

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>



Introduction

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.

```
10
11  const cache = {};
12
13  const catalogReplicas = [
14    "http://catalog-service-1:3001",
15    "http://catalog-service-2:3002",
16  ];
17  const orderReplicas = [
18    "http://order-service-1:3003",
19    "http://order-service-2:3004",
20  ];
21
22  let catalogIndex = 0;
23  let orderIndex = 0;
24
25  function getNextCatalogReplica() {
26    catalogIndex = (catalogIndex + 1) % catalogReplicas.length;
27    console.log(catalogReplicas[catalogIndex]);
28    return catalogReplicas[catalogIndex];
29  }
30
31  function getNextOrderReplica() {
32    orderIndex = (orderIndex + 1) % orderReplicas.length;
33    console.log(orderReplicas[orderIndex]);
34    return orderReplicas[orderIndex];
35  }
36
```

φ 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:

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments', 'History', 'Flows', and 'Files (BETA)'. The main area has tabs for 'Overview', 'POST http://localhost...', 'POST http://localhost...', 'GET http://192.168...', and 'GET http://localhost...'. A new tab is being created. Below these tabs, there's a search bar for 'Search collections' and a button to 'New Collection'. The central part of the screen shows an 'HTTP' request to 'http://localhost:3001/search/undergraduate%20school'. The method is 'GET', and the URL is 'http://localhost:3001/search/undergraduate%20school'. There are buttons for 'Save', 'Share', and 'Send'. Below the URL, there are tabs for 'Docs', 'Params' (selected), 'Authorization', 'Headers (6)', 'Body', 'Scripts', 'Tests', and 'Settings'. Under 'Params', there's a table with columns 'Key', 'Value', 'Description', and 'Bulk Edit'. The 'Body' tab is selected, showing a JSON response with two items:

```
1 [  
2 {  
3   "item_number": "3",  
4   "title": "Xen and the Art of Surviving Undergraduate School",  
5   "quantity": "18",  
6   "price": "150",  
7   "topic": "undergraduate school"  
8 },  
9 {  
10   "item_number": "4",  
11   "title": "Cooking for the Impatient Undergrad",  
12   "quantity": "20",  
13   "price": "20",  
14   "topic": "undergraduate school"  
15 }
```

The response status is '200 OK' with a duration of '39 ms' and a size of '505 B'. There are also icons for copy, share, and refresh.

ریم عبد الرحیم شاهر حسن's Workspace

New Import

Overview POST http://localhost:3001/update-quantity/3 POST http://localhost:3001/info/3 GET http://192.168.1.10:3001/info/3 POST http://localhost:3001/info/3 + No environment

Collections Environments History Flows

http://localhost:3001/update-quantity/3

POST http://localhost:3001/update-quantity/3

Docs Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
{}	JSON	Preview Visualize	200 OK 15 ms 321 B
1 {			
2 "message": "Successfully purchased Xen and the Art of Surviving Undergraduate School"			
3 }			

Body Cookies Headers (7) Test Results

Cloud View Find and replace Console Terminal Runner Start Proxy Cookies Vault Trash

ریم عبد الرحیم شاهر حسن's Workspace

New Import

Overview POST http://localhost:3001/update-quantity/3 POST http://localhost:3001/info/3 GET http://192.168.1.10:3001/info/3 GET http://localhost:3001/info/3 + No environment

Collections Environments History Flows

http://localhost:3001/info/3

GET http://localhost:3001/info/3

Docs Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
{}	JSON	Preview Visualize	200 OK 10 ms 376 B
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 }			

Body Cookies Headers (7) Test Results



With using cache

Yousef Abd Alrahim Shaheen's Workspace

Overview POST http://localhost:3001/search/undergraduate%20school GET http://192.168.1.10:3001/search/undergraduate%20school + No environment

HTTP http://localhost:3001/search/undergraduate%20school

GET http://localhost:3001/search/undergraduate%20school Send

Docs Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Body Cookies Headers (7) Test Results

200 OK 13 ms 505 B

```
[{"item_number": "3", "title": "Xen and the Art of Surviving Undergraduate School", "quantity": "17", "price": "150", "topic": "undergraduate school"}, {"item_number": "4", "title": "Cooking for the Impatient Undergrad", "quantity": "20", "price": "20", "topic": "undergraduate school"}]
```

Yousef Abd Alrahim Shaheen's Workspace

Overview POST http://localhost:3001/update-quantity/3 GET http://192.168.1.10:3001/update-quantity/3 + No environment

HTTP http://localhost:3001/update-quantity/3

POST http://localhost:3001/update-quantity/3 Send

Docs Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Body Cookies Headers (7) Test Results

200 OK 9 ms 321 B

```
{"message": "Successfully purchased Xen and the Art of Surviving Undergraduate School"}
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments', 'History', and 'Flows'. The main area has tabs for 'Overview', 'POST http://localhost:3001/info/3' (status 200 OK), 'GET http://localhost:3001/info/3' (status 200 OK), and 'GET http://192.168.1.10:3001/info/3' (status 200 OK). A search bar at the top says 'Search collections' and a button says 'New Collection'. Below the tabs, there's a 'Send' button and a 'Params' tab under 'Headers'. The 'Body' tab is selected, showing a JSON response:

```
1 {  
2   "item_number": "3",  
3   "title": "Xen and the Art of Surviving Undergraduate School",  
4   "quantity": "12",  
5   "price": "150",  
6   "topic": "undergraduate school"  
7 }
```

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

The screenshot shows a Docker container interface for a service named 'frontend-service'. The container ID is 7ad939007a80, and the image is 'frontend-service:latest'. The status is 'Running (13 minutes ago)'. The 'Exec' tab is selected, showing a terminal session.

```
Choose an option (1-4): 2
Enter the item number of the book: 1

Book info (from cache):
(index) item_number title quantity price topic
0 '1' 'How to get a good grade in DOS in 40 minutes a day' '16' '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
```

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 x + v - □ ×
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.



```
JS client.js ↓M, M   catalog.csv M   docker-compose.yml M X

docke docker-compose.yml > { } services > { } frontend-service > [ ] depends_on
      docker-compose.yml - The Compose specification establishes a standard for the definitio
1  version: "3"
  ▷Run All Services
2  services:
  ▷Run Service
3    catalog-service-1:
4      image: catalog-service
5      container_name: catalog-service-1
6      build:
7        context: ./catalog-service
8      ports:
9        - "3001:3001"
10     environment:
11       - PORT=3001
12     volumes:
13       - ./catalog-service/catalog.csv:/app/catalog.csv
14     networks:
15       - network1
16
17
18    catalog-service-2:
19      image: catalog-service
20      container_name: catalog-service-2
21      ports:
22        - "3002:3001"
23      environment:
24        - PORT=3002
25      volumes:
26        - ./catalog-service/catalog.csv:/app/catalog.csv

0          ⚡ ReemHassan31 (1 month ago)  Lr
```

JS client.js !M, M catalog.csv M docker-compose.yml M

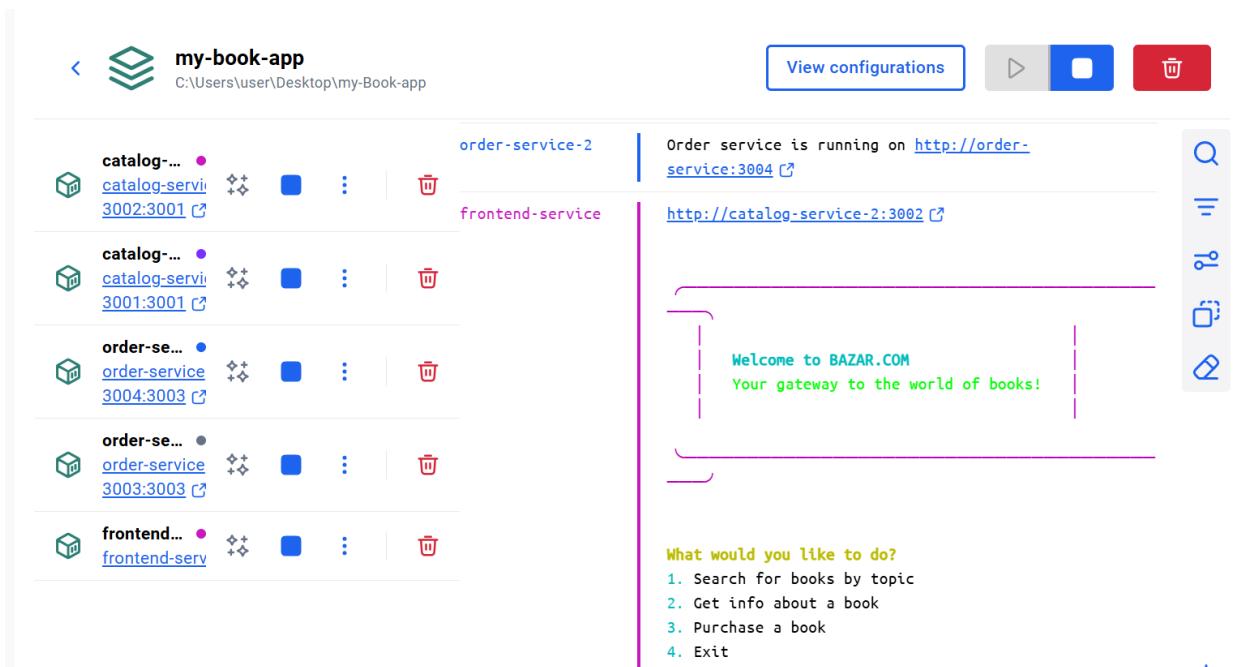
docker-compose.yml > {} services > {} order-service-2 > [] depends_on > abc 1

```
2   services:
30
31     ▷Run Service
32     order-service-1:
33       image: order-service
34       container_name: order-service-1
35       build:
36         context: ./order-service
37       ports:
38         - "3003:3003"
39       environment:
40         - PORT=3003
41       networks:
42         - network1
43       depends_on:
44         - catalog-service-1
45         - catalog-service-2
46
47     ▷Run Service
48     order-service-2:
49       image: order-service
50       container_name: order-service-2
51       ports:
52         - "3004:3003"
53       environment:
54         - PORT=3004
55       networks:
56         - network1
57       depends_on:
58         - catalog-service-1
59         - catalog-service-2
```

```
55
56
57     ▷Run Service
58     frontend-service:
59       image: frontend-service
60       container_name: frontend-service
61       build:
62         context: ./Frontend
63       networks:
64         - network1
65       depends_on:
66         - order-service-1
67         - order-service-2
68         - catalog-service-1
69         - catalog-service-2
70
71       stdin_open: true
72       tty: true
73
74       networks:
75         network1:
76           driver: bridge
77
```

Result

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



Logs Inspect Bind mounts **Exec** Files Stats Debug mode [Open in external terminal ↗](#)

2. Get info about a book
3. Purchase a book
4. Exit

Choose an option (1-4): 1
Enter the topic: distributed systems

Books found 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'
1	'2'	'RPCs for Noobs'	'20'	'50'	'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): 1
Enter the topic: distributed systems

Books found (from cache):

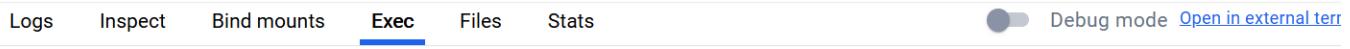
(index)	item_number	title	quantity	price	topic
0	'1'	'How to get a good grade in DOS in 40 minutes a day'	'13'	'100'	'distributed systems'
1	'2'	'RPCs for Noobs'	'20'	'50'	'distributed systems'

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

✓ 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.



Logs Inspect Bind mounts **Exec** Files Stats Debug mode [Open in external term](#)

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): 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?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit

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.

[Containers](#) / frontend-service

frontend-service

7ad939007a80 [frontend-service:latest](#)

STATUS
Running (48 seconds ago)

Logs Inspect Bind mounts **Exec** Files Stats [Open in external terminal](#)

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"

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

[Containers](#) / frontend-service

frontend-service

7ad939007a80 [frontend-service:latest](#)

STATUS
Running (48 seconds ago)

Logs Inspect Bind mounts **Exec** Files Stats [Open in external terminal](#)

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-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):

* 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