BAZAR.COM

PART2

Mohammed Jaddou - 12112568

Ahmad Sultan - 12112656

INTRODUCTION

The goal of Part 2 of this project is to enhance the Bazar.com system developed in Part 1 by improving performance, scalability, and consistency. With the increased number of users and requests, the original system experienced higher response times. To address this issue, replication, caching, and consistency mechanisms were introduced.

This part focuses on redesigning the system architecture to reduce latency, distribute workload across multiple servers, and maintain data correctness. The system was also optionally containerized using Docker to simplify deployment and management.

SYSTEM ARCHITECTURE OVERVIEW

The improved Bazar.com system consists of the following components:

- **Front-End Server (single instance)**
- **Catalog Server (replicated)**
- **Order Server (replicated)**
- **In-Memory Cache**
- **CSV files for persistent storage**
- **Docker containers**

The front-end server receives all client requests and is responsible for load balancing, caching, and request routing.

REPLICATION

Replication is used to improve system performance and availability by running multiple copies of the catalog and order servers. The front-end server receives all client requests and distributes them among the replicas using a round-robin load balancing algorithm. Each replica runs on a different port and contains the same data and logic. Write operations are propagated between replicas to keep them synchronized and consistent.

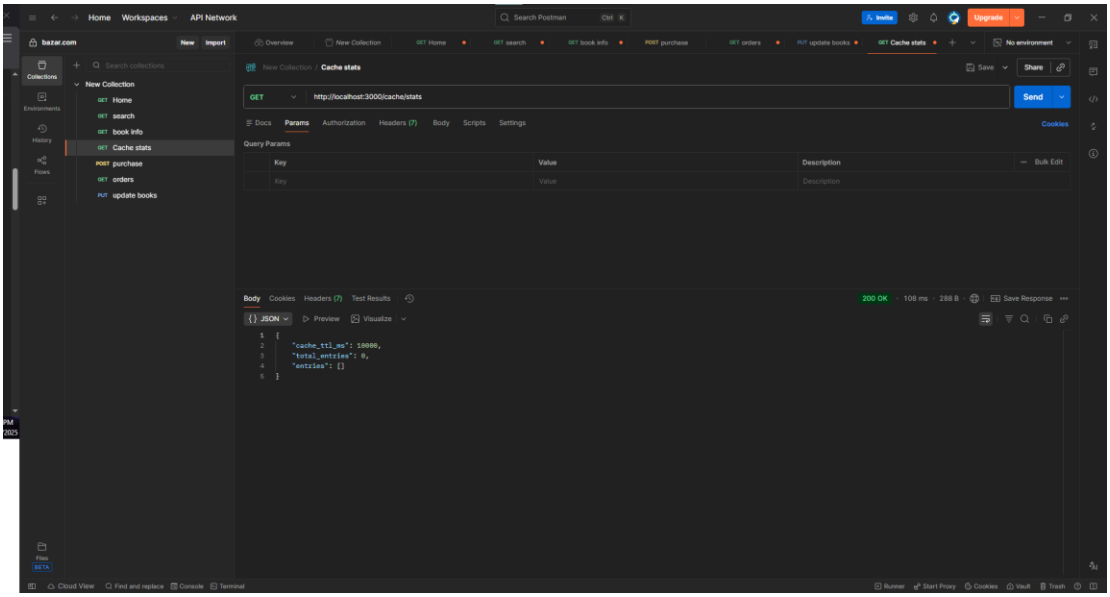
```
[FRONTEND] Frontend service running on port 3000
[FRONTEND] Catalog servers: http://catalog-server:3001, http://catalog-server-replica:3001
[FRONTEND] Order servers: http://order-server:3002, http://order-server-replica:3002
[FRONTEND] Cache TTL: 10000ms
[FRONTEND] Info request for item: 2
[FRONTEND] Using catalog server: http://catalog-server:3001
[FRONTEND] Cached info for item: 2
[FRONTEND] Info request for item: 2
[FRONTEND] Using catalog server: http://catalog-server-replica:3001
[FRONTEND] Cached info for item: 2
[FRONTEND] Info request for item: 2
[FRONTEND] Cache hit for item: 2
```

CACHING

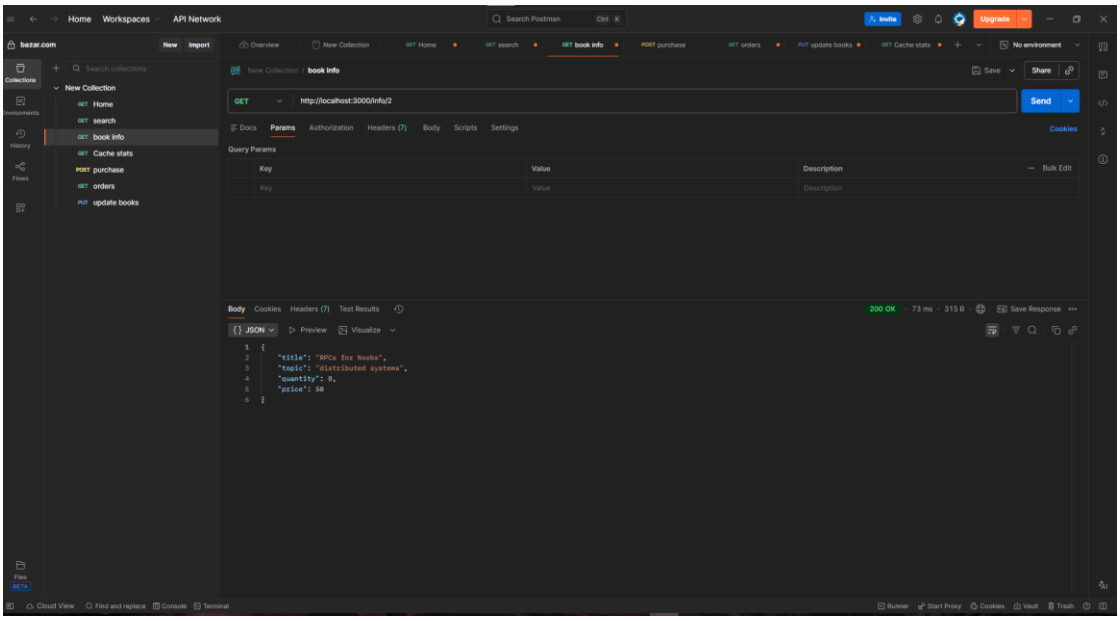
Caching is used to reduce response time for frequently accessed read requests. An in-memory cache is implemented at the front-end server to store recent book information. For each query, the front-end first checks the cache before forwarding the request to a catalog replica. Write operations bypass the cache, and cache entries are invalidated before updates to ensure consistency.

CACHE PERFORMANCE

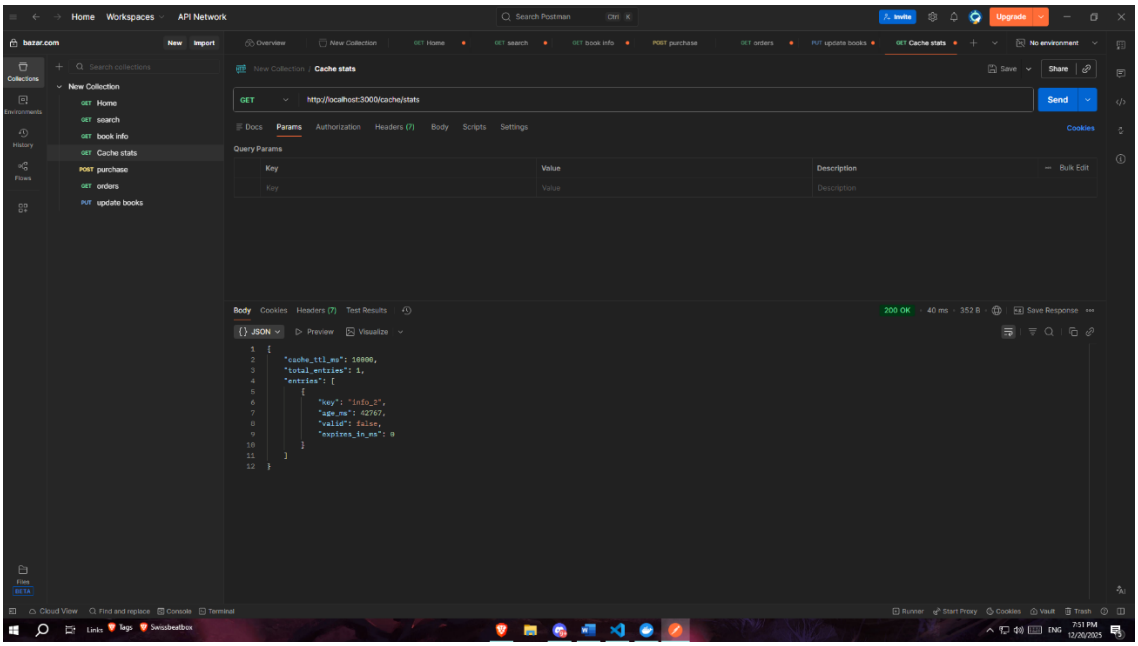
At the start of the experiment, the cache is empty, meaning no book information is stored in memory. When the client sends the first request, the front-end server does not find the requested data in the cache and forwards the request to one of the catalog server replicas using round-robin load balancing. The response is then returned to the client and stored in the cache.



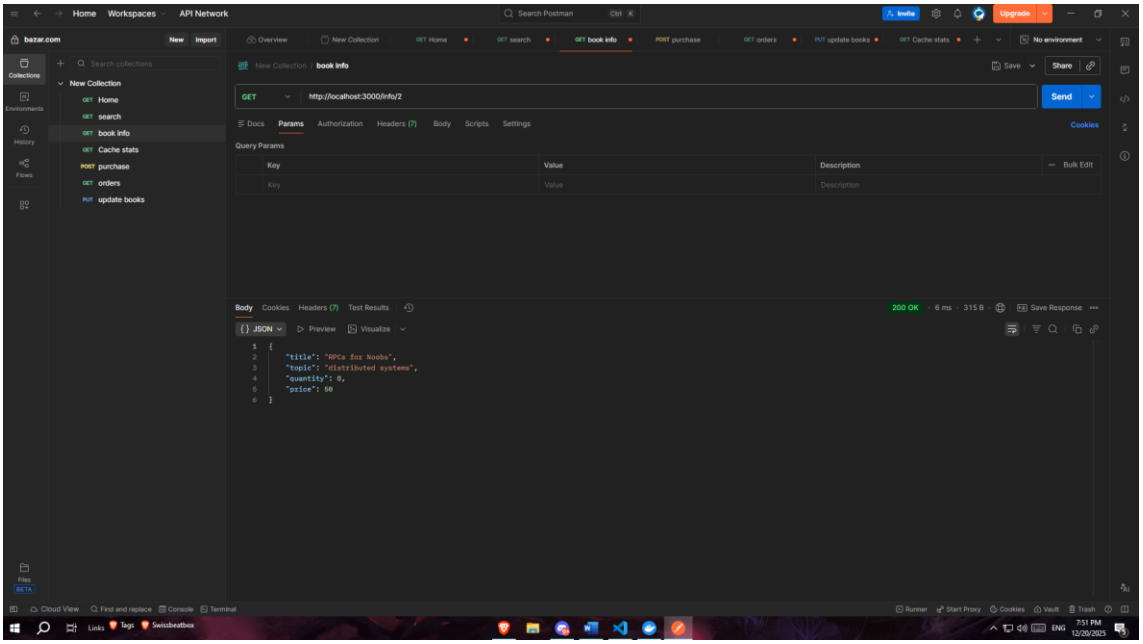
The response time for this initial request is higher because it involves communication with the backend server. In our experiment, the first request took approximately 108 ms to complete.



After this request, the cache becomes populated with the requested book data. When the same or similar request is sent again, the front-end server retrieves the data directly from the cache without contacting the catalog server.



As a result, the response time is significantly reduced. Subsequent requests are served much faster, with an average response time of approximately 6 ms, demonstrating the effectiveness of caching in reducing system latency.



CONCLUSION

In this part, Bazar.com was successfully transformed into a more scalable and efficient system. Replication and caching significantly improved response times, while consistency mechanisms ensured data correctness. Docker further simplified deployment and system management.