An-Najah National University

Faculty of Engineering & Information Technology

Dos-Project - Part 2

Abdulqader mohammad – 12112218

Mohammad raddad – 12027855

1. Introduction

In this lab, we extend the Bazar.com online bookstore developed in Lab 1 to handle higher workloads and improve performance. The main objective is to apply key distributed systems concepts including replication, caching, and consistency using a microservices-based architecture and RESTful APIs.

2. System Architecture:

The system consists of the following components:

- **Frontend Server**
  - Receives all client requests
  - Implements load balancing using Round Robin
  - Maintains an in-memory cache with LRU eviction
- **Catalog Service**
  - Stores book information (title, price, quantity)
  - Implemented with SQLite
  - Replicated into two instances running on different ports
- **Order Service**
  - Handles purchase requests
  - Replicated into two instances
  - Updates all catalog replicas to maintain consistency

3. Replication

Replication is implemented for both the Catalog Service and the Order Service.

The frontend server distributes incoming requests among replicas using a Round Robin load-balancing strategy.

For write operations (purchases), the Order Service propagates updates to all catalog replicas, ensuring that all copies remain synchronized.

4. Caching

An in-memory cache is implemented inside the frontend server to store responses for read-only requests (/info/:id).

Cache characteristics:

- **Used only for read requests**
- **Limited cache size**
- **Uses Least Recently Used (LRU) eviction policy**
- **Significantly reduces response time for repeated requests**

5. Cache Consistency

To maintain strong consistency, a server-push invalidation mechanism is used.

When a write operation occurs:

- **The Order Service updates the catalog replicas**
- **The frontend cache is explicitly invalidated for the affected item**
- **The next read request results in a cache miss and fetches updated data**

This mechanism prevents stale data from being served to clients.

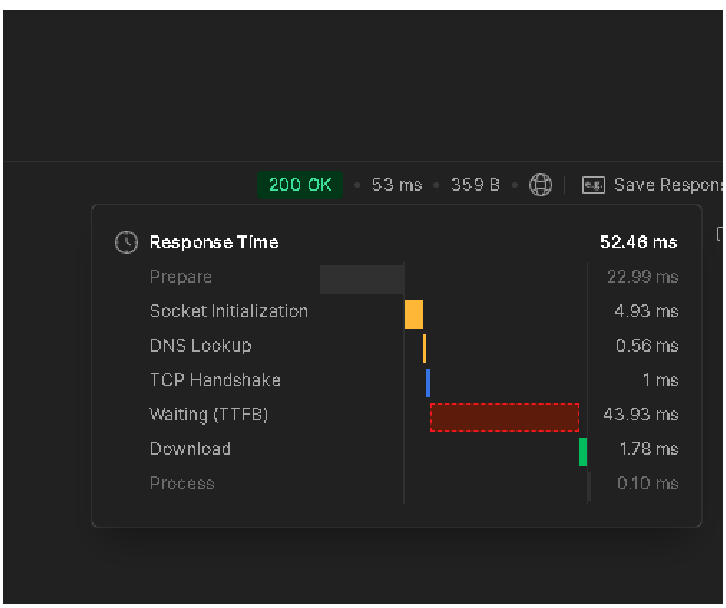
6. Experimental Evaluation

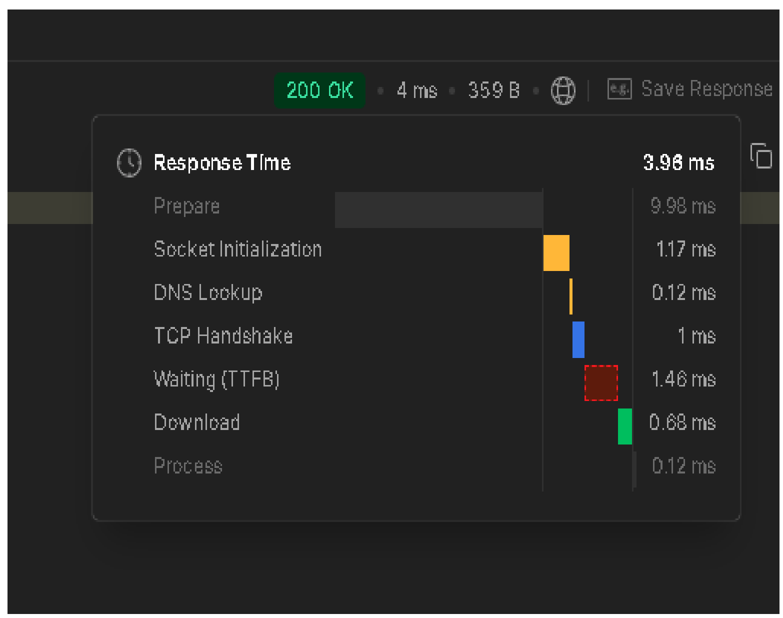
6.1 Response Time Measurement

We measured the average response time for /info/:id requests in two scenarios:

We measured the average response time by issuing 50 consecutive /info/:id requests using a Node.js script. The experiment was performed with caching disabled and enabled.

Scenario	Average Response Time
Without Cache	52.46 ms
With Cache	4 ms





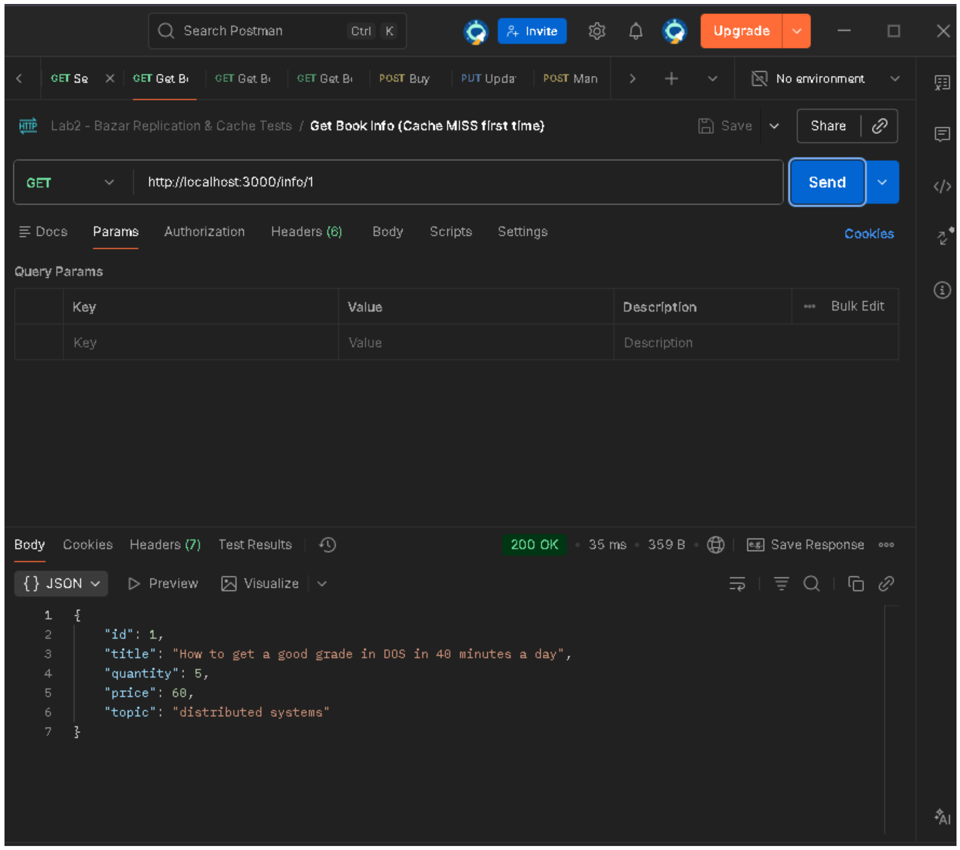
6.2 Cache Invalidation Experiment

The following experiment was conducted:

1. **Request /info/:id → Cache HIT**
2. **Execute /purchase/:id**
3. **Cache invalidation occurs**
4. **Request /info/:id again → Cache MISS**

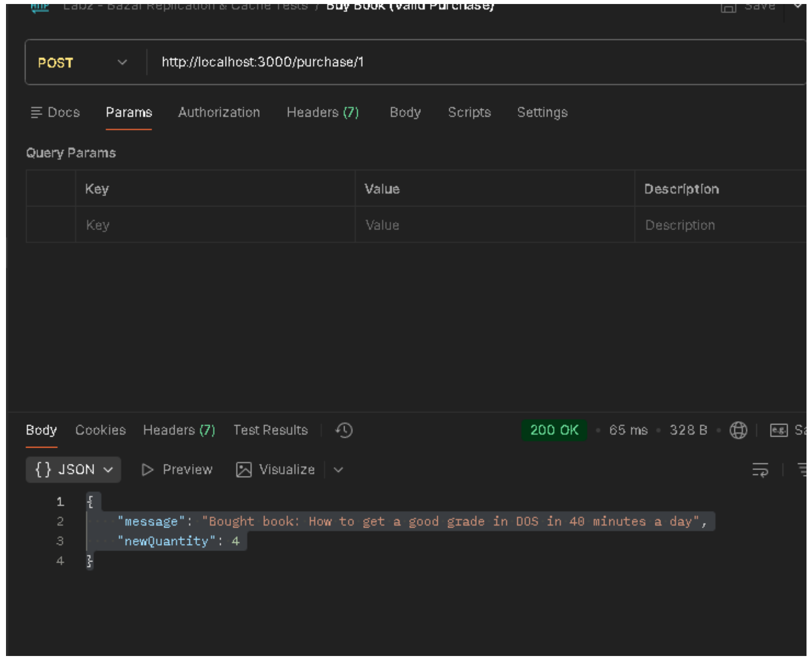
Operation	Time (ms)
Cache Invalidation	80
Request after invalidation (MISS)	7

- **first request**

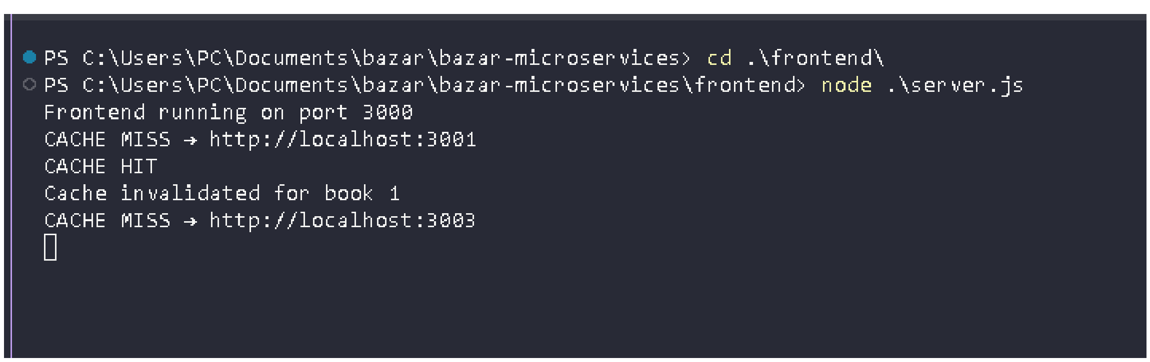


- **second request** **GET http://localhost:3000/info/1**

- **third request**



- <u>**Fourth request**</u> <u>**GET http://localhost:3000/info/1**</u>



7. Design Tradeoffs

- **Replication improves availability and scalability**
- **Caching reduces latency but introduces consistency complexity**
- **Strong consistency is achieved at the cost of additional invalidation overhead**
- **SQLite was chosen for simplicity but limits scalability**

8. Possible Improvements

- **Add health checks for replicas**
- **Use adaptive load balancing (e.g., least-loaded)**
- **Dockerize all services**
- **Replace SQLite with a distributed database**

9. How to Run the System

Start Catalog replicas:

- **node server.js**
- **$env:PORT=3003; node server.js**

Start Order replicas:

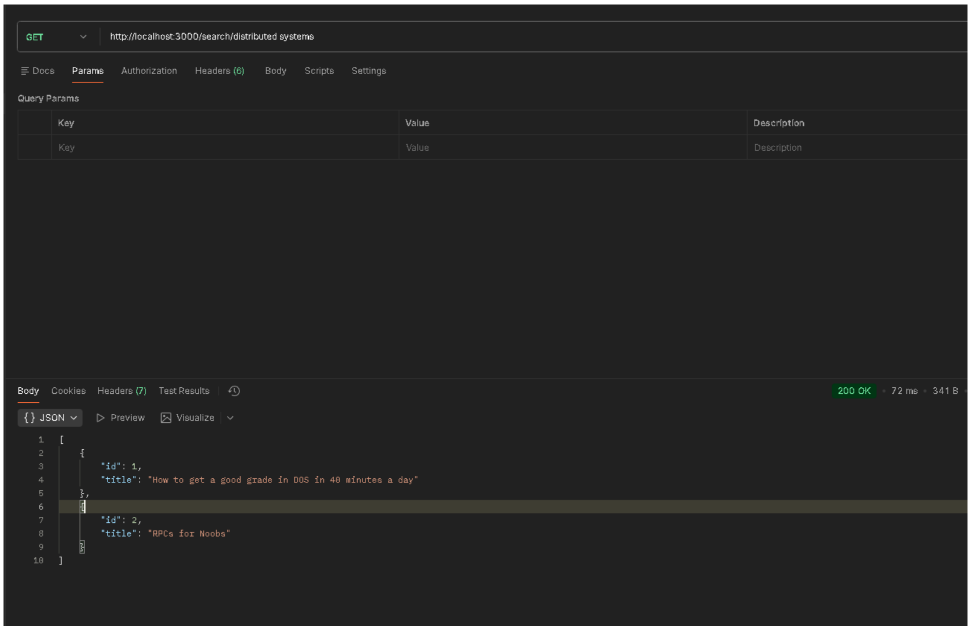
- **node server.js**
- **$env:PORT=3004; node server.js**

Start Frontend server:

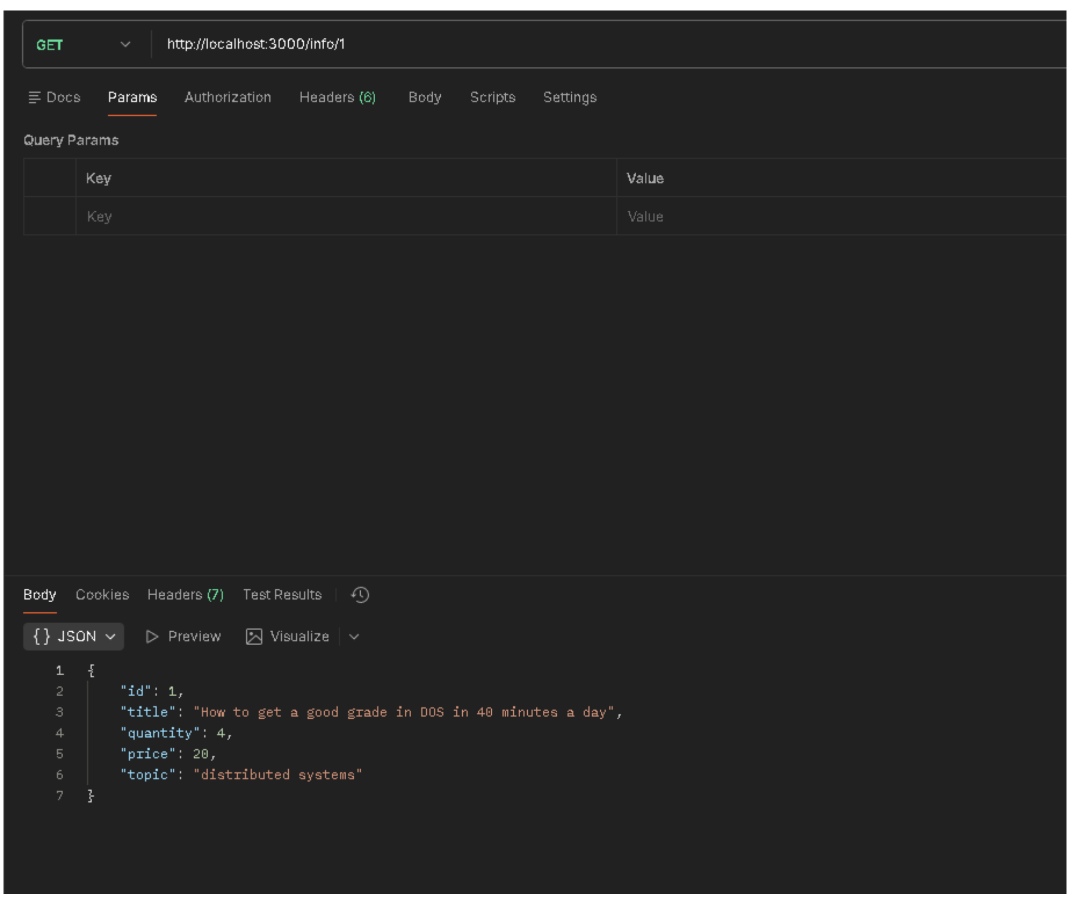
- **node server.js**

10.result :

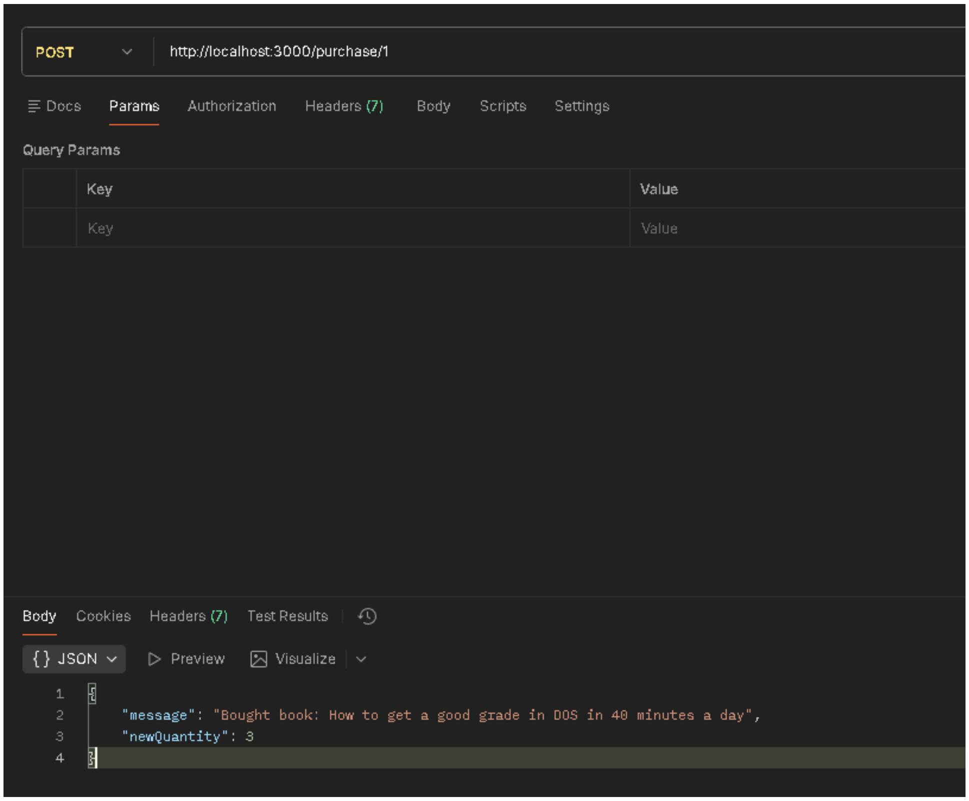
- **http://localhost:3000/search/distributed systems**



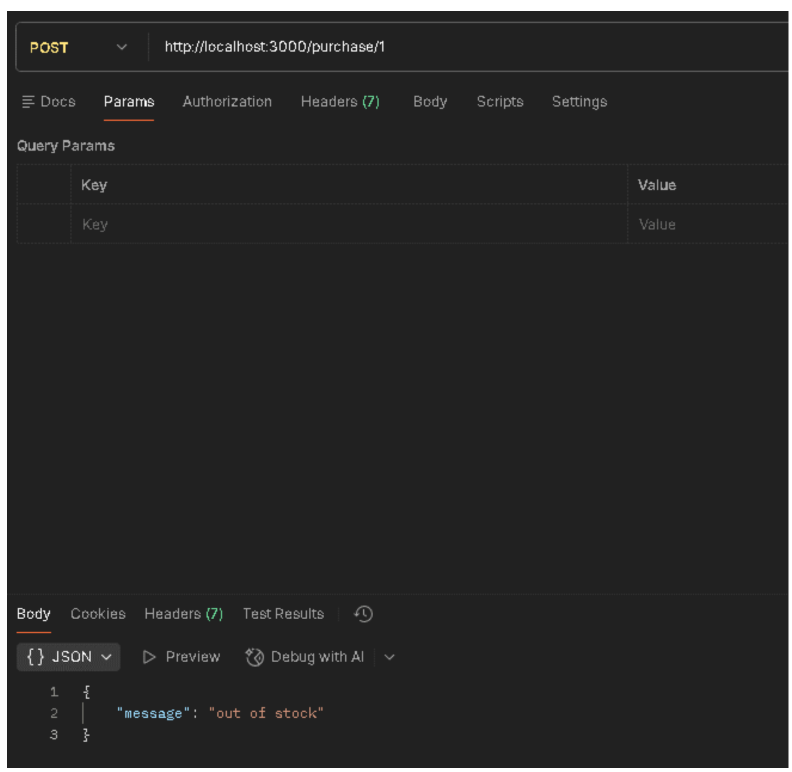
- **http://localhost:3000/info/1**



- **http://localhost:3000/purchase/1**



- **http://localhost:3000/purchase/1 when the out of stock**



11. Conclusion

This lab demonstrates how replication, caching, and consistency mechanisms can be combined to build a scalable and efficient distributed system. The experimental results confirm improved performance while maintaining correct system behavior.