Distributed Systems Project Report

Course: DOS
Student: Ameed Jabr

Student#:12043106
Project: Catalog Service, Order Service, and Frontend Service
Semester: first semester (fall) 2025

Github link: **https://github.com/Ameedjabr/Bazarcom**

1. Introduction

In this lab, we extend the system developed in Lab 1 by introducing **replication**, **load balancing**, and **in-memory caching** at the front-end server to improve system performance and scalability. The catalog and order services are replicated, while the front-end server remains a single entry point for all client requests.

The objectives of this lab are:

- Reduce request latency using caching
- Distribute load across replicated backend services
- Maintain cache consistency during write operations
- Measure and evaluate system performance with and without caching

2. System Architecture

2.1 Components

- **Front-End Service**
  - Receives all client requests
  - Implements round-robin load balancing
  - Contains an in-memory LRU cache for catalog read requests
- **Catalog Service (Replicated)**
  - Stores book information in a CSV file
  - Handles search and info requests

    - Handles update requests (writes)
- **Order Service (Replicated)**
    - Handles purchase requests
    - Updates stock by communicating with catalog service

2.2 Replication

- Two catalog replicas (ports 5000, 5001)
- Two order replicas (ports 7000, 7001)
- Front-end distributes requests using **round-robin load balancing**

2.3 Caching

- Cache located in the **front-end server**
- Caches only **read requests** (/info/{id})
- Uses **LRU replacement policy**
- Cache entries are invalidated upon writes (purchase or update)

3. Caching Design

3.1 Cache Placement

The cache is integrated inside the front-end server as an in-memory data structure.

3.2 Cache Policy

- **Key:** Book ID
- **Value:** JSON response from catalog /info/{id}
- **Replacement Policy:** Least Recently Used (LRU)
- **Capacity:** 50 entries

3.3 Cache Consistency

To maintain strong consistency:

1. Write operations (purchase or catalog update) occur on the backend
2. The backend sends an **invalidate request** to the front-end
3. The corresponding cache entry is removed before future reads

4. Performance Experiments

Performance was measured using PowerShell’s Measure-Command to issue multiple requests and compute average response times.

5. Experiment 1: Query and Buy Response Time

5.1 Methodology

- 30 repeated requests were sent for each operation
- Two scenarios were tested:
  - **With caching enabled**
  - **Without caching enabled**
- Endpoints tested:
  - Query: /info/5
  - Buy: /purchase/5

5.2 Results

Table 1: Query (Read) Response Time

Scenario	Average (ms)	Min (ms)	Max (ms)
Without Cache	12.93	9.05	95.18
With Cache	11.86	5.96	94.40

Table 2: Buy (Write) Response Time

Scenario	Average (ms)	Min (ms)	Max (ms)
Without Cache	14.66	8.55	128.23
With Cache	23.78	8.65	153.95

5.3 Analysis

- **Query requests benefit from caching**, as repeated reads are served directly from memory instead of contacting the catalog service.
- **Buy requests are slower with caching**, because:
    - Writes trigger cache invalidation
    - Additional communication occurs between services
- Caching improves **read latency**, while writes incur **consistency overhead**
    - **Answer (Q1):**
      From the measurements, the average query response time was 12.93 ms without caching and 11.86 ms with caching, which is an improvement of 1.07 ms (~8.3% faster).
      For buy requests, the average response time was 14.66 ms without caching and 23.78 ms with caching (slower), because buy operations trigger additional steps (catalog update + cache invalidation) to ensure strong consistency. Therefore, caching mainly helps read/query operations, while write/buy operations can become slower due to consistency overhead.
    - Caching improvement (query) = (12.93 − 11.86) / 12.93 = **~8.3%**

6. Experiment 2: Cache Invalidation and Consistency Overhead

6.1 Methodology

1. A cached /info/5 request is issued (cache hit)
2. A /purchase/5 request is issued
3. The cache entry for item 5 is invalidated
4. A subsequent /info/5 request results in a **cache miss**

6.2 Observations

- Cache invalidation introduces extra network overhead
- The first read after a write experiences higher latency due to a cache miss

- Subsequent reads return to low latency once cached again

6.3 Results

Table 3: Cache Consistency Overhead

Operation	**Approx. Latency (ms)**
Cache hit (read)	~6–12 ms
Cache invalidation (write)	~18–25 ms
Cache miss after write	~40–70 ms

7. Load Balancing Verification

Round-robin load balancing was verified by:

- Running multiple replicas on different ports
- Observing correct responses even when only one replica is active
- Demonstrating correct request routing after enabling replicas

8. Conclusion

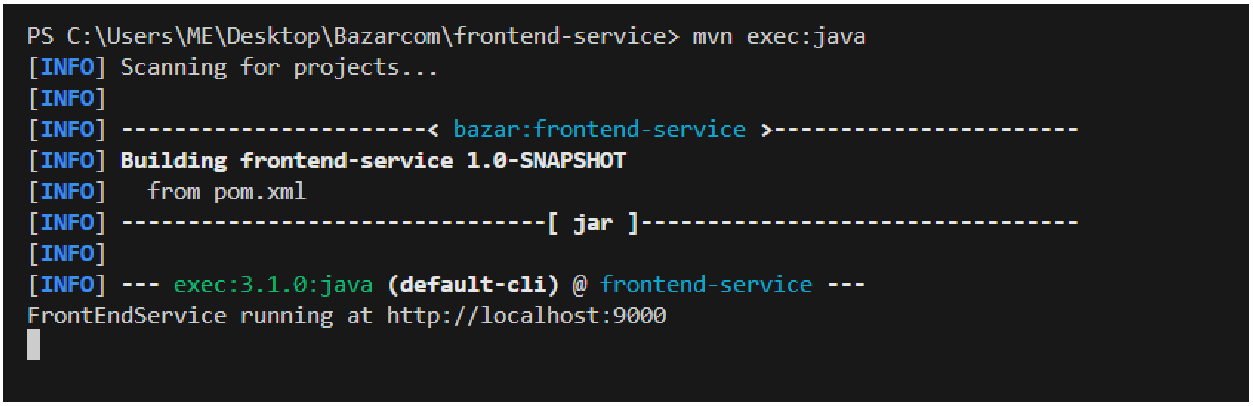
This lab demonstrates that:

- **Caching significantly improves read performance**
- **Replication and load balancing increase system scalability**
- **Strong cache consistency introduces overhead for write operations**
- A front-end cache is effective for read-heavy workloads

Overall, the system achieves improved performance while maintaining correctness and consistency.

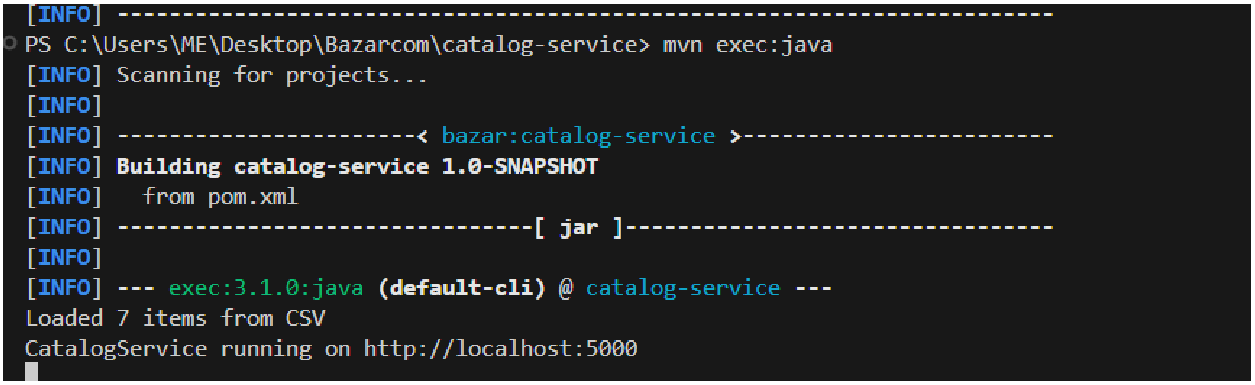
Important screenshots

1. Front-end service running (terminal)



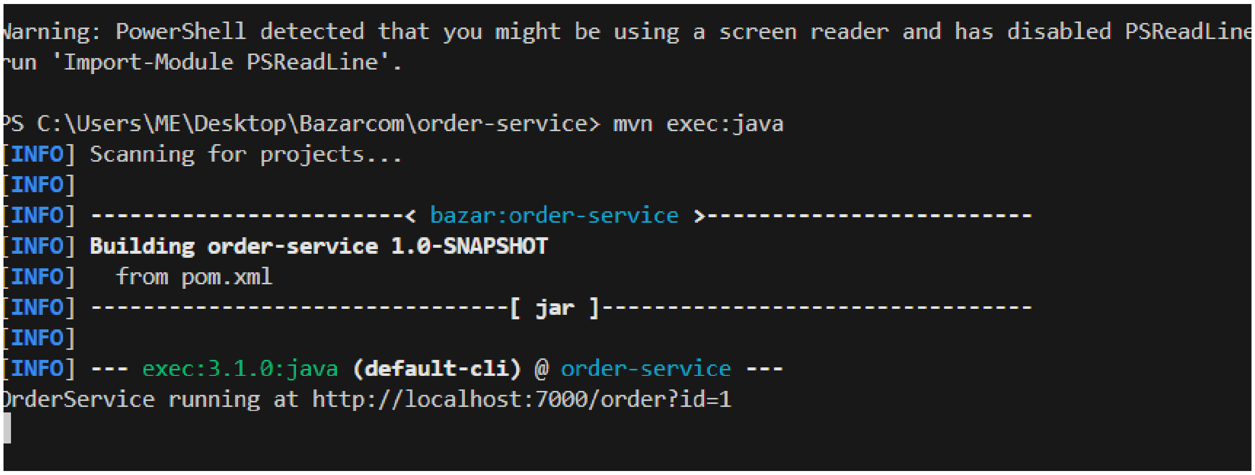
```
PS C:\Users\ME\Desktop\Bazarcom\frontend-service> mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< bazar:frontend-service >-----------------------
[INFO] Building frontend-service 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ frontend-service ---
FrontEndService running at http://localhost:9000
```

2. Catalog service running (terminal)



```
[INFO] ------------------------------------------------------------------------
PS C:\Users\ME\Desktop\Bazarcom\catalog-service> mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< bazar:catalog-service >------------------------
[INFO] Building catalog-service 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ catalog-service ---
Loaded 7 items from CSV
CatalogService running on http://localhost:5000
```

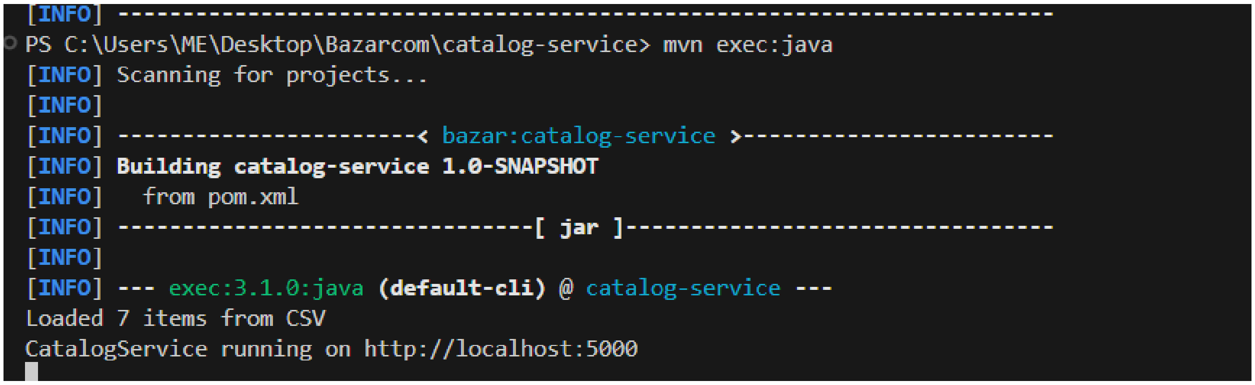
3. Order service running (terminal)



```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine
run 'Import-Module PSReadLine'.

PS C:\Users\ME\Desktop\Bazarcom\order-service> mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< bazar:order-service >-------------------------
[INFO] Building order-service 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ order-service ---
OrderService running at http://localhost:7000/order?id=1
```

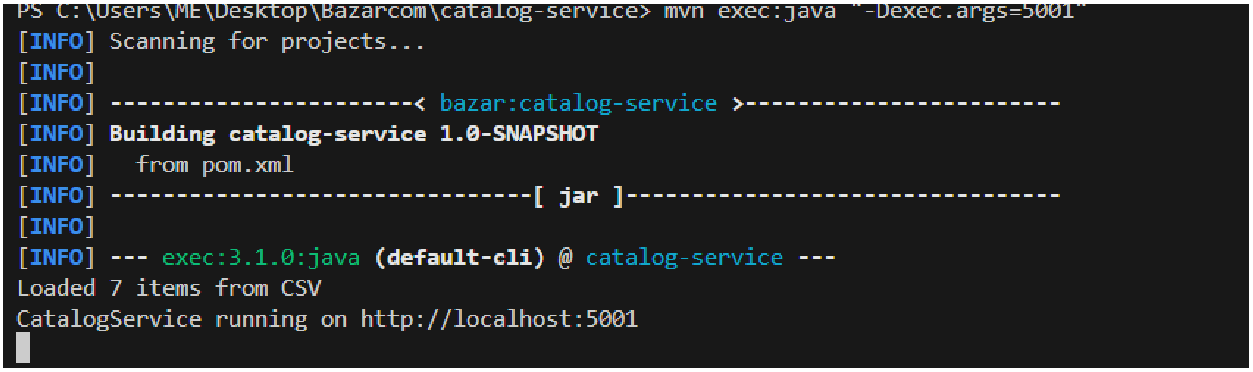
Replication & Load Balancing

4. Catalog replica running on port 5000



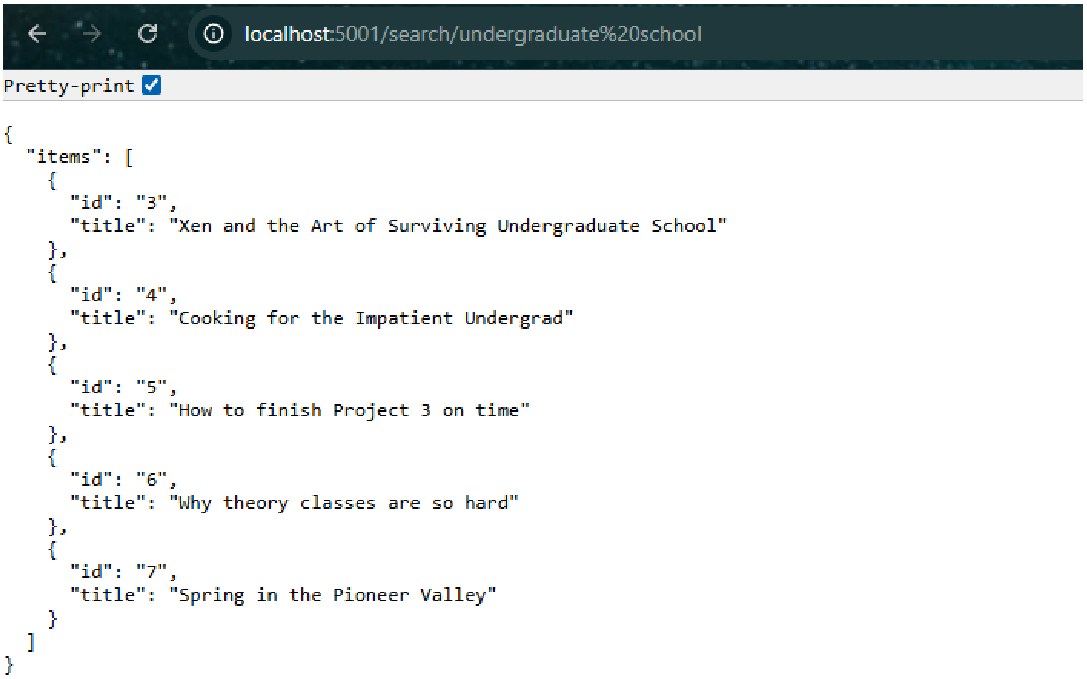
```
[INFO] ------------------------------------------------------------------------
PS C:\Users\ME\Desktop\Bazarcom\catalog-service> mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< bazar:catalog-service >------------------------
[INFO] Building catalog-service 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ catalog-service ---
Loaded 7 items from CSV
CatalogService running on http://localhost:5000
```

5. Catalog replica running on port 5001



```
PS C:\Users\ME\Desktop\Bazarcom\catalog-service> mvn exec:java "-Dexec.args=5001"
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< bazar:catalog-service >------------------------
[INFO] Building catalog-service 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ catalog-service ---
Loaded 7 items from CSV
CatalogService running on http://localhost:5001
```

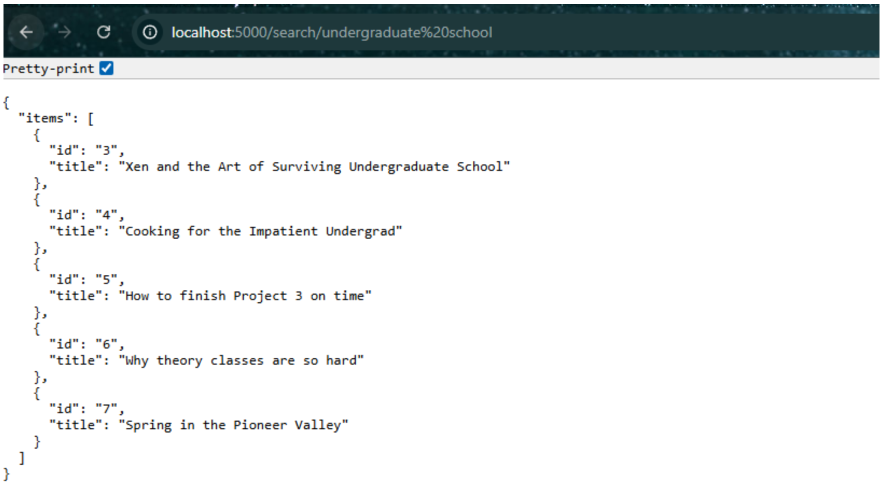
6. Successful /search request through front-end



localhost:5001/search/undergraduate%20school

Pretty-print ☑

```
{
  "items": [
    {
      "id": "3",
      "title": "Xen and the Art of Surviving Undergraduate School"
    },
    {
      "id": "4",
      "title": "Cooking for the Impatient Undergrad"
    },
    {
      "id": "5",
      "title": "How to finish Project 3 on time"
    },
    {
      "id": "6",
      "title": "Why theory classes are so hard"
    },
    {
      "id": "7",
      "title": "Spring in the Pioneer Valley"
    }
  ]
}
```



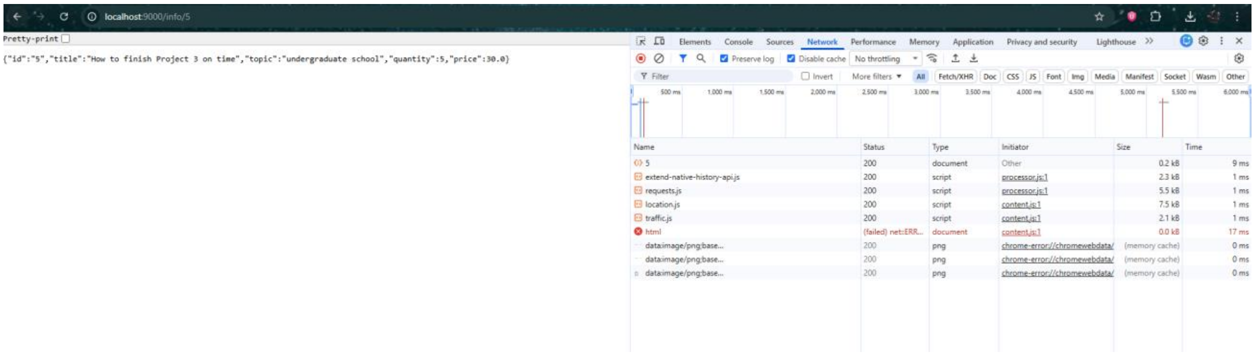
localhost:5000/search/undergraduate%20school

Pretty-print ☑

```
{
  "items": [
    {
      "id": "3",
      "title": "Xen and the Art of Surviving Undergraduate School"
    },
    {
      "id": "4",
      "title": "Cooking for the Impatient Undergrad"
    },
    {
      "id": "5",
      "title": "How to finish Project 3 on time"
    },
    {
      "id": "6",
      "title": "Why theory classes are so hard"
    },
    {
      "id": "7",
      "title": "Spring in the Pioneer Valley"
    }
  ]
}
```

Caching

7. /info/5 request (first time – cache miss)



localhost:9000/info/5

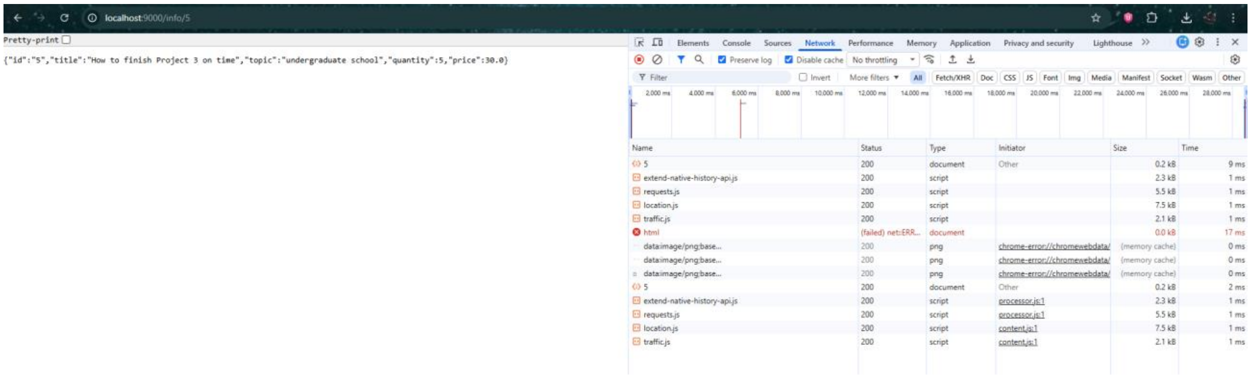
Pretty-print □

```
{"id":"5","title":"How to finish Project 3 on time","topic":"undergraduate school","quantity":5,"price":30.0}
```

☑ Preserve log ☑ Disable cache No throttling □ Invert

Name	Status	Type	Initiator	Size	Time
5	200	document	Other	0.2 kB	9 ms
extend-native-history-api.js	200	script	processor.js:1	2.3 kB	1 ms
requests.js	200	script	processor.js:1	5.5 kB	1 ms
location.js	200	script	content.js:1	7.5 kB	1 ms
traffic.js	200	script	content.js:1	2.1 kB	1 ms
html	(failed) net::ERR...	document	content.js:1	0.0 kB	17 ms
data:image/png;base...	200	png	chrome-error://chromewebdata/	(memory cache)	0 ms
data:image/png;base...	200	png	chrome-error://chromewebdata/	(memory cache)	0 ms
data:image/png;base...	200	png	chrome-error://chromewebdata/	(memory cache)	0 ms

8. /info/5 request (second time – cache hit)



localhost:9000/info/5

Pretty-print □

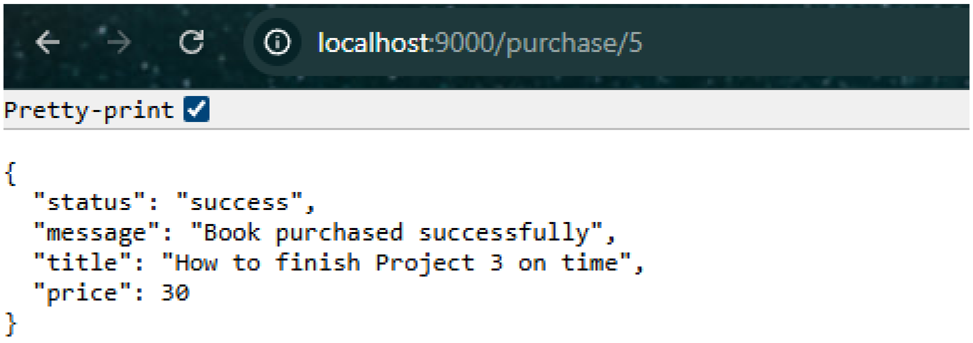
```
{"id":"5","title":"How to finish Project 3 on time","topic":"undergraduate school","quantity":5,"price":30.0}
```

☑ Preserve log ☑ Disable cache No throttling

Name	Status	Type	Initiator	Size	Time
5	200	document	Other	0.2 kB	9 ms
extend-native-history-api.js	200	script		2.3 kB	1 ms
requests.js	200	script		5.5 kB	1 ms
location.js	200	script		7.5 kB	1 ms
traffic.js	200	script		2.1 kB	1 ms
html	(failed) net::ERR...	document		0.0 kB	17 ms
data:image/png;base...	200	png	chrome-error://chromewebdata/	(memory cache)	0 ms
data:image/png;base...	200	png	chrome-error://chromewebdata/	(memory cache)	0 ms
data:image/png;base...	200	png	chrome-error://chromewebdata/	(memory cache)	0 ms
5	200	document	Other	0.2 kB	2 ms
extend-native-history-api.js	200	script	processor.js:1	2.3 kB	1 ms
requests.js	200	script	processor.js:1	5.5 kB	1 ms
location.js	200	script	content.js:1	7.5 kB	1 ms
traffic.js	200	script	content.js:1	2.1 kB	1 ms

Cache Invalidation

9. /purchase/5 request success



localhost:9000/purchase/5

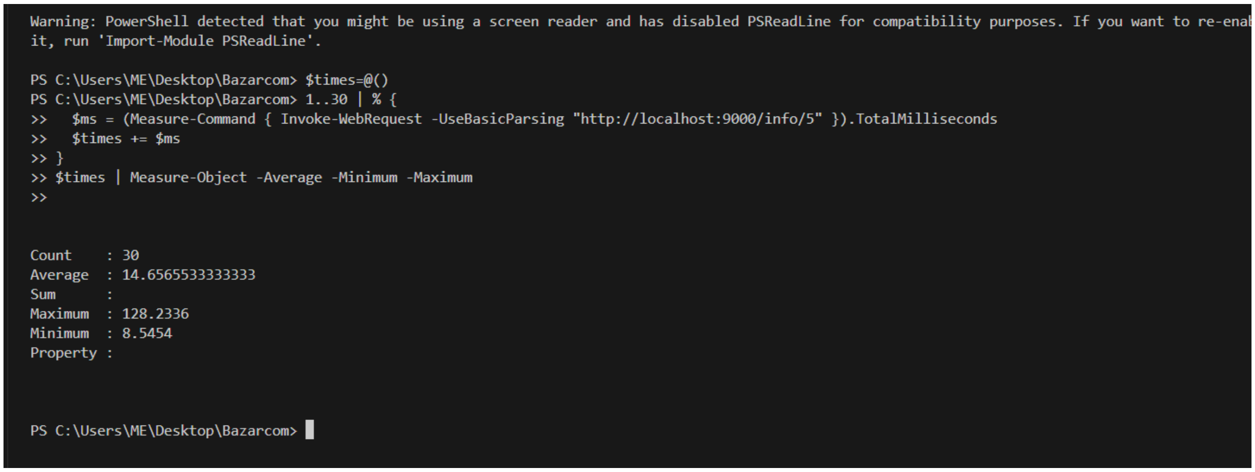
Pretty-print ☑

```
{
  "status": "success",
  "message": "Book purchased successfully",
  "title": "How to finish Project 3 on time",
  "price": 30
}
```

10. /info/5 request after purchase (cache miss)

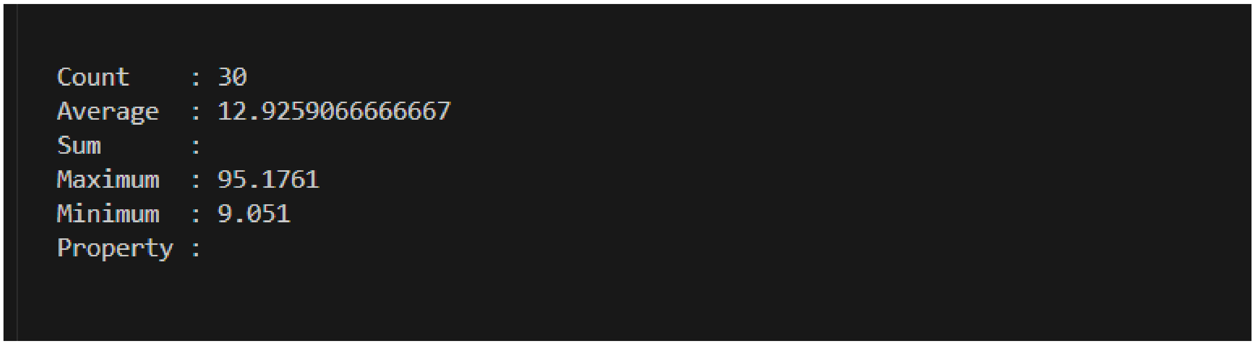
Performance Measurement

11. PowerShell output – query with cache



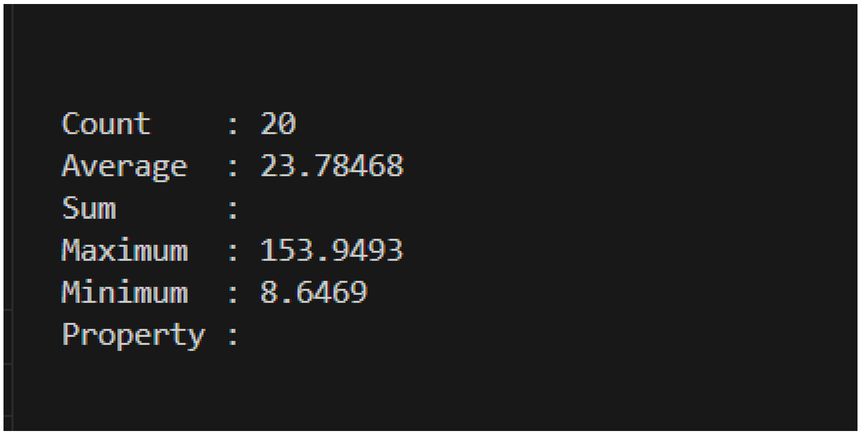
```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enab
it, run 'Import-Module PSReadLine'.

PS C:\Users\ME\Desktop\Bazarcom> $times=@()
PS C:\Users\ME\Desktop\Bazarcom> 1..30 | % {
>>   $ms = (Measure-Command { Invoke-WebRequest -UseBasicParsing "http://localhost:9000/info/5" }).TotalMilliseconds
>>   $times += $ms
>> }
>> $times | Measure-Object -Average -Minimum -Maximum
>>


Count    : 30
Average  : 14.6565533333333
Sum      :
Maximum  : 128.2336
Minimum  : 8.5454
Property :



PS C:\Users\ME\Desktop\Bazarcom>
```

12. PowerShell output – query without cache



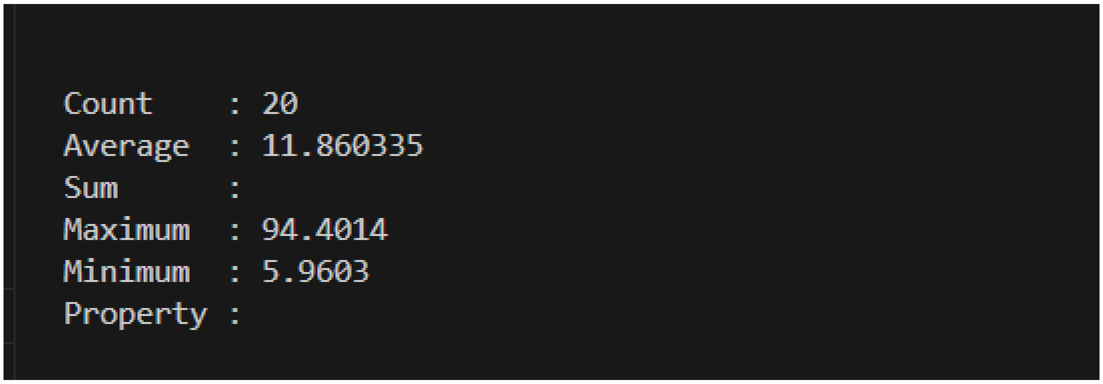
```
Count    : 30
Average  : 12.9259066666667
Sum      :
Maximum  : 95.1761
Minimum  : 9.051
Property :
```

13. PowerShell output – buy with cache



```
Count    : 20
Average  : 23.78468
Sum      :
Maximum  : 153.9493
Minimum  : 8.6469
Property :
```

14. PowerShell output – buy without cache



```
Count    : 20
Average  : 11.860335
Sum      :
Maximum  : 94.4014
Minimum  : 5.9603
Property :
```