Project Report

AL-NAJAH UNIVERSITY

DOS

SUBMITTED TO:

Dr.Samer Arandi

PREPARED BY:

Shahd Khader

12027906

Ahmad Nazzal

12027687

Introduciton:

In this lab, I restructured the online bookstore, Bazar.com, which I developed in Lab 1. The goal is to improve request processing time as the system struggles to handle a higher volume of requests due to increased customer demand. To achieve this, I introduced replication, caching, and other design improvements to create a more scalable and efficient architecture. This project also aims to teach concepts related to multi-tier web design, microservices, and containerization using Docker.

The following report details the modifications made to the system to support replication, caching, and consistency. Additionally, I describe the technical solutions implemented to reduce latency and enhance scalability.

New Architecture with Replication and Caching

To improve performance, I implemented a multi-tier architecture with the following components:

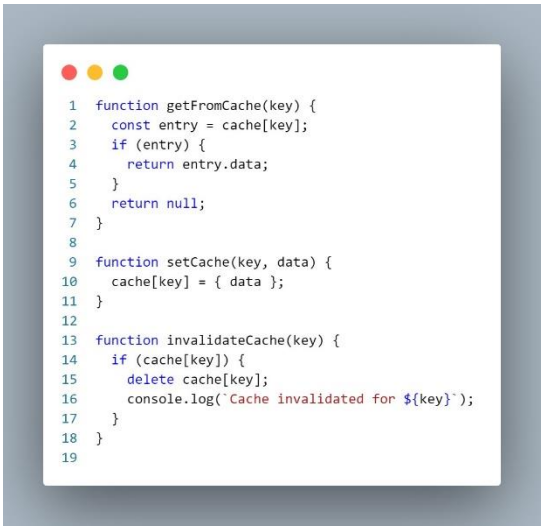
- **Front-End Server with In-Memory Cache**: The front-end now includes an in-memory cache to store frequently accessed catalog data and recent orders.
- **Replicated Catalog and Order Services**: I added multiple replicas of the catalog and order services to improve redundancy and decrease response times by distributing the load.

Implementation Details

In-Memory Caching

To reduce response time, an in-memory cache was implemented on the front-end server. This cache stores recent requests to avoid repeated database queries for the same data.

- **Cache Design**: The cache is implemented as a key-value store. Frequently requested items (e.g., popular books) and recent orders are stored in memory.



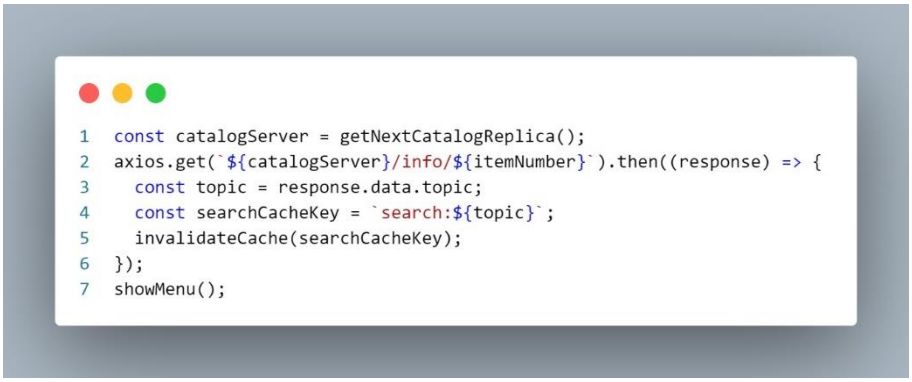
```
function getFromCache(key) {
  const entry = cache[key];
  if (entry) {
    return entry.data;
  }
  return null;
}

function setCache(key, data) {
  cache[key] = { data };
}

function invalidateCache(key) {
  if (cache[key]) {
    delete cache[key];
    console.log(`Cache invalidated for ${key}`);
  }
}

```

Data Replication

Data replication was added to improve availability and reduce latency. Multiple instances of the catalog and order services were created, with the front-end server distributing requests across these instances.

- **Replication Strategy**: I used a round-robin method for distributing requests to replicas.
- **Load Balancing**: The front-end server sends requests to catalog and order service replicas to balance the workload.

```
const cache = {};

const catalogReplicas = ["http://catalog-service-1:3001", "http://catalog-service-2:3002"];
const orderReplicas = ["http://order-service-1:3003", "http://order-service-2:3004"];

let catalogReplicaIndex = 0;
let orderReplicaIndex = 0;

function getNextCatalogReplica() {
  catalogReplicaIndex = (catalogReplicaIndex + 1) % catalogReplicas.length;
  console.log(catalogReplicas[catalogReplicaIndex]);
  return catalogReplicas[catalogReplicaIndex];
}

function getNextOrderReplica() {
  orderReplicaIndex = (orderReplicaIndex + 1) % orderReplicas.length;
  console.log(orderReplicas[orderReplicaIndex]);
  return orderReplicas[orderReplicaIndex];
}
```



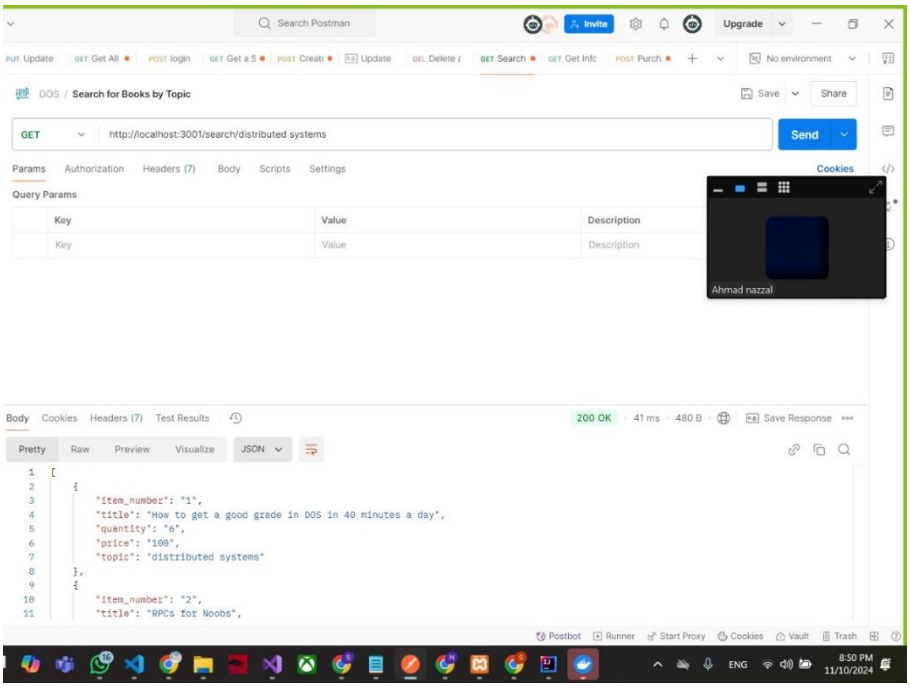
```
const catalogServer = getNextCatalogReplica();
axios.get(`${catalogServer}/info/${itemNumber}`).then((response) => {
  const topic = response.data.topic;
  const searchCacheKey = `search:${topic}`;
  invalidateCache(searchCacheKey);
});
showMenu();
```

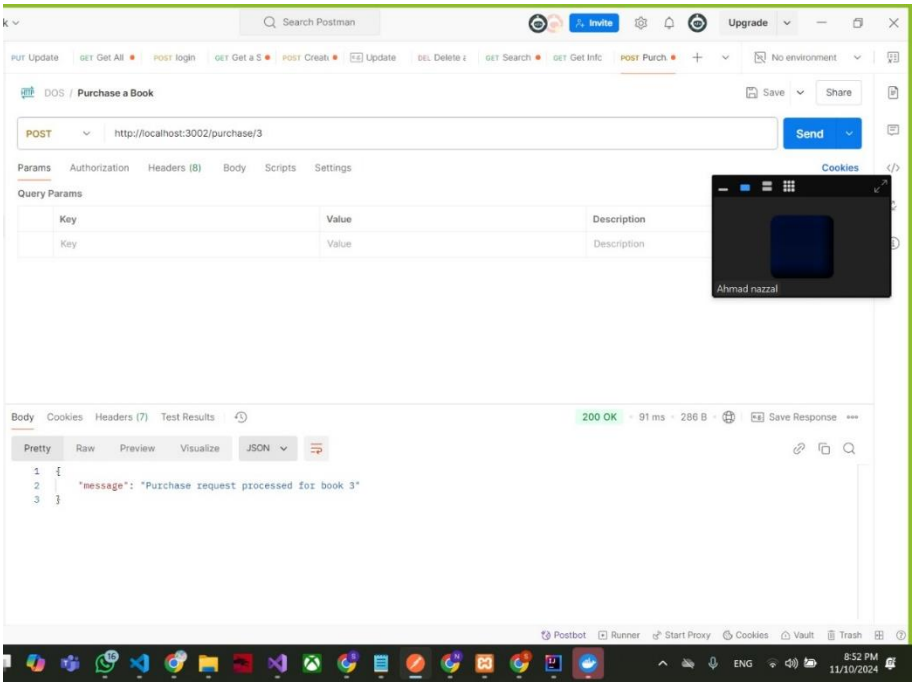
Testing and Results

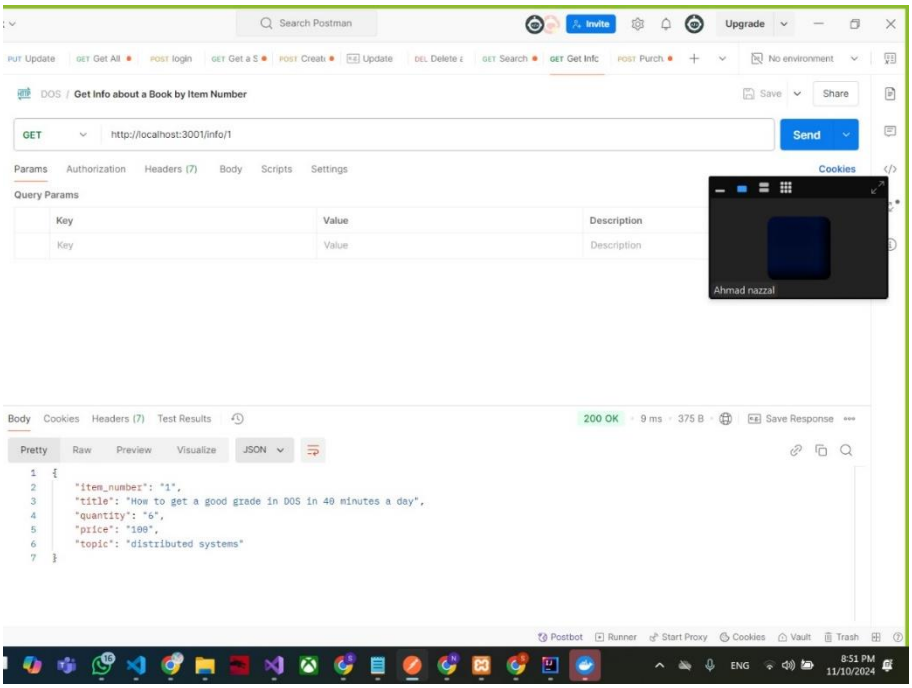
Latency Improvement

I measured the response times before and after implementing caching and replication. After the changes, the average response time decreased significantly, as shown in the following results:

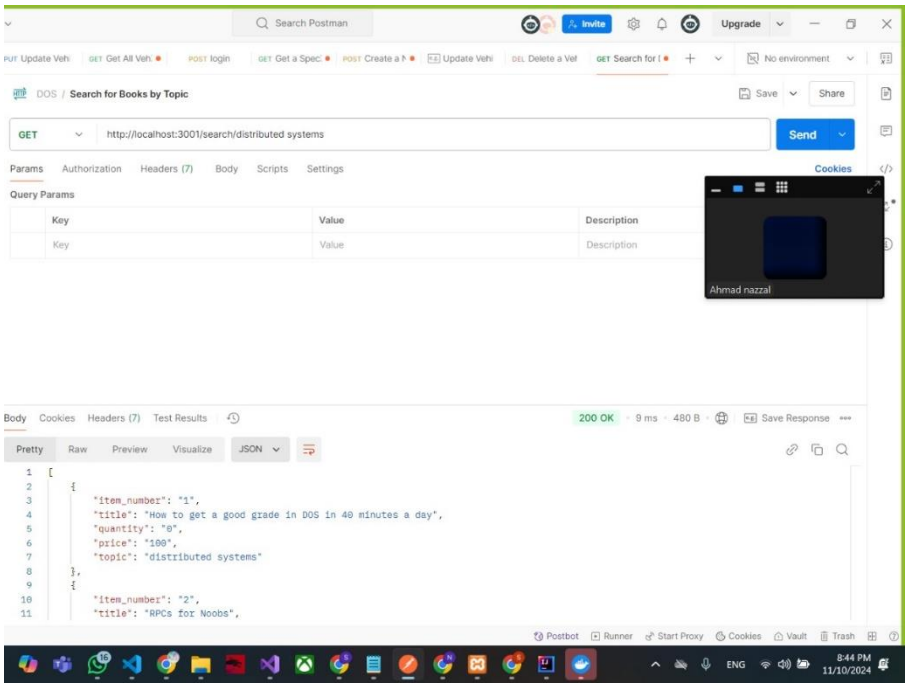
Without using cache

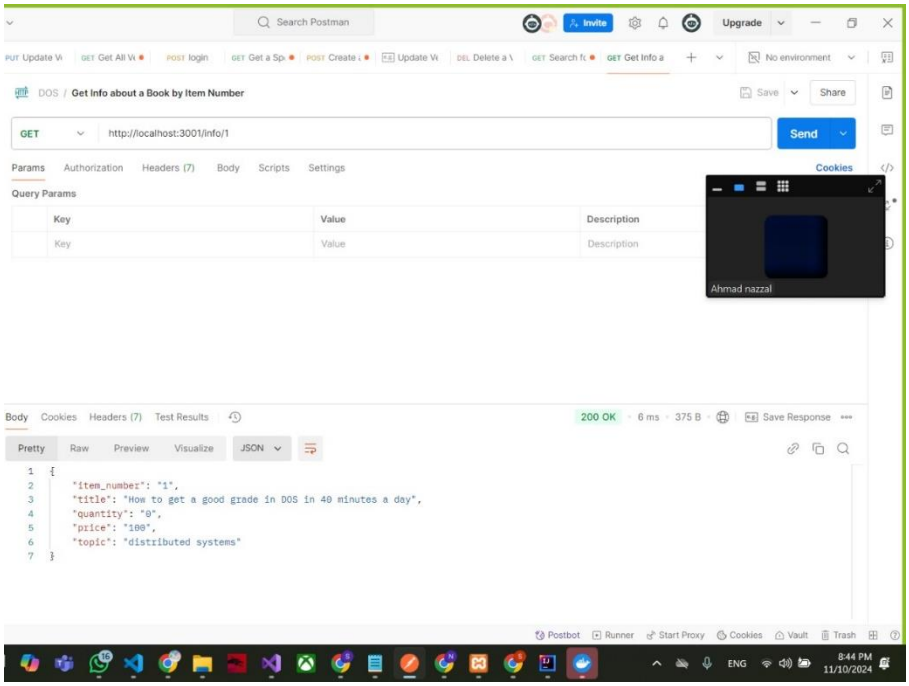


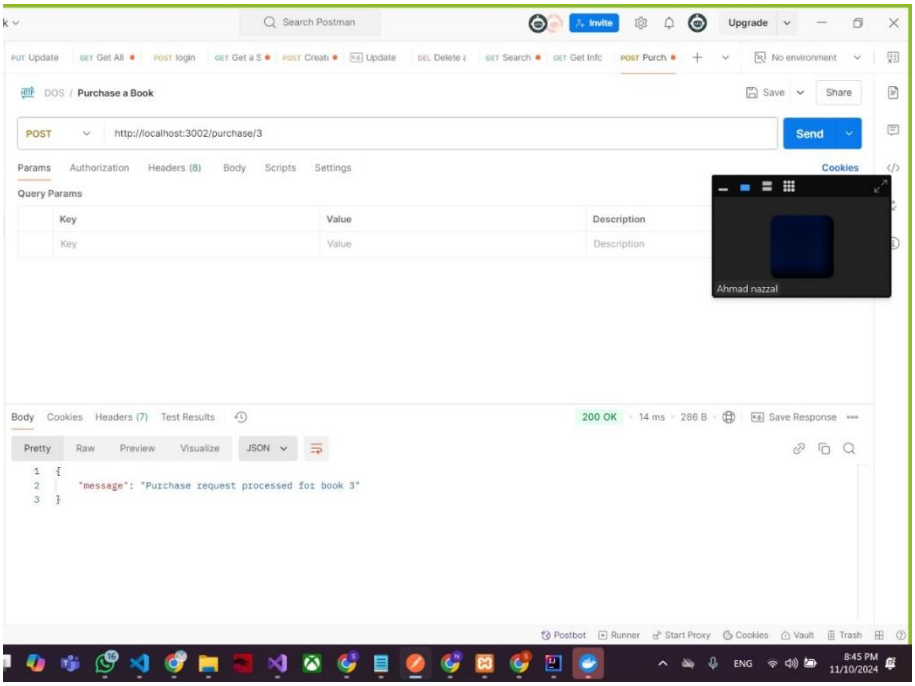




With using cache





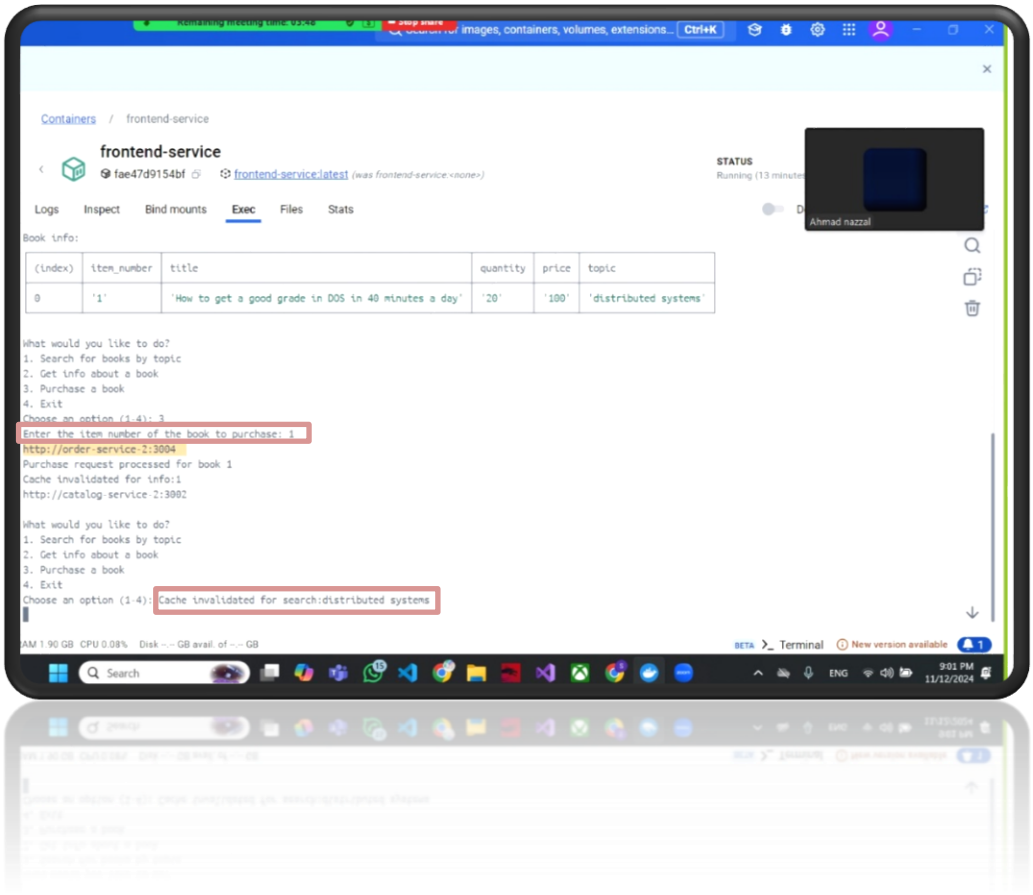


The table that shows the differences

Feature	Before (ms)	Before (ms)
Search by topic – Catalog	41	9
Search by id - Catalog	9	6
Purchase - Order	91	14

Consistency Testing

Consistency was verified by simulating concurrent requests. For example, if a book's stock was updated in one replica, all replicas reflected this change within a short time to maintain data integrity.



Optional: Containerization with Docker

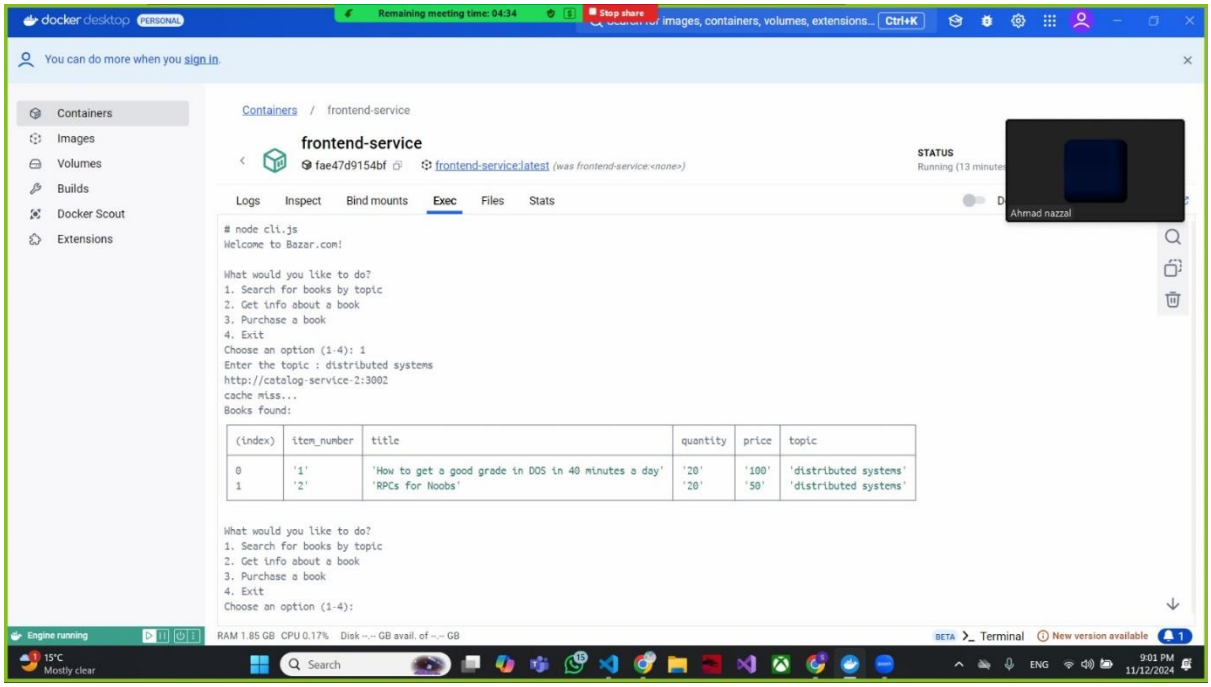
To make the system easily deployable, I containerized each service using Docker. This setup enables the application to run in isolated containers, making it easy to deploy and scale.

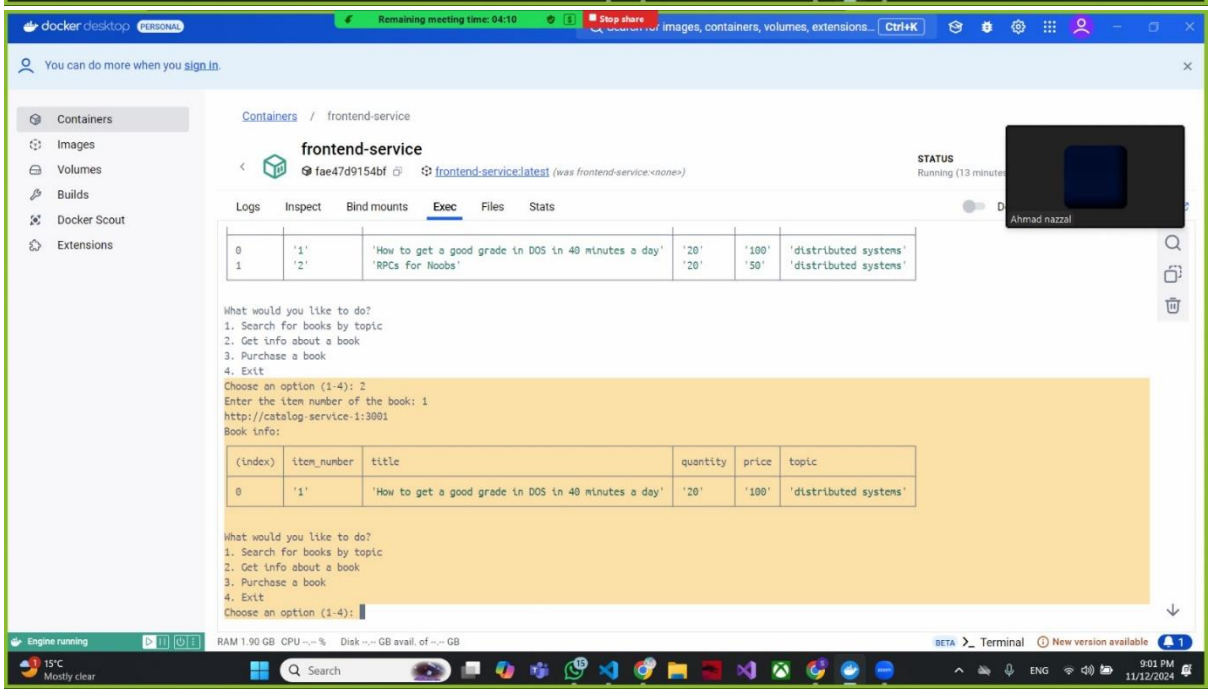
Docker-Compose File:

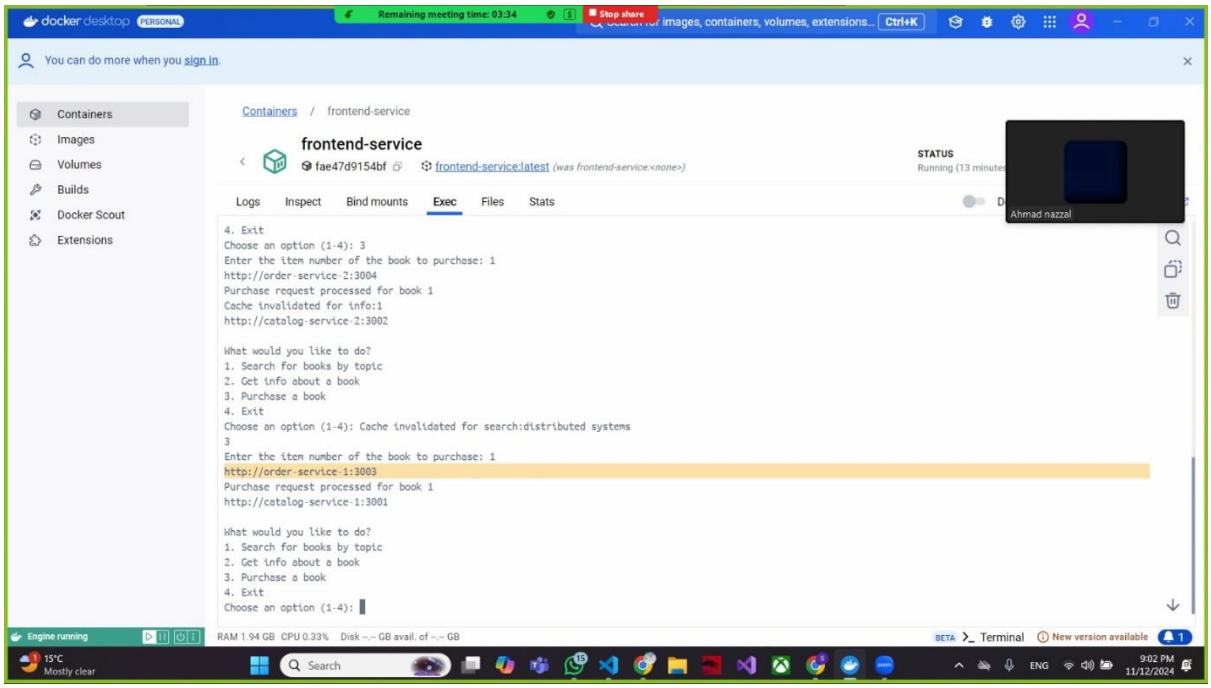
```yaml
version: "3"
services:
  catalog-service-1:
    image: catalog-service
    container_name: catalog-service-1
    build:
      context: ./catalog-service
    ports:
      - "3001:3001"
    environment:
      - PORT=3001
    volumes:
      - ./catalog-service/catalog.csv:/app/catalog.csv
    networks:
      - network1

  catalog-service-2:
    image: catalog-service
    container_name: catalog-service-2
    ports:
      - "3002:3001"
    environment:
      - PORT=3002
    volumes:
      - ./catalog-service/catalog.csv:/app/catalog.csv
    networks:
      - network1

  order-service-1:
    image: order-service
    container_name: order-service-1
    build:
      context: ./order-service
    ports:
      - "3003:3003"
    environment:
      - PORT=3003
    networks:
      - network1
    depends_on:
      - catalog-service-1
      - catalog-service-2

  order-service-2:
    image: order-service
    container_name: order-service-2
    ports:
      - "3004:3003"
    environment:
      - PORT=3004
    networks:
      - network1
    depends_on:
      - catalog-service-1
      - catalog-service-2

  frontend-service:
    image: frontend-service
    container_name: frontend-service
    build:
      context: ./frontend
    networks:
      - network1
    depends_on:
      - order-service-1
      - order-service-2
      - catalog-service-1
      - catalog-service-2
    stdin_open: true
    tty: true

networks:
  network1:
    driver: bridge

```

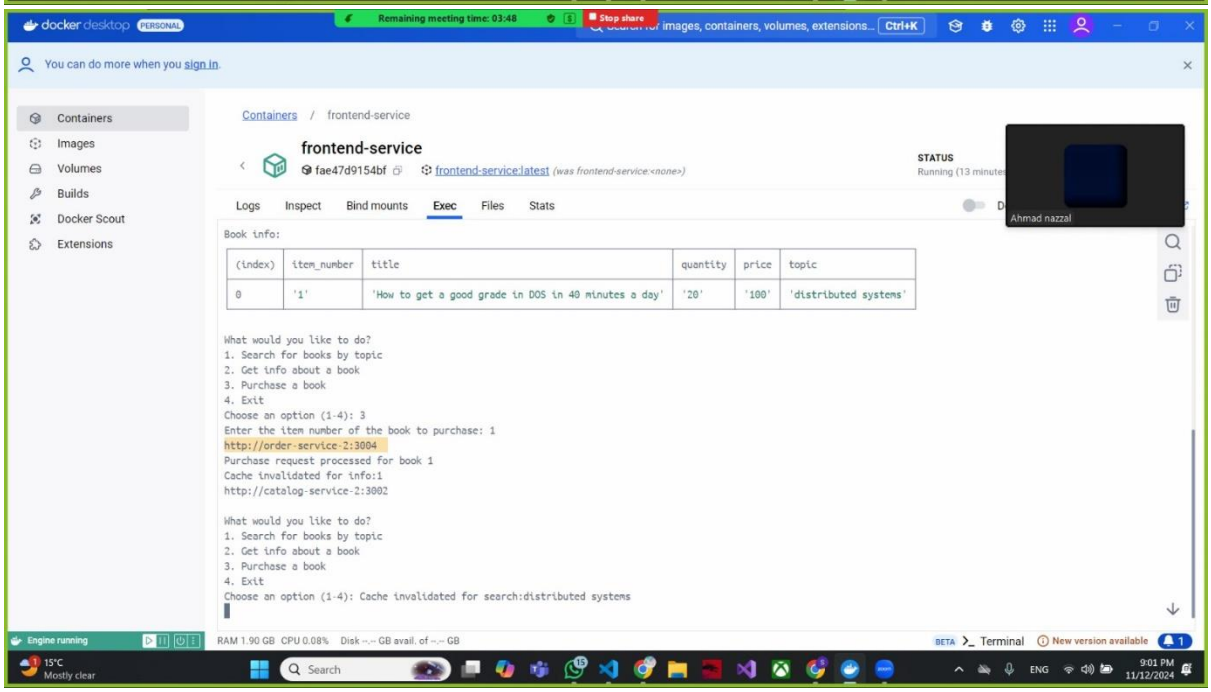
Result

The images below show each request sent to different replica and save the consistency between the containers






Containers / frontend-service
frontend-service
fae47d9154bf
frontend-service:latest
Logs Inspect Bind mounts Exec Files Stats
4. Exit
Choose an option (1-4): 3
Enter the item number of the book to purchase: 1
http://order-service-2:3004
Purchase request processed for book 1
Cache invalidated for info:1
http://catalog-service-2:3002
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): Cache invalidated for search:distributed systems
3
Enter the item number of the book to purchase: 1
http://order-service-1:3003
Purchase request processed for book 1
http://catalog-service-1:3001
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):


Containers / frontend-service
frontend-service
fae47d9154bf
frontend-service:latest
Logs Inspect Bind mounts Exec Files Stats
Book info:
(index) item_number title quantity price topic
0 '1' 'How to get a good grade in DOS in 40 minutes a day' '20' '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 of the book to purchase: 1
http://order-service-2:3004
Purchase request processed for book 1
Cache invalidated for info:1
http://catalog-service-2:3002
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): Cache invalidated for search:distributed systems

Good Buy 😊