Art of Surviving Undergraduate School",
4 "quantity": "17",
5 "price": "150",
6 "topic": "undergraduate school"
7 }

With using cache







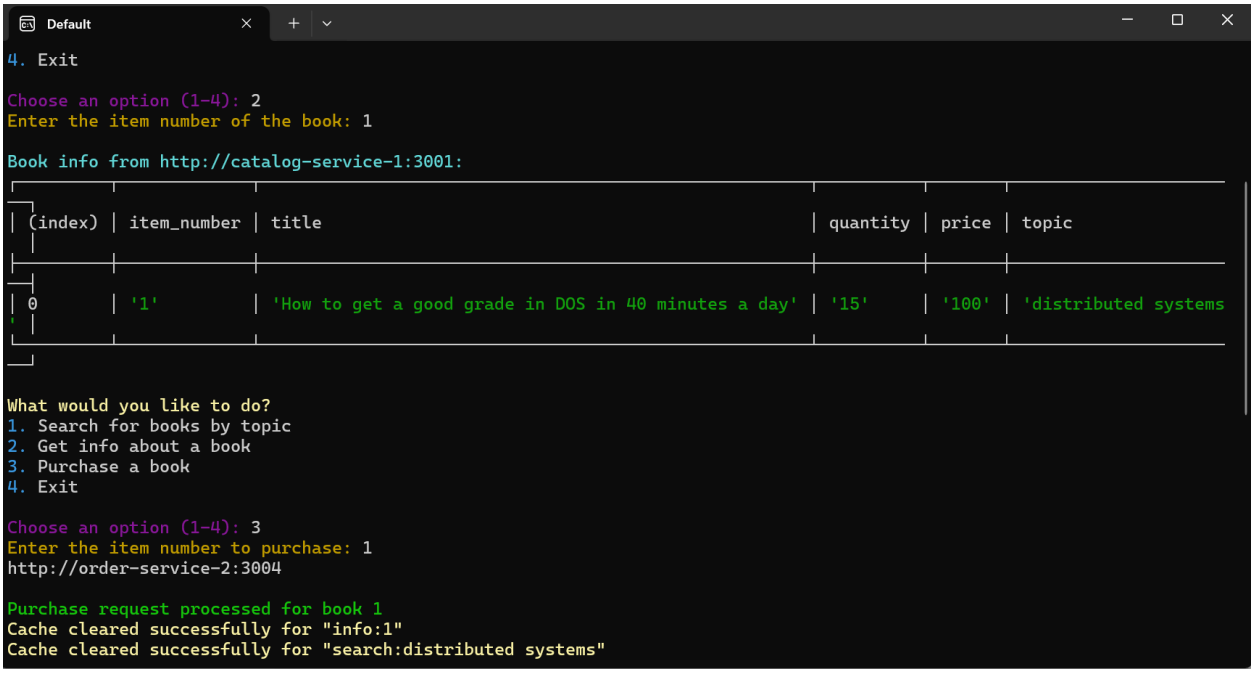
The table that shows the differences

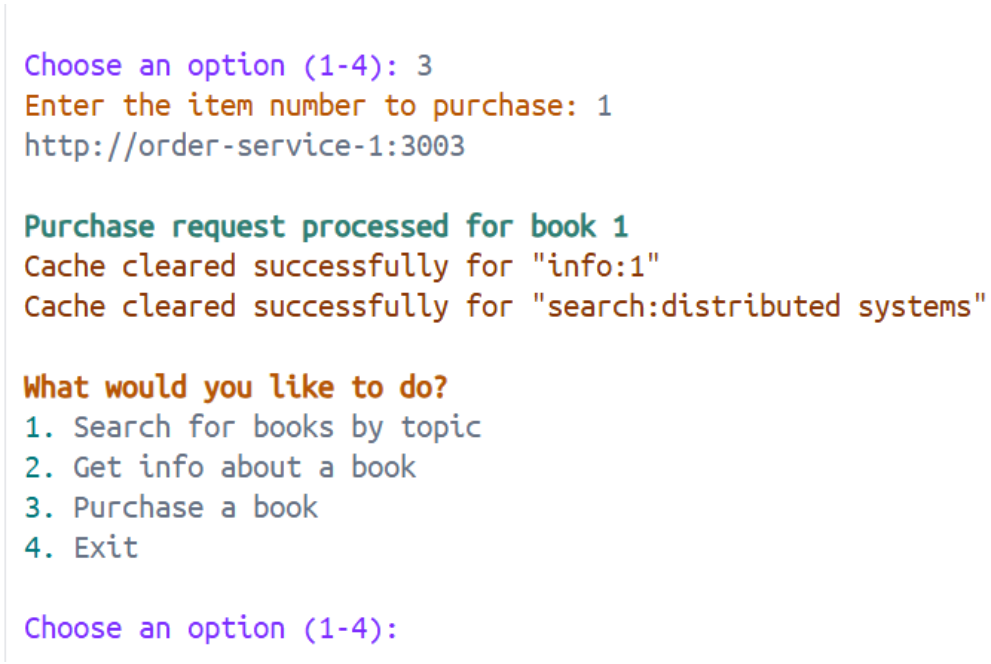
Feature	Response Time Before (ms)	Response Time After (ms)
Search by topic – Catalog	39	13
Search by ID – Catalog	10	6
Purchase – Order	15	9

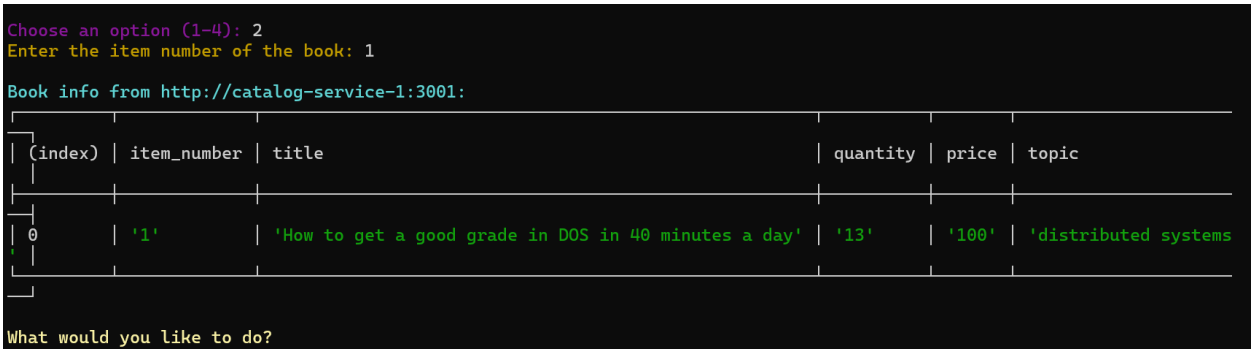
Consistency Testing

To verify data consistency, multiple concurrent requests were simulated. For instance, when the stock of a book was updated in one replica, all other replicas reflected the change shortly afterward, ensuring data integrity. This was tested by opening multiple client interfaces simultaneously and performing operations on the same book, confirming that all replicas stayed


```
Purchase request processed for book 1
Cache cleared successfully for "info:1"
Cache cleared successfully for "search:distributed systems"

What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit
```


Default
4. Exit
Choose an option (1-4): 2
Enter the item number of the book: 1
Book info from http://catalog-service-1:3001:
(index) | item_number | title | quantity | price | topic
0 | '1' | 'How to get a good grade in DOS in 40 minutes a day' | '15' | '100' | 'distributed systems
What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit
Choose an option (1-4): 3
Enter the item number to purchase: 1
http://order-service-2:3004
Purchase request processed for book 1
Cache cleared successfully for "info:1"
Cache cleared successfully for "search:distributed systems"


Choose an option (1-4): 3
Enter the item number to purchase: 1
http://order-service-1:3003
Purchase request processed for book 1
Cache cleared successfully for "info:1"
Cache cleared successfully for "search:distributed systems"
What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit
Choose an option (1-4):


Choose an option (1-4): 2
Enter the item number of the book: 1
Book info from http://catalog-service-1:3001:
(index) | item_number | title | quantity | price | topic
0 | '1' | 'How to get a good grade in DOS in 40 minutes a day' | '13' | '100' | 'distributed systems
What would you like to do?

Optional: Docker Containerization

To simplify deployment and ensure consistent environments, each service was packaged into its own Docker container. This approach allows the application to run in isolated, lightweight containers, which makes deployment, scaling, and management much easier.

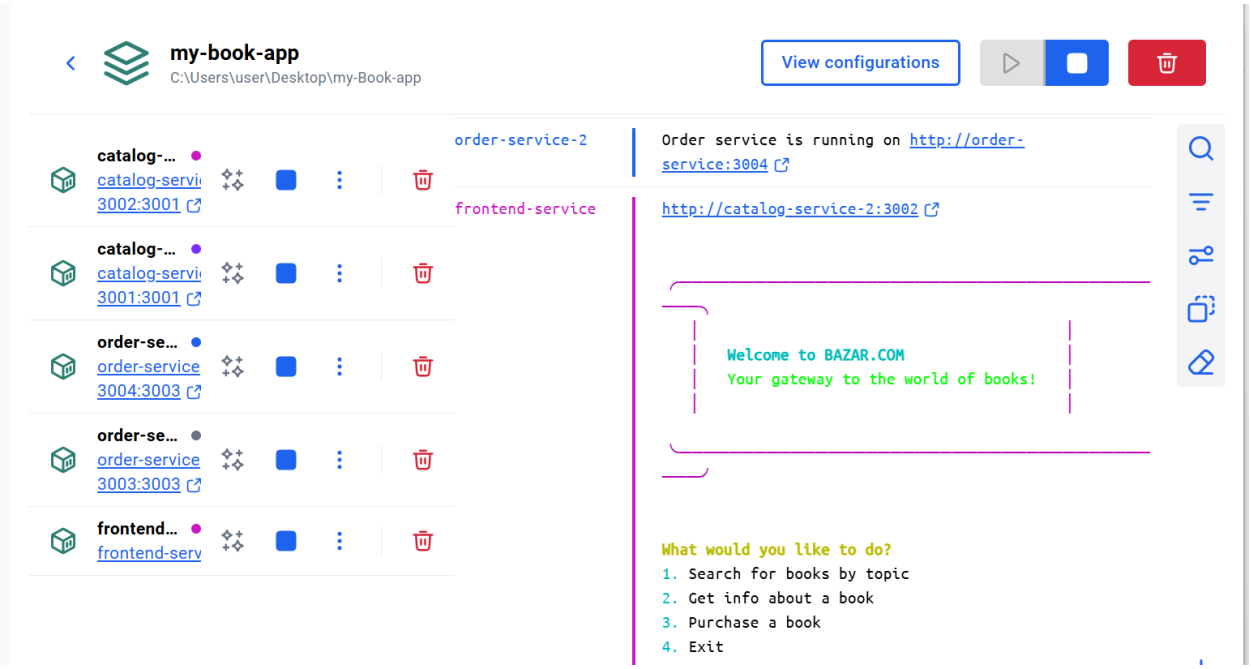
Docker-Compose Setup:

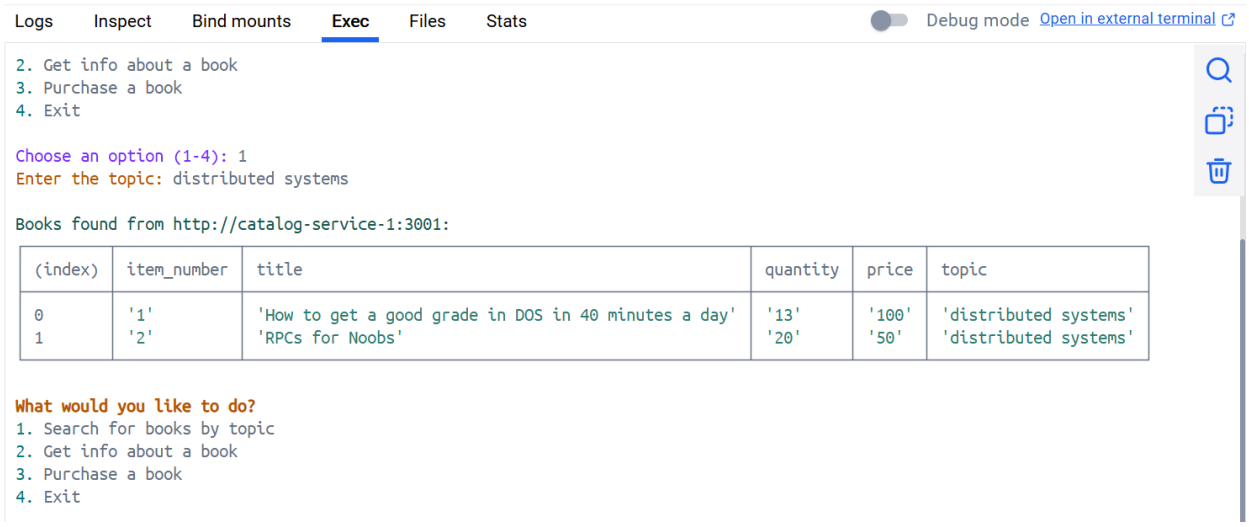
The docker-compose.yml file orchestrates all services, including the catalog and order services, as well as the front-end, ensuring they communicate seamlessly over a shared network.

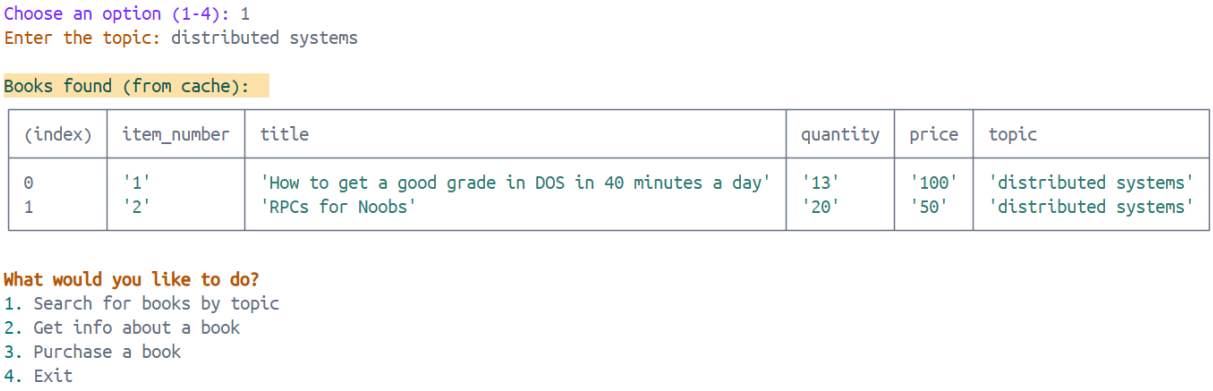

client.js 1M, M
catalog.csv M
docker-compose.yml M
docker-compose.yml > { } services > { } order-service-2 > [ ] depends_on > abc 1
services:
Run Service
order-service-1:
image: order-service
container_name: order-service-1
build:
context: ./order-service
ports:
- "3003:3003"
environment:
- PORT=3003
networks:
- network1
depends_on:
- catalog-service-1
- catalog-service-2
Run Service
order-service-2:
image: order-service
container_name: order-service-2
ports:
- "3004:3003"
environment:
- PORT=3004
networks:
- network1
depends_on:
- catalog-service-1
depends_on:
- catalog-service-1
- catalog-service-2
Run Service
frontend-service:
image: frontend-service
container_name: frontend-service
build:
context: ./Frontend
networks:
- network1
depends_on:
- order-service-1
- order-service-2
- catalog-service-1
- catalog-service-2
stdin_open: true
tty: true
networks:
network1:
driver: bridge
ReemHassan31

Result

The images below demonstrate that incoming requests are distributed among different replicas, ensuring that data consistency is preserved across all containers.







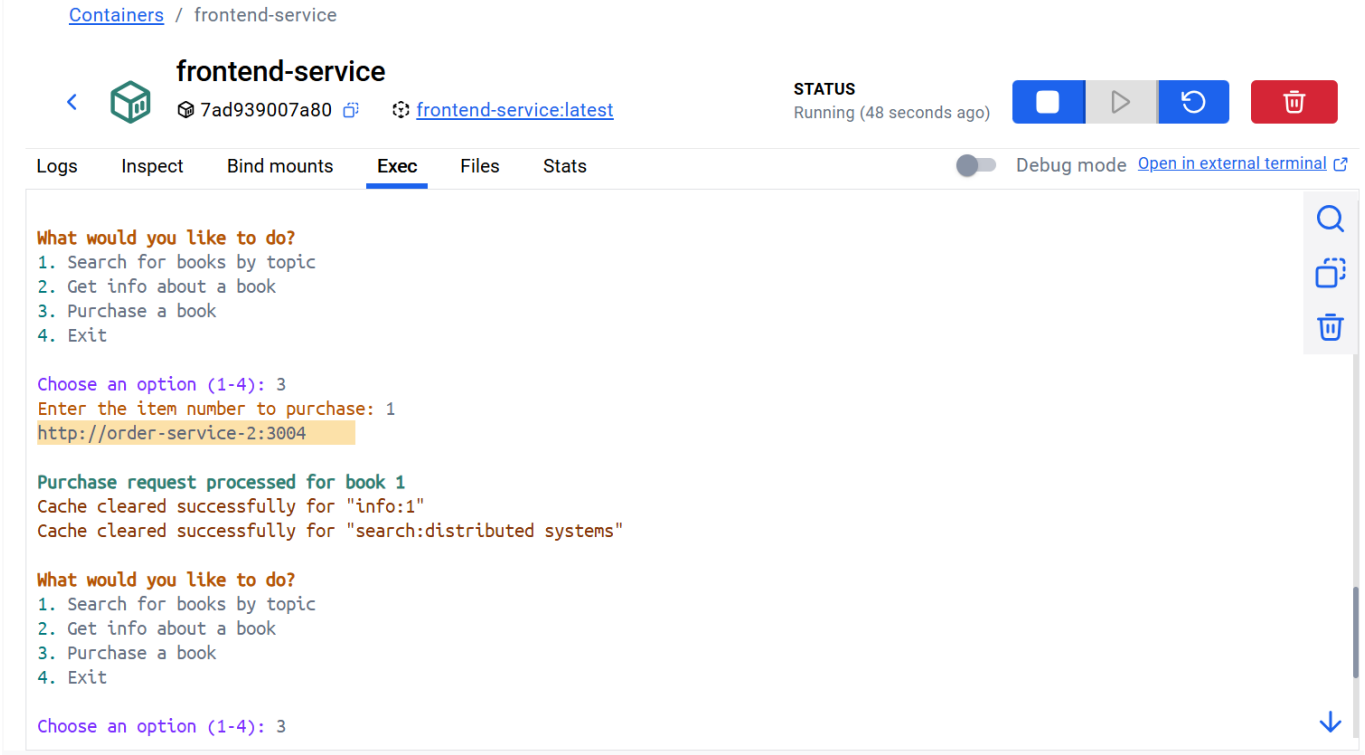
✓ Caching Evaluation

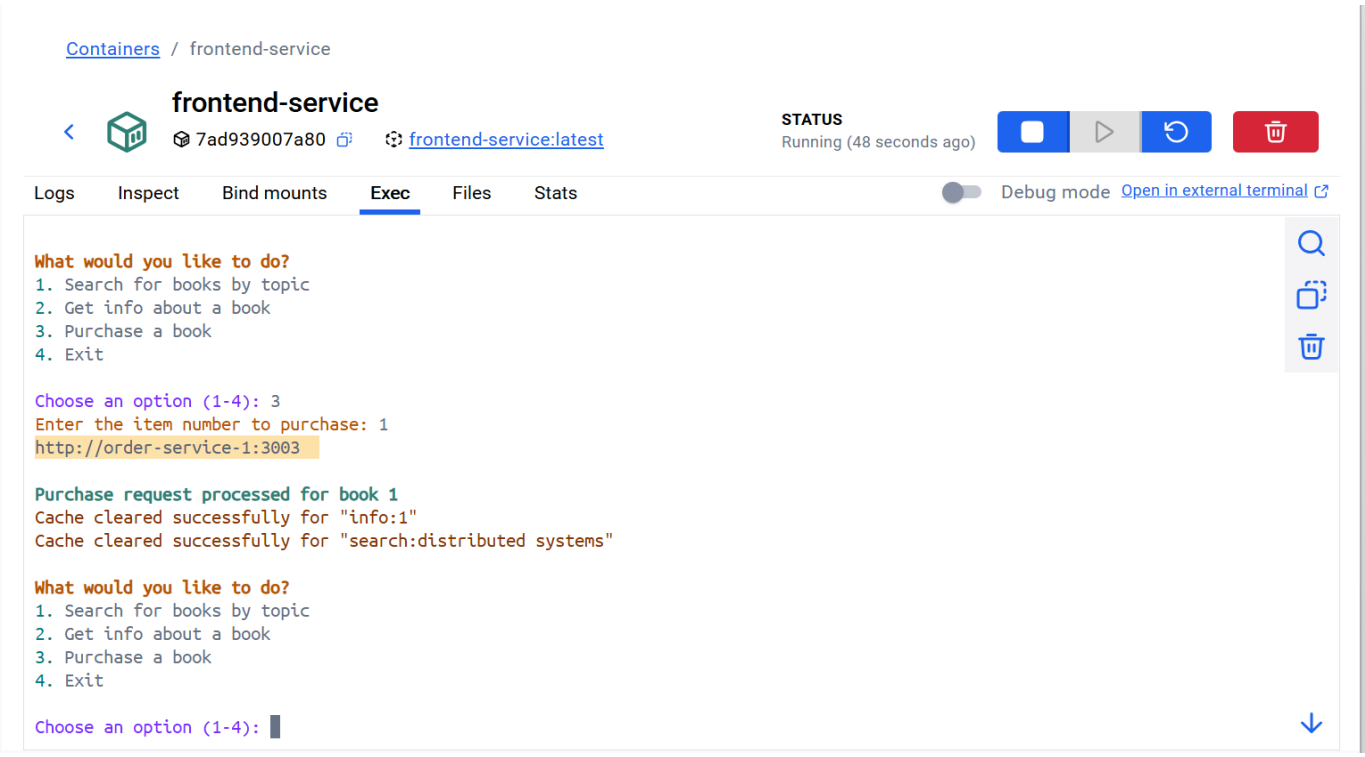
The caching mechanism was evaluated by performing repeated search requests for the same topic. During the first request, the data was retrieved from the catalog service and stored in the cache. When the same request was issued again, the results were served directly from the cache, as indicated by the "Books found (from cache)" message. This confirms that caching is functioning correctly and contributes to reduced latency and lower load on the catalog service.

- When requesting detailed information for a specific book by its item number, the system correctly retrieved the data from the catalog service.

The system correctly handles item-based queries and routes them to the catalog service.

- After purchasing a book, the system automatically cleared the relevant cache entries to maintain data consistency.





System Behavior Evaluation

The system was tested for caching, cache invalidation, load balancing, and consistency. Search requests were cached after the first access and served from the cache on subsequent requests. New topics correctly resulted in cache misses. After purchasing a book, the system invalidated related cache entries to maintain consistency. Purchase requests were successfully distributed across different order service replicas, confirming proper load balancing.

Thank you