An-Najah National University
Faculty of Engineering and IT

جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا المعلومات

Computer Engineering Department
DOS
Project - Part 2

Instructor Name: Dr.Samer Alarandi

Students Names: Adam Abahri (12113269), Ameed Salameh (12111988)

Academic Year: First 2025-2026

---

1. Abstract

This project is based on extending the distributed online bookstore developed in Project Part 1 to include advanced concepts in distributed systems, which include replication, load balancing, caching, and consistency management, using a microservices-based system designed around a microservices architecture with RESTful APIs using a three-service model, which includes Client, Catalog, and Orders services communicating using RESTful APIs.

In this phase, the Catalog and Order Services are replicated to enhance scalability and availability. For the Client Service, round-robin load balancing and an in-memory cache are added to enable faster handling of requests. To ensure correctness, a server push-based cache invalidation approach is integrated to ensure consistency of the cache even after performing write operations. The final phase involves performance testing of the system both with and without caching.

API Testing and Experimental Validation

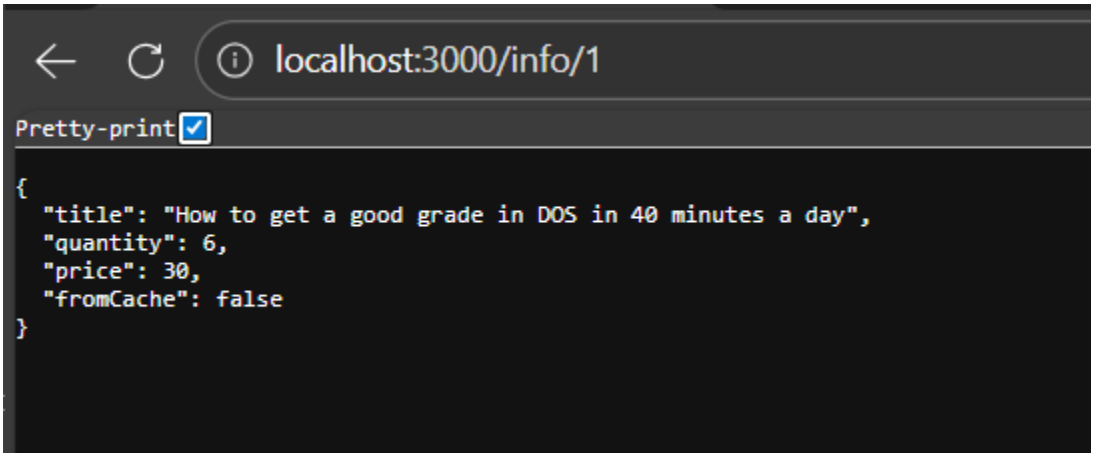
This section documents system validation through **manual API testing** and **service-level execution testing**. All services were executed concurrently and tested using **web browsers**, **Postman**, and **Visual Studio terminal commands**.

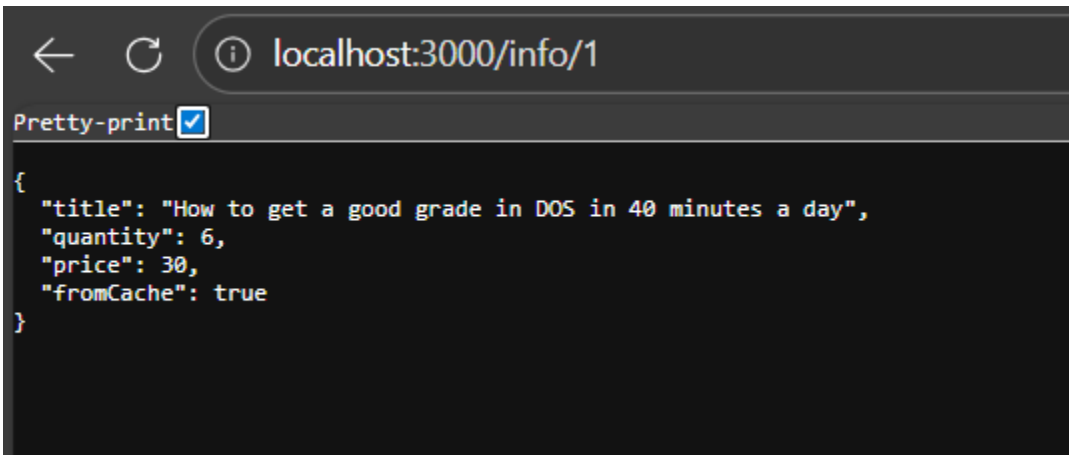
2.1 Client Service API Testing (Port 3000)

A. Book Information API (Cached Read)

Endpoint:

http://localhost:3000/info/1

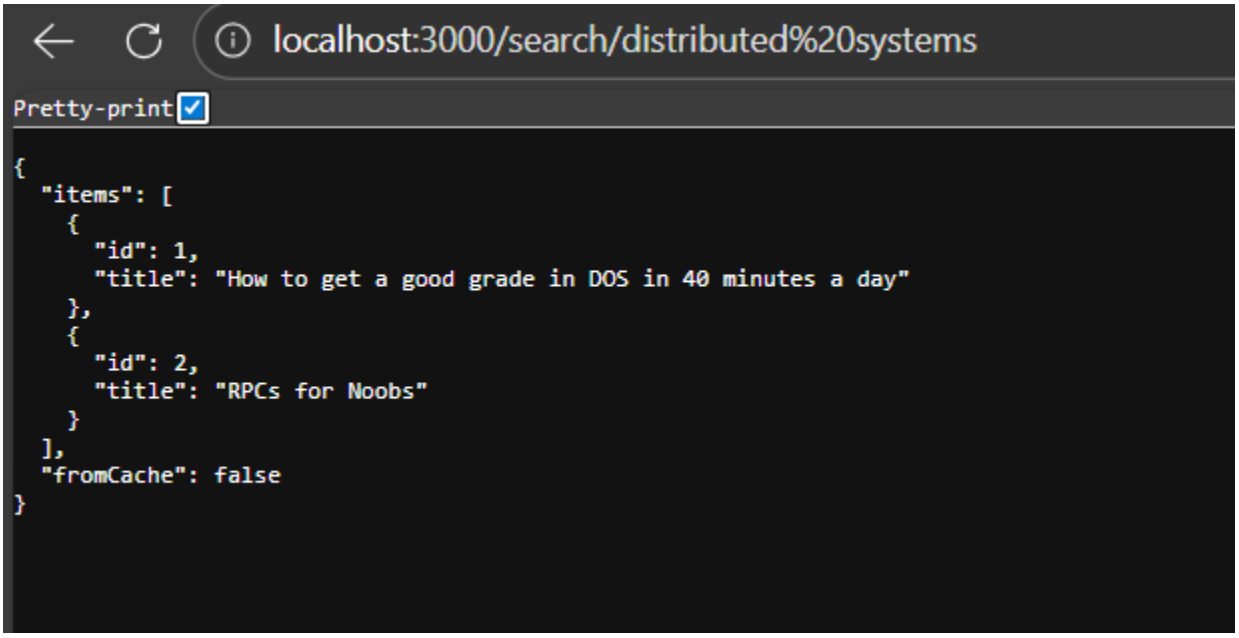


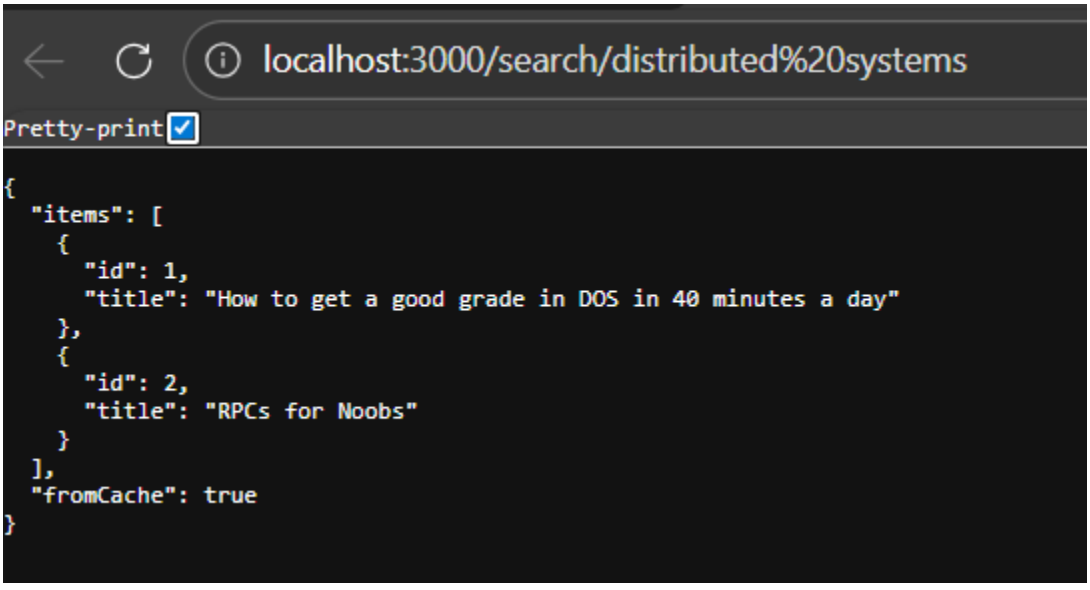


B. Search API (Cached Read)

Endpoint:

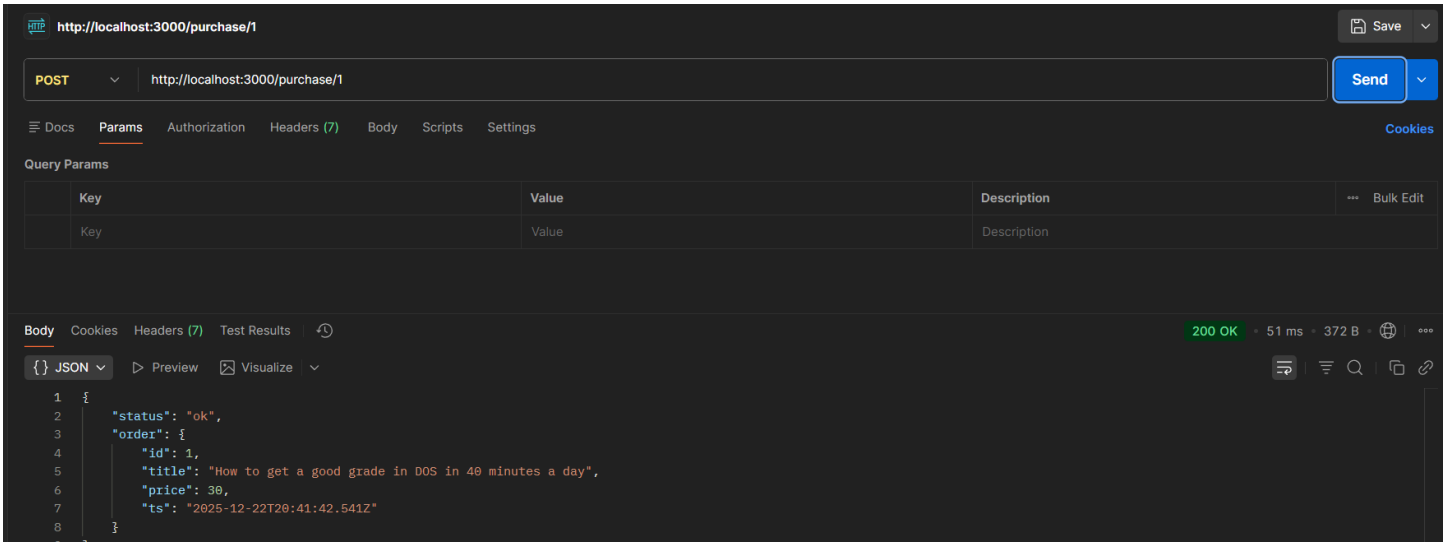
http://localhost:3000/search/distributed%20systems





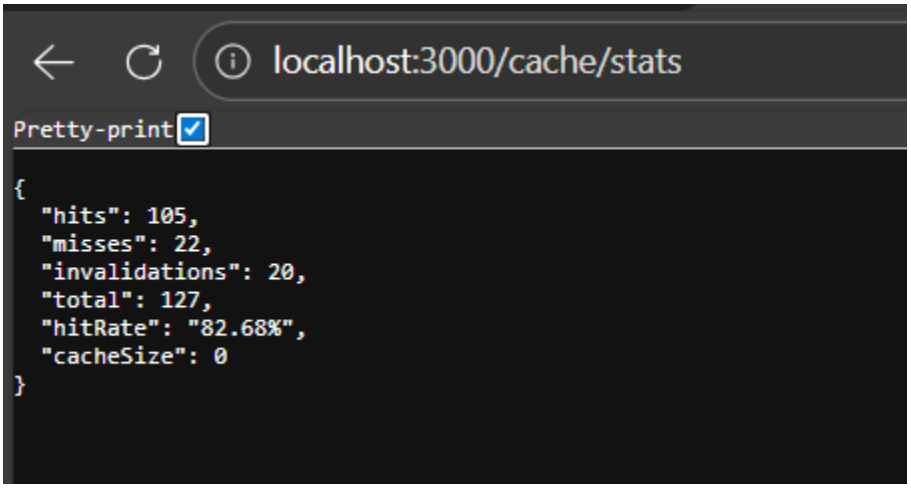
C. Purchase API (Write Operation)

Endpoint:
`POST http://localhost:3000/purchase/1`



D. Cache Statistics API

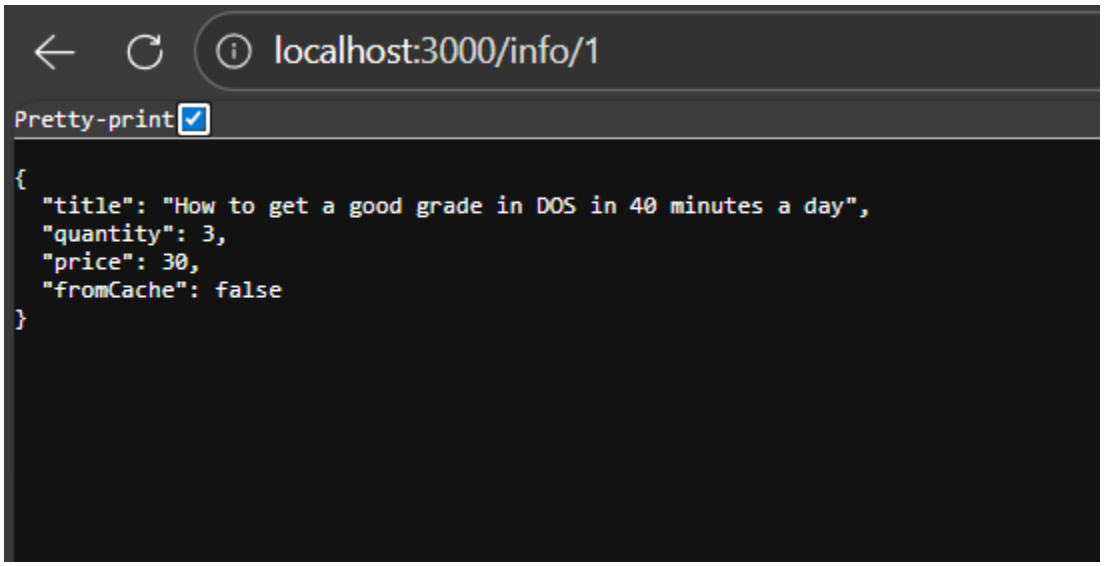
Endpoint:
`GET localhost:3000/cache/stats`



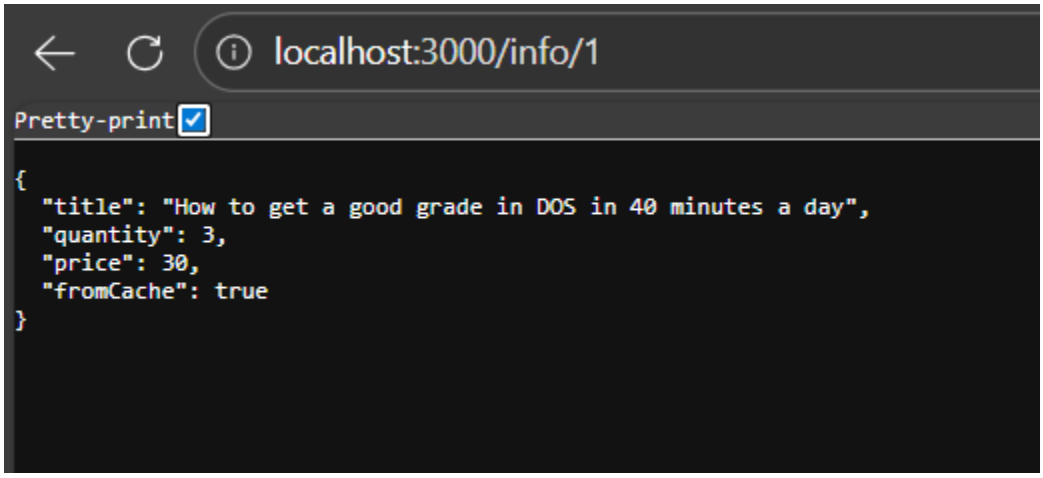
E. Manual Cache Invalidation API

Endpoint:
`POST /invalidate`

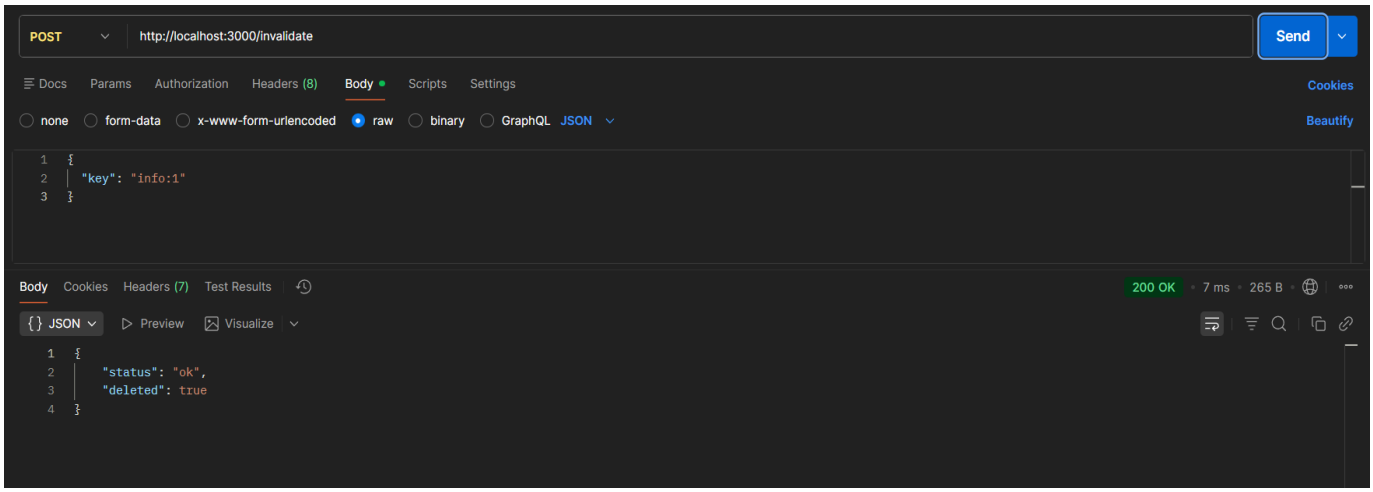
First: we need to search about something that is not in the cache



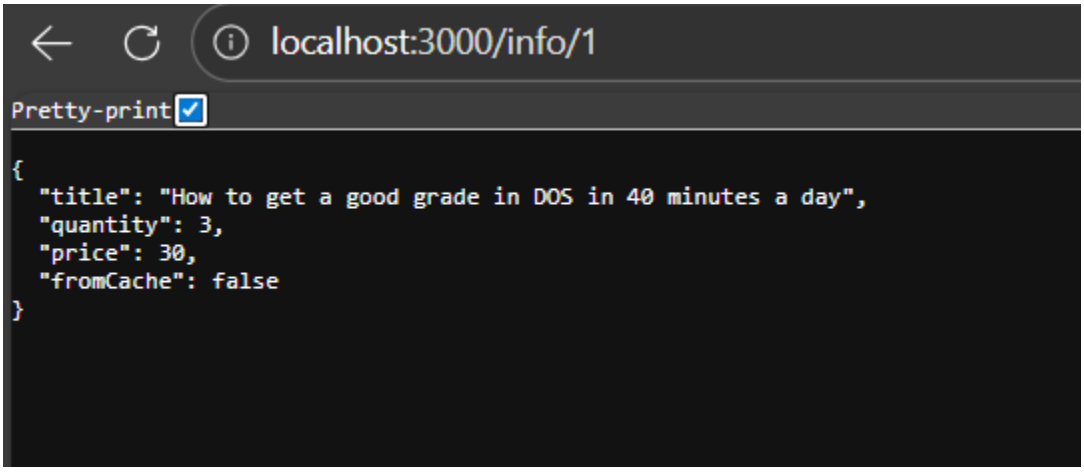
Second: it should be stored in the cache



Third: now we run invalidate API



Finally: we see it has been deleted from the cache



2.2 Catalog Service API Testing (Replicas)

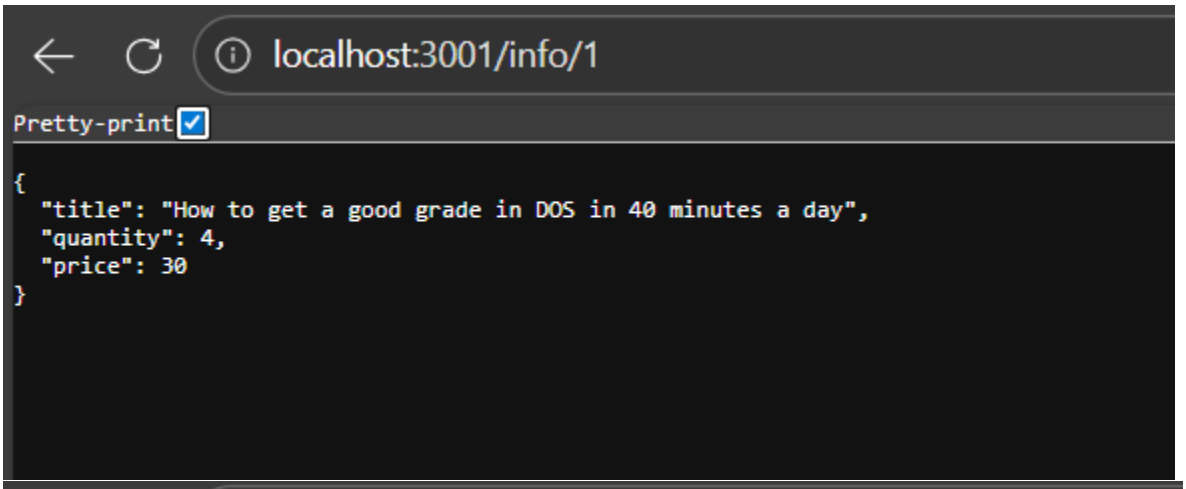
Both Catalog replicas were tested independently.

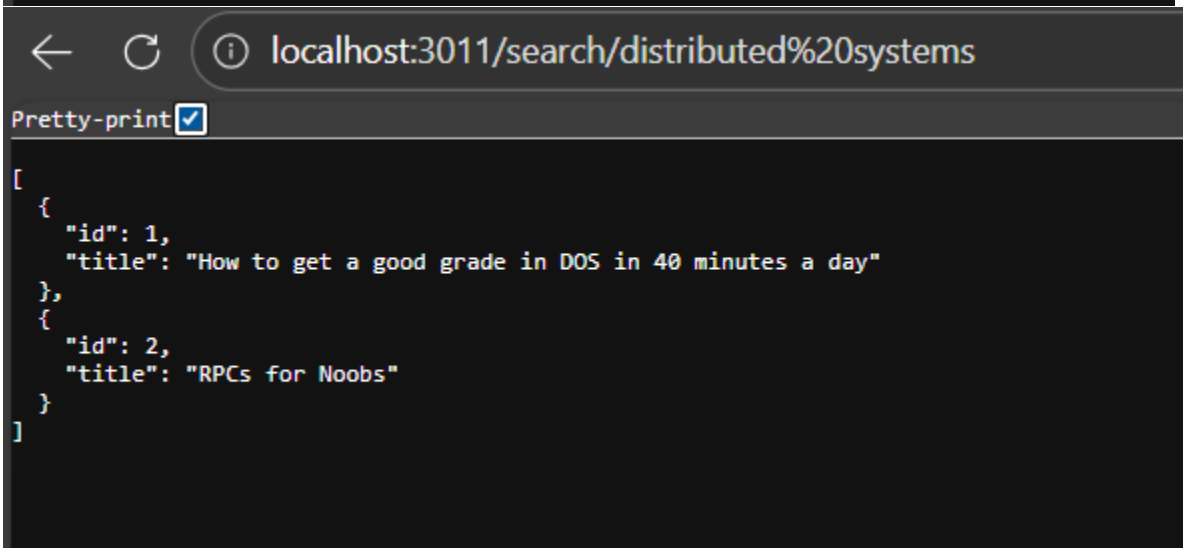
Example:

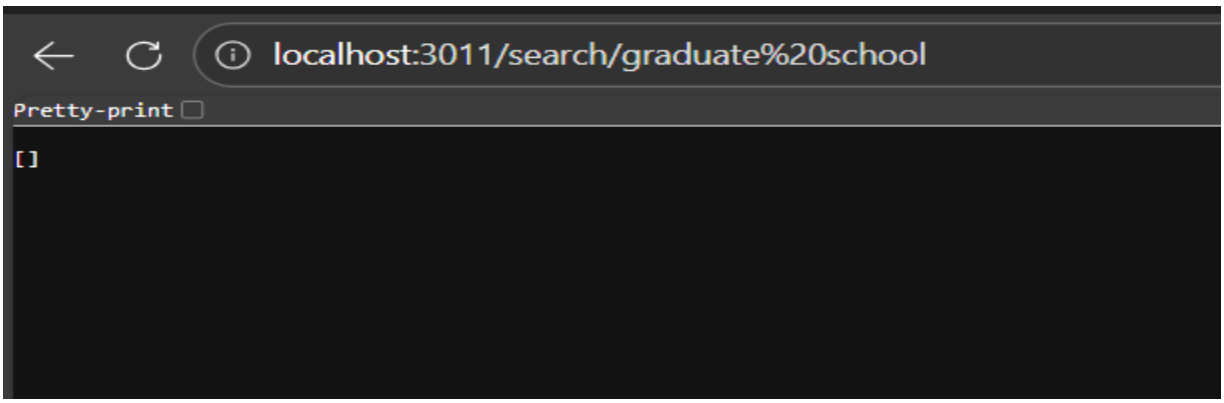
http://localhost:3001/info/1 --> (exists book from replica 1)

http://localhost:3011/search/graduate%20school --> (unexists book from replica 2)

localhost:3011/search/distributed systems --> (exists book from replica 2)





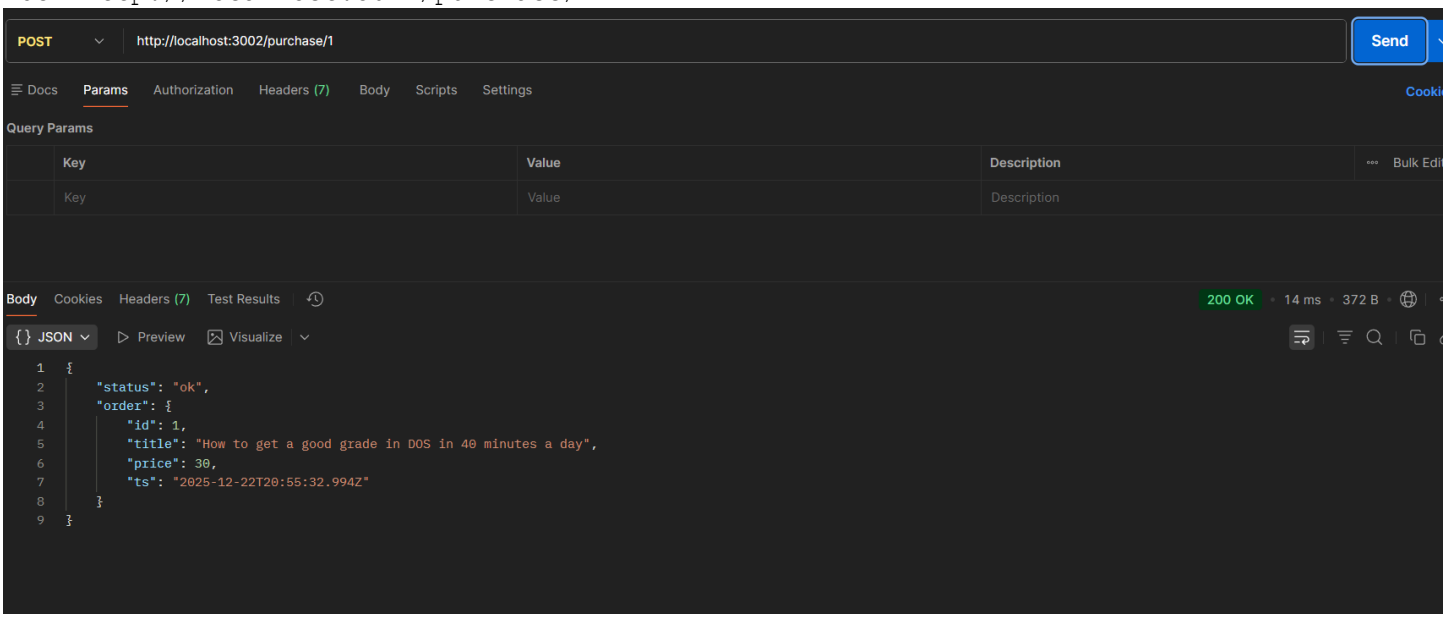


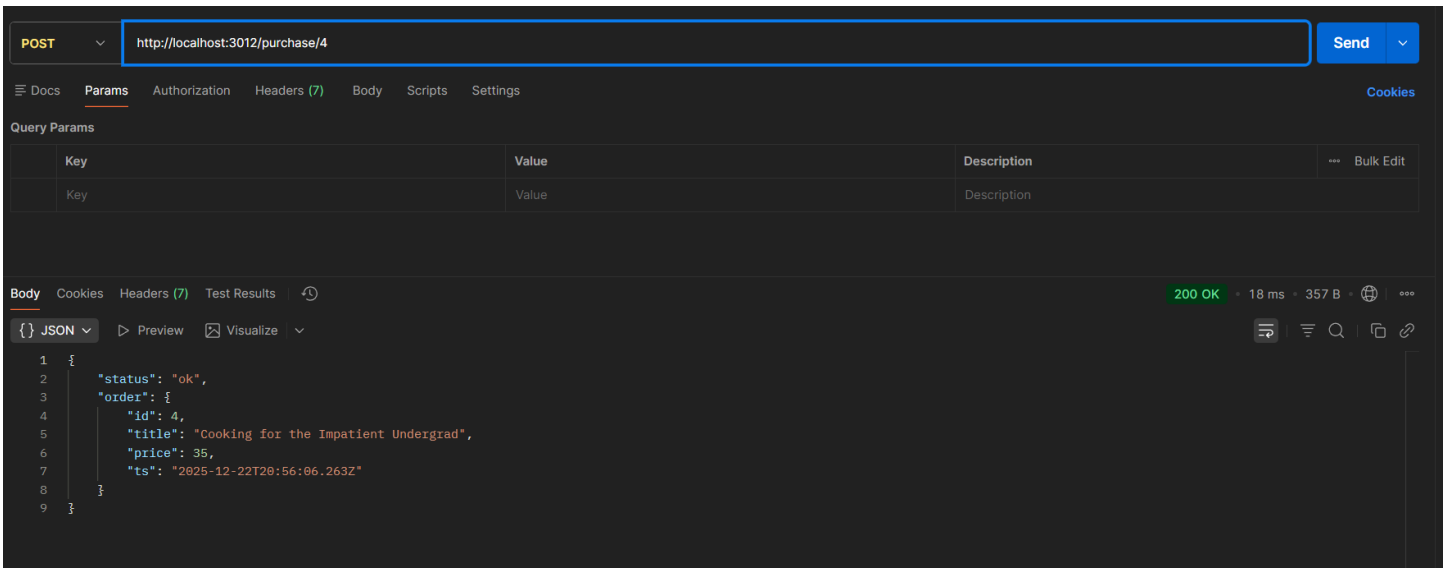
2.3 Order Service API Testing (Replicas)

Order replicas were tested for correct purchase handling.

Example:

```
POST http://localhost:3002/purchase/1
POST http://localhost:3012/purchase/4
```




POST
http://localhost:3012/purchase/4
Send
Docs
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
200 OK
18 ms
357 B
{} JSON
Preview
Visualize
{
"status": "ok",
"order": {
"id": 4,
"title": "Cooking for the Impatient Undergrad",
"price": 35,
"ts": "2025-12-22T20:56:06.263Z"
}
}

2.4 Service Execution at VS Terminal

All services were executed using Visual Studio and terminal scripts.

test_1_cache :

```
============================================================
Test 1: Cache Performance
============================================================

Step 1: Clearing cache...
   Cache cleared

Step 2: First request (Cache Miss - goes to server)
   Response Time: 5ms
   fromCache: false
   Book: How to get a good grade in DOS in 40 minutes a day
   Quantity: 3

Step 3: Second request (Cache Hit - from memory)
   Response Time: 3ms
   fromCache: true
   Book: How to get a good grade in DOS in 40 minutes a day
   Quantity: 3

============================================================
RESULTS:
============================================================
Without Cache: 5ms
With Cache:    3ms
Speedup:       1.67x faster!
Improvement:   40.00%

Test completed!
============================================================
```

test_2_invalidation :

```
PS C:\Users\lenovo\Desktop\project_Distributed-and-Operating-Systems_part1-Adam-P2> node .\test_2_invalidation.js
============================================================
Test 2: Cache Invalidation
============================================================

Step 1: Preparing test (clearing cache)...
   Ready

Step 2: First request - Get book info
   fromCache: false (NOT from cache - first time)
   Book: How to get a good grade in DOS in 40 minutes a day
   Quantity BEFORE purchase: 3

Step 3: Second request - Same book (should be from cache)
   fromCache: true (from cache - fast!)
   Quantity: 3 (same as before)

Step 4: Purchase book (this triggers CACHE INVALIDATION!)
   Sending purchase request...
   Purchase successful!
   Order ID: 1
   Note: Cache was automatically invalidated by the server!

Step 5: Third request - Check if cache was invalidated
   fromCache: false (NOT from cache - was invalidated!)
   Quantity AFTER purchase: 2

============================================================
```

```
============================================================
RESULTS:
============================================================
Request 1: fromCache = false (first time, saves to cache)
Request 2: fromCache = true (from cache, fast!)
Purchase:  Cache INVALIDATED automatically
Request 3: fromCache = false (cache was cleared, fresh data!)

Quantity before: 3
Quantity after:  2
Difference:      1 (should be 1)

============================================================
VERIFICATION:
============================================================
[PASS] Test 1: First request NOT from cache
[PASS] Test 2: Second request FROM cache
[PASS] Test 3: Third request NOT from cache (invalidation worked!)
[PASS] Test 4: Quantity decreased by 1

SUCCESS: ALL TESTS PASSED! Cache Invalidation is working correctly!
============================================================

EXPLANATION:
------------------------------------------------------------
Cache Invalidation = Deleting stale data from cache

Why invalidate?
  - After purchase, quantity changes in database
  - Cache contains old data (old quantity)
  - Need to delete cache so next request gets fresh data

How it works:
  1. Order service processes purchase
  2. Order service updates all Catalog replicas
  3. Catalog service sends invalidation to Front-End
  4. Front-End deletes the cache entry
  5. Next request fetches fresh data from server
```

test_3_loadbalancing :

```
============================================================
Test 3: Load Balancing
============================================================

Note: We will clear cache before each request
      so that requests go to the Catalog servers

Initial state:
  Catalog-1 log lines: 35
  Catalog-2 log lines: 37

Sending 10 requests with Round-Robin...
------------------------------------------------------------
Request 1/10 sent... (Expected: Replica 1)
Request 2/10 sent... (Expected: Replica 2)
Request 3/10 sent... (Expected: Replica 1)
Request 4/10 sent... (Expected: Replica 2)
Request 5/10 sent... (Expected: Replica 1)
Request 6/10 sent... (Expected: Replica 2)
Request 7/10 sent... (Expected: Replica 1)
Request 8/10 sent... (Expected: Replica 2)
Request 9/10 sent... (Expected: Replica 1)
Request 10/10 sent... (Expected: Replica 2)
------------------------------------------------------------

Waiting for logs to be written...

============================================================
RESULTS:
============================================================
Total requests sent: 10

Catalog Replica 1:
  Requests handled: ~5
  Percentage: ~50.0%

Catalog Replica 2:
  Requests handled: ~5
```

```
RESULTS:
============================================================
Total requests sent: 10

Catalog Replica 1:
  Requests handled: ~5
  Percentage: ~50.0%

Catalog Replica 2:
  Requests handled: ~5
  Percentage: ~50.0%

============================================================
VERIFICATION:
============================================================
[PASS] Load balancing is working correctly!
       Requests are distributed evenly between replicas
       Expected ~5 each, got 5 and 5

============================================================
============================================================
CHECK LOGS:
============================================================
You can manually check the logs:

  logs/catalog-1.log
  logs/catalog-2.log

Look for lines containing "GET /info" to see which replica
handled each request.

============================================================
EXPLANATION:
============================================================
Load Balancing = Distributing requests across multiple servers

Algorithm: Round-Robin
  Request 1 -> Catalog Replica 1
  Request 2 -> Catalog Replica 2
```

```
============================================================
EXPLANATION:
============================================================
Load Balancing = Distributing requests across multiple servers

Algorithm: Round-Robin
  Request 1 -> Catalog Replica 1
  Request 2 -> Catalog Replica 2
  Request 3 -> Catalog Replica 1
  Request 4 -> Catalog Replica 2
  ... and so on

Benefits:
  [+] Even load distribution
  [+] No single replica is overloaded
  [+] If one replica fails, the other continues working
============================================================
```

test_4_consistency :

```
============================================================
Test 4: Replicas Consistency
============================================================

Step 1: Checking initial state of both replicas...
------------------------------------------------------------
Catalog Replica 1:
  Book: RPCs for Noobs
  Quantity: 2
  Price: 50

Catalog Replica 2:
  Book: RPCs for Noobs
  Quantity: 2
  Price: 50

[PASS] Both replicas are in sync BEFORE purchase

Step 2: Purchasing book (this should update ALL replicas)...
------------------------------------------------------------
[PASS] Purchase successful!
       Order: RPCs for Noobs
       Price: 50

Waiting for replicas to sync...

Step 3: Checking state AFTER purchase...
------------------------------------------------------------
Catalog Replica 1:
  Quantity: 1
Catalog Replica 2:
  Quantity: 1

============================================================
```

```
============================================================
RESULTS:
============================================================
Quantity BEFORE purchase:
  Replica 1: 2
  Replica 2: 2

Quantity AFTER purchase:
  Replica 1: 1
  Replica 2: 1

Change:
  Replica 1: -1
  Replica 2: -1

============================================================
VERIFICATION:
============================================================
[PASS] Test 1: Both replicas have SAME quantity
[PASS] Test 2: Quantity decreased by 1
[PASS] Test 3: BOTH replicas were updated

SUCCESS: ALL TESTS PASSED! Replicas are in sync!
============================================================
```

test_performance :

```
PS C:\Users\lenovo\Desktop\project_Distributed-and-Operating-Systems_part1-Adam-P2> node .\test_performance.js
============================================================
LAB 2 - Performance Evaluation
============================================================
Client URL: http://localhost:3000
Number of requests: 100
============================================================

[TEST 1] Response Time WITHOUT Cache
------------------------------------------------------------
Testing /info endpoint (first request - cache miss)...
  Request 1: 24ms
  Request 2: 15ms
  Request 3: 7ms
  Request 4: 7ms
  Request 5: 7ms

[TEST 2] Response Time WITH Cache
------------------------------------------------------------
Testing /info endpoint (cache hits)...
  Request 1: 1ms (fromCache: true)
  Request 2: 1ms (fromCache: true)
  Request 3: 1ms (fromCache: true)
  Request 4: 0ms (fromCache: true)
  Request 5: 1ms (fromCache: true)
  ...
  Request 96: 1ms (fromCache: true)
  Request 97: 1ms (fromCache: true)
  Request 98: 1ms (fromCache: true)
  Request 99: 0ms (fromCache: true)
```

Ctrl+I to generate a command

```
[TEST 2] Response Time WITH Cache
------------------------------------------------------------
Testing /info endpoint (cache hits)...
  Request 1: 1ms (fromCache: true)
  Request 2: 1ms (fromCache: true)
  Request 3: 1ms (fromCache: true)
  Request 4: 0ms (fromCache: true)
  Request 5: 1ms (fromCache: true)
  ...
  Request 96: 1ms (fromCache: true)
  Request 97: 1ms (fromCache: true)
  Request 98: 1ms (fromCache: true)
  Request 99: 0ms (fromCache: true)
  Request 100: 0ms (fromCache: true)

[TEST 3] Cache Invalidation Cost
------------------------------------------------------------
  Invalidation 1: 0ms
  Invalidation 2: 1ms
  Invalidation 3: 0ms
  Invalidation 4: 0ms
  Invalidation 5: 1ms
  Invalidation 6: 1ms
  Invalidation 7: 1ms
  Invalidation 8: 1ms
  Invalidation 9: 1ms
  Invalidation 10: 1ms

[TEST 4] Cache Statistics
------------------------------------------------------------
  Cache Hits: 101
  Cache Misses: 5
  Cache Hit Rate: 95.28%
  Cache Invalidations: 5
  Cache Size: 0

------------------------------------------------------------
```

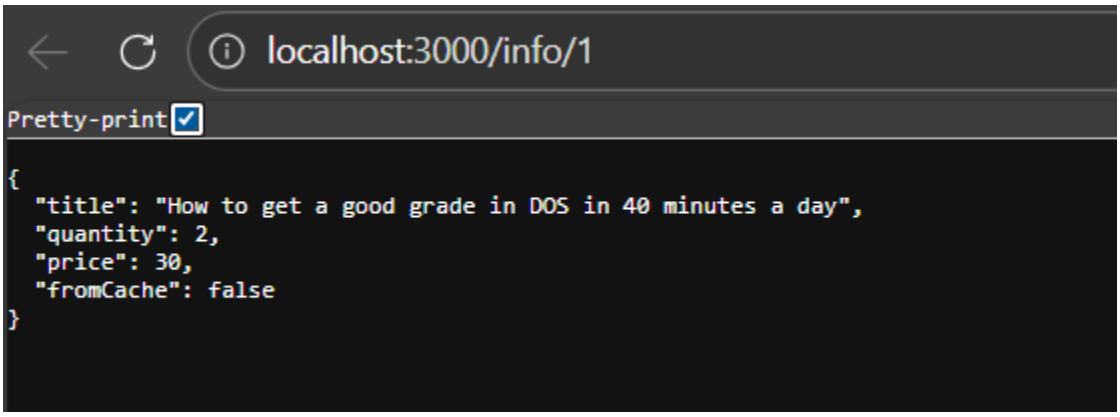
```
[TEST 4] Cache Statistics
------------------------------------------------------------
  Cache Hits: 101
  Cache Misses: 5
  Cache Hit Rate: 95.28%
  Cache Invalidations: 5
  Cache Size: 0

============================================================
RESULTS SUMMARY
============================================================

Response Time WITHOUT Cache:
  Average: 12.00ms
  Std Dev: 6.75ms
  Min: 7ms
  Max: 24ms

Response Time WITH Cache:
  Average: 0.52ms
  Std Dev: 0.50ms
  Min: 0ms
  Max: 1ms

Cache Performance:
  Speedup: 23.08x
  Improvement: 95.67%
  Cache Hit Rate: 95.28%

Cache Invalidation:
  Average Time: 0.70ms
  Min: 0ms
  Max: 1ms

============================================================
[SUCCESS] Performance evaluation completed!
============================================================
```

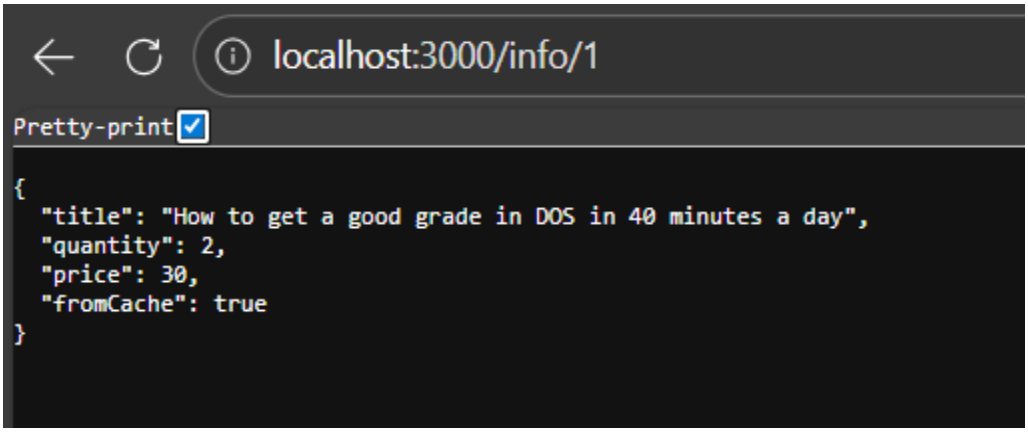
2.4 Using Docker

After that, same APIs are workig using docker exactly the same as using the services, and here some of the API testing at the browser:

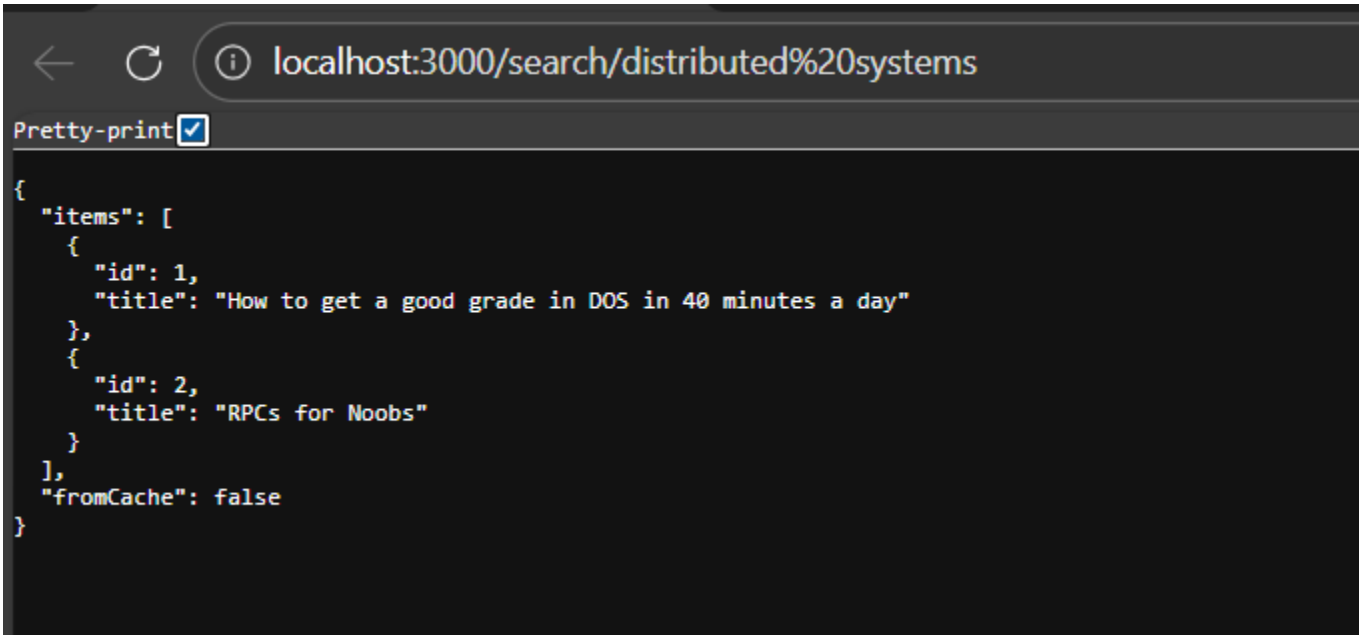
```
 ✓ project_distributed-and-operating-systems_part1-adam-p2-client            Built0.0s er-replica-2
-2    Built0.0s er-replica-1
 ✓ project_distributed-and-operating-systems_part1-adam-p2-client            Built0.0s er-replica-2
 ✓ project_distributed-and-operating-systems_part1-adam-p2-catalog-repli[+] Running 10/10
 ✓ project_distributed-and-operating-systems_part1-adam-p2-catalog-replica-2  Built0.0s .4s
 ✓ project_distributed-and-operating-systems_part1-adam-p2-order-replica-1    Built0.0s .4s
 ✓ project_distributed-and-operating-systems_part1-adam-p2-order-replica-2    Built0.0s .2s
 ✓ project_distributed-and-operating-systems_part1-adam-p2-client            Built0.0s .2s
 ✓ project_distributed-and-operating-systems_part1-adam-p2-catalog-replica-1  Built0.0s 1s
 ✓ Container catalog-replica-2                                               Recreated0.4s
 ✓ Container catalog-replica-1                                               Recreated0.4s
 ✓ Container order-replica-1                                                 Recreated0.2s
 ✓ Container order-replica-2                                                 Recreated0.2s
 ✓ Container client                                                          Created0.1s
Attaching to catalog-replica-1, catalog-replica-2, client, order-replica-1, order-replica-2
catalog-replica-1  | catalog_service listening on port 3001
catalog-replica-2  | catalog_service listening on port 3001
order-replica-1    | order_service listening on port 3002
order-replica-2    | order_service listening on port 3002
client             | client_service listening on port 3000

v View in Docker Desktop   o View Config   w Enable Watch
```

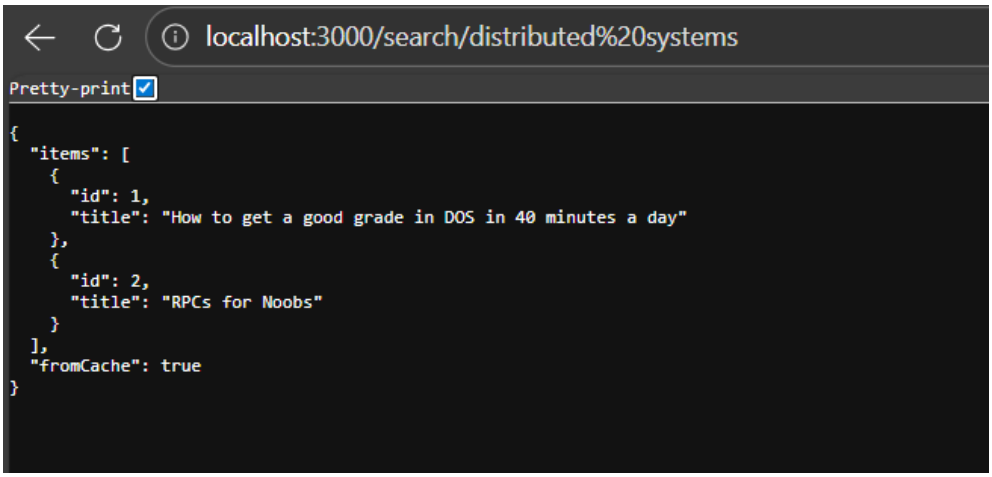
➔ localhost:3000/info/1



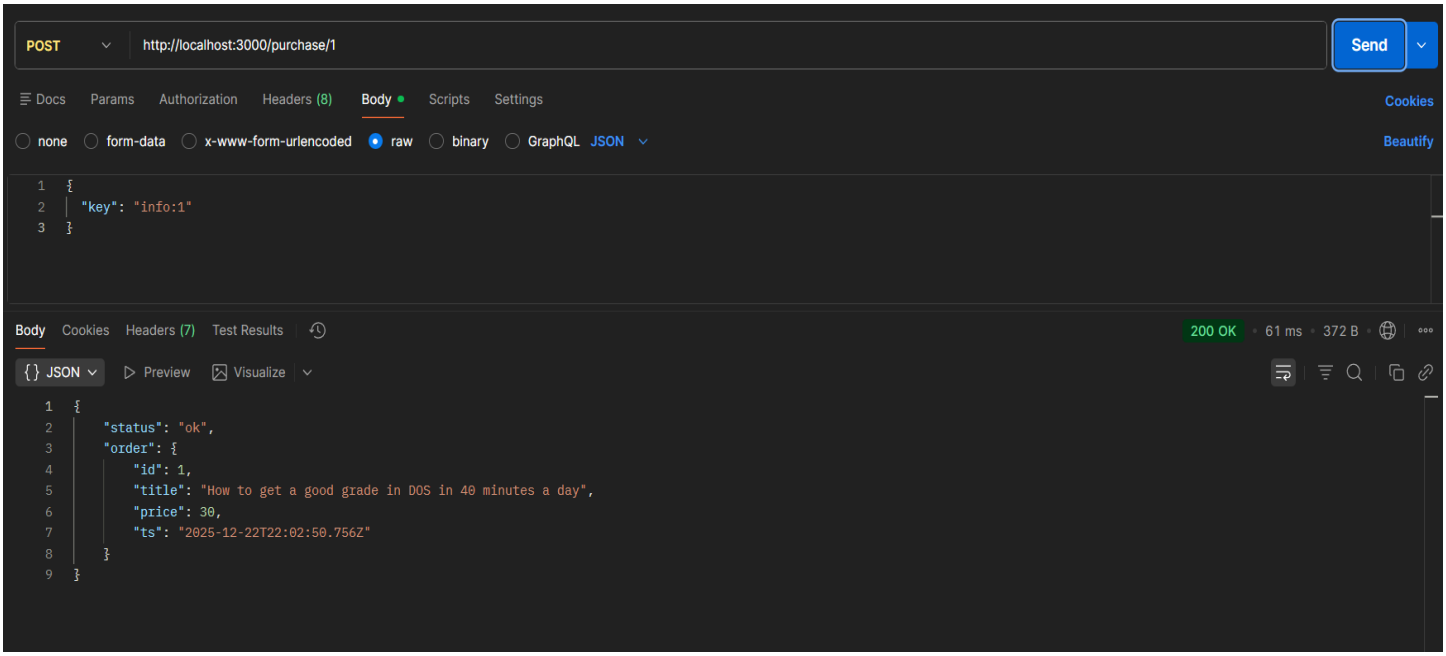


➔ localhost:3000/search/distributed systems



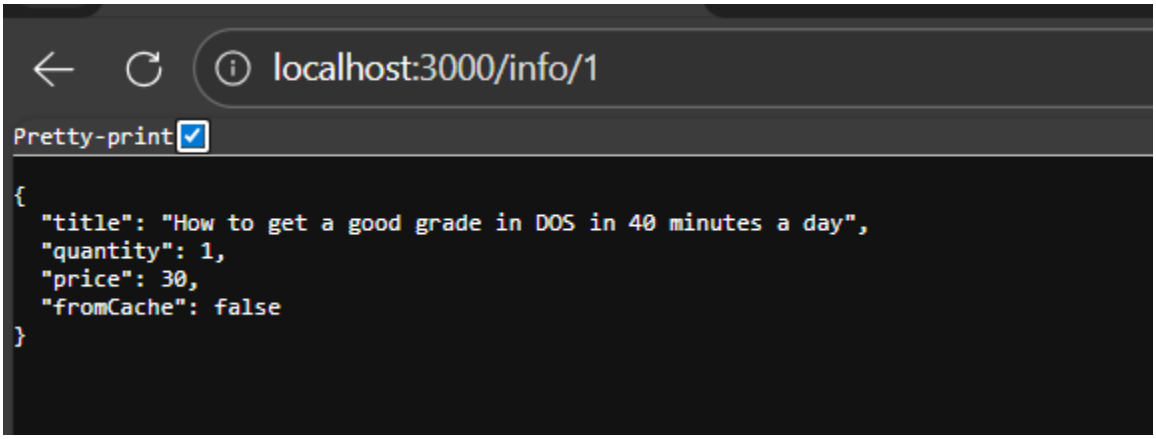


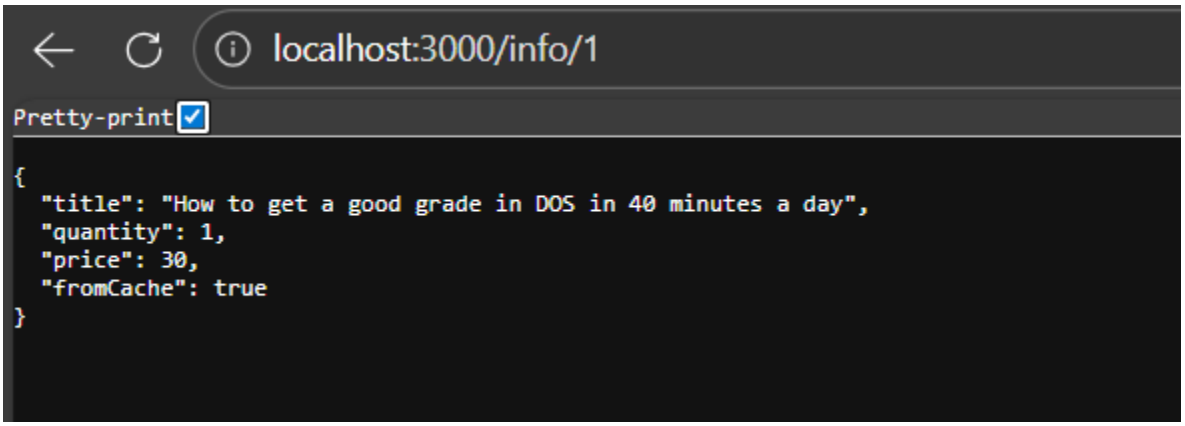
➔ http://localhost:3000/purchase/1

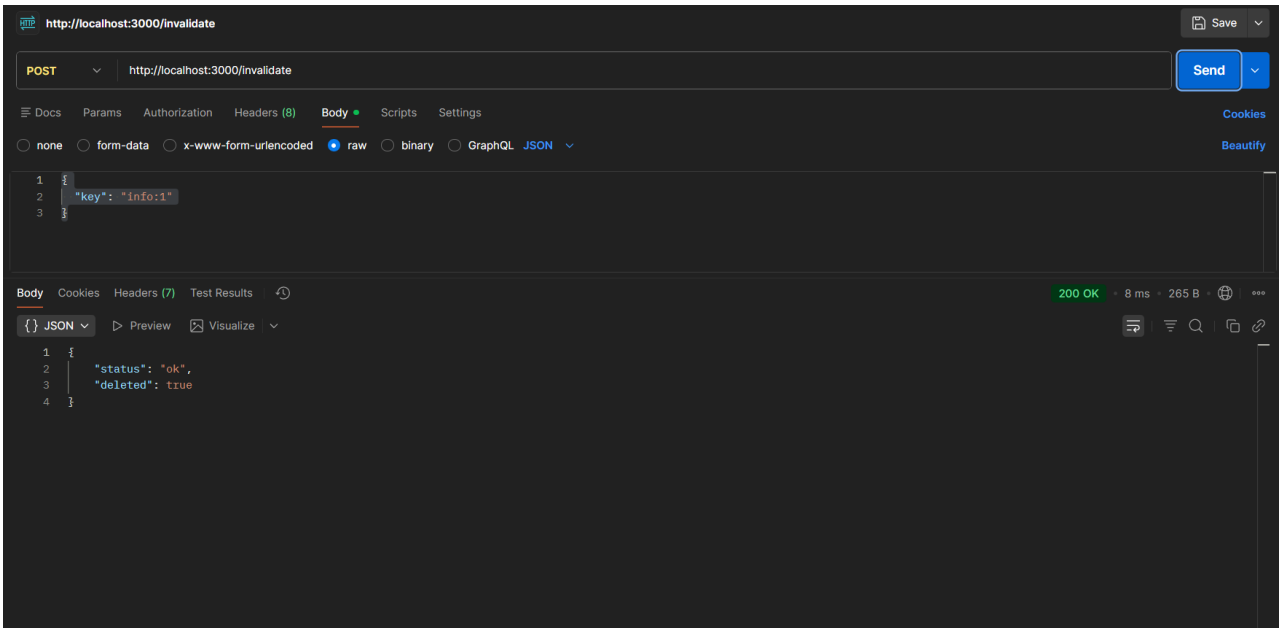


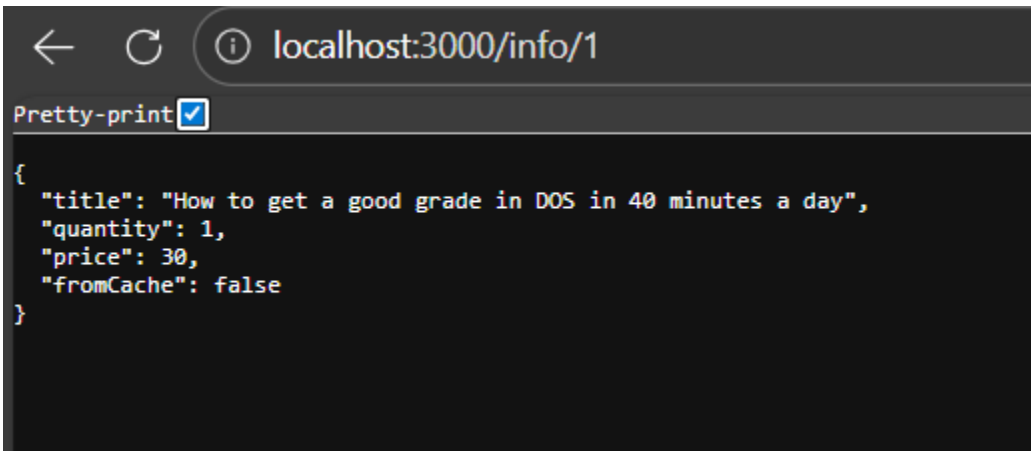
→ localhost:3000/cache/stats

➔ localhost:3000/info/1









2.5 Validation Summary

The testing confirms:

- Correct API functionality.
- Effective caching behavior.
- Proper cache invalidation after updates.
- Balanced load distribution across replicas.
- Consistent data across replicated services.

---

3. Performance Evaluation

Performance experiments were conducted to measure:

- Average response time with caching.
- Average response time without caching.
- Cache hit rate.
- Cache invalidation overhead.

Results demonstrate that caching significantly reduces response time for repeated read requests, while invalidation overhead remains minimal compared to network latency.

(Tables and graphs are included in the appendix.)

4. Conclusion

The implementation of Project Part 2 has effectively extended the functionalities of the original system introduced in Project Part 1 through replication, load balancing, and caching techniques that ensure consistency. The testing of the project has certifications of faster execution time during repeated reading tasks without altering the correctness of the process despite writing operations.

GitHub Repo. Link : https://github.com/Ameed-salameh/project_Distributed-and-Operating-Systems_part1/tree/Final_project_delivery