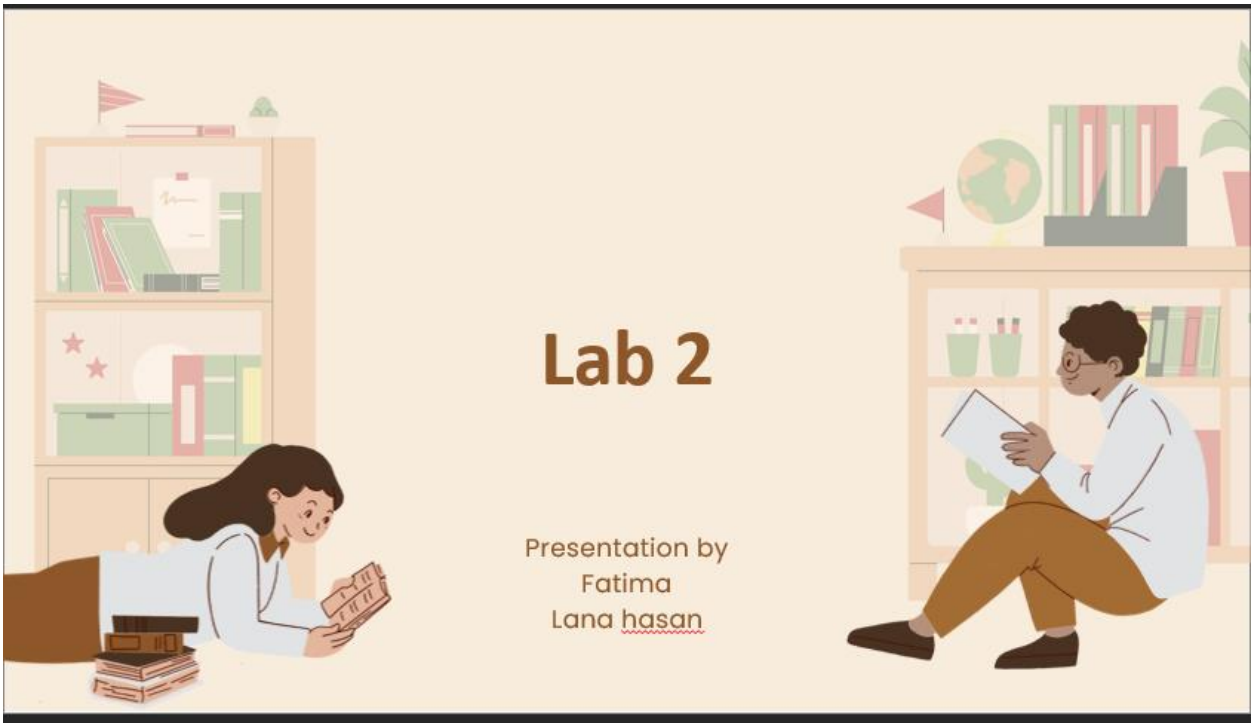


Dos project part2

INTODUCTION:

Continuing from the previous lab (Lab 1) where we developed a book management application using Python and Flask, we worked on improving the performance and efficiency of the application. Key improvements include enhancing redundancy to ensure data availability and continuity, using caching to speed up response to repeated requests, implementing load balancing to distribute requests evenly across servers, and ensuring consistency between different copies of data for a smoother and more stable user experience.

Operation:

Implementation of improved mechanisms in the application:

1.Redundancy:

Two replicas of the catalog and request servers were created, with each replica running on a separate IP address (one on the local machine and the other on the VMware platform). This redundancy aims to ensure continuous service availability and maintain connectivity using a shared port, ensuring that the application continues to run even in the event of any failure in one of the servers.

2.Consistency:

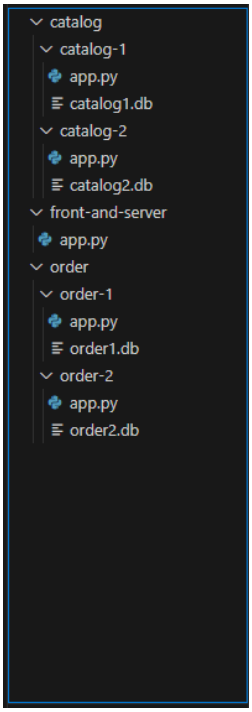
A caching mechanism using the "least recently used" (LRU) principle via the medium (cashMid) was adopted. This mechanism speeds up the retrieval of frequently requested data from the cache, reducing the need to execute external queries and improving the overall efficiency of the application.

3.Load balancing:

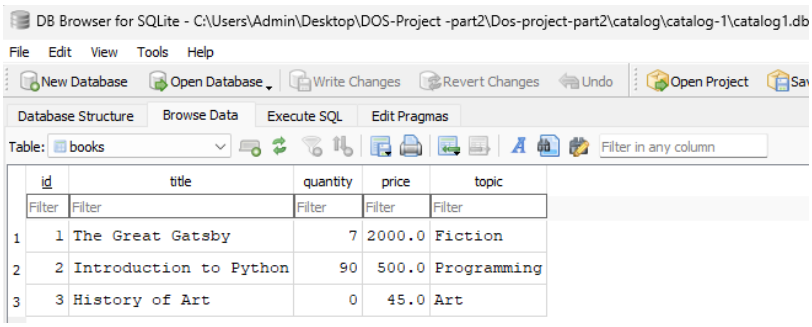
A cycle distribution algorithm was implemented using two dedicated counters to manage traffic between the catalog and request servers. The replicas (1 and 2) are dynamically switched to distribute the load evenly. The use of variable ports and the assignment of an IP address to each server also contribute to improving the efficiency of load balancing and providing stable and fast response to users.

Test and run project:

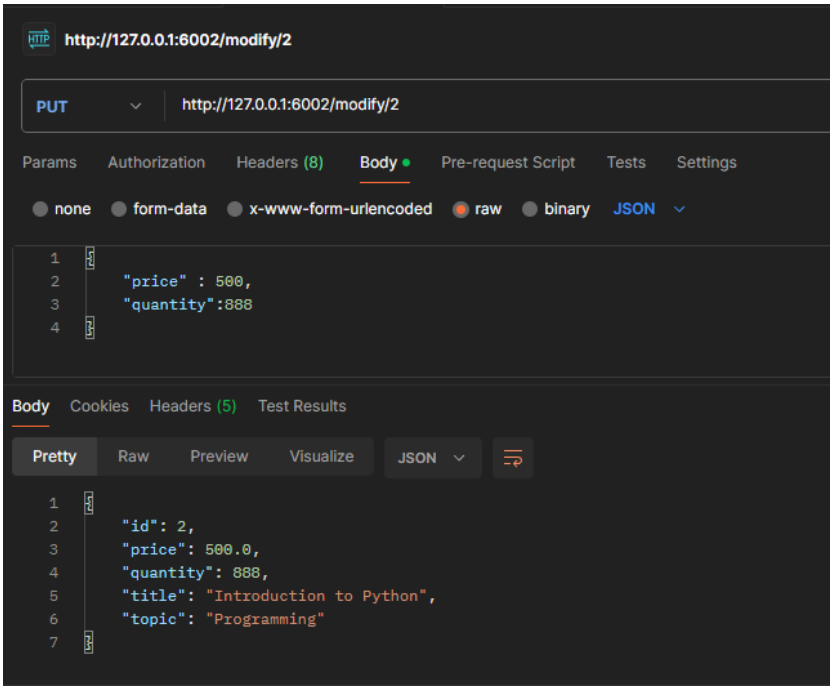
The project contain these file :



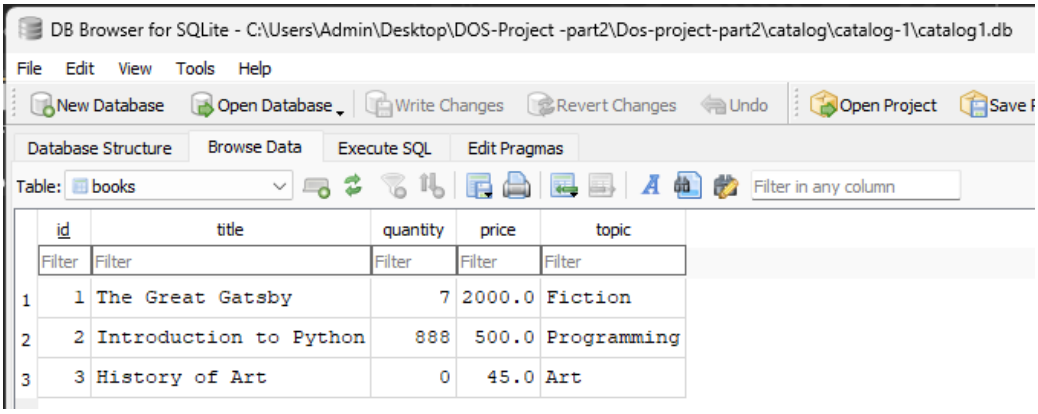
The data base before request Api:

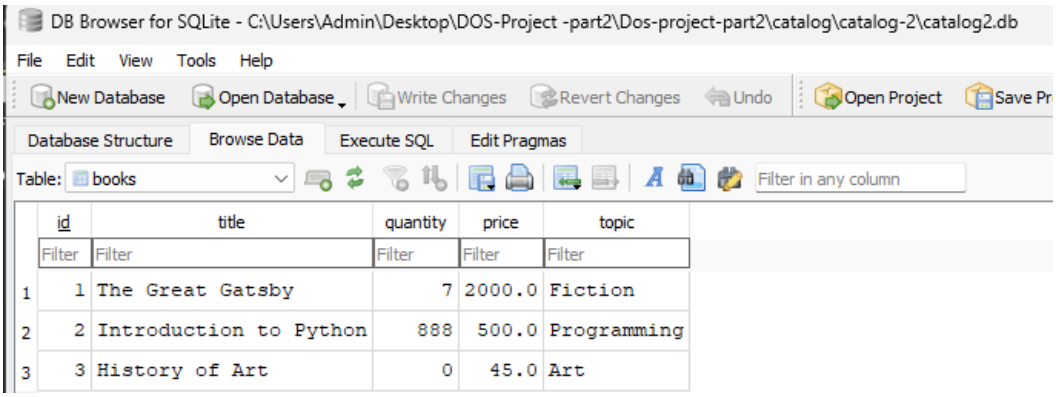


Update book quantity:

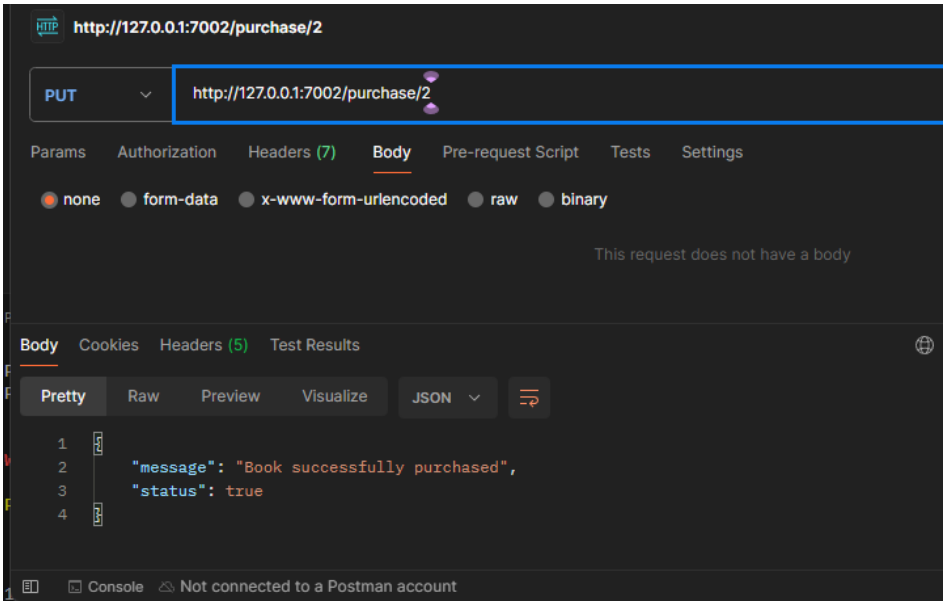


The effect on both database:

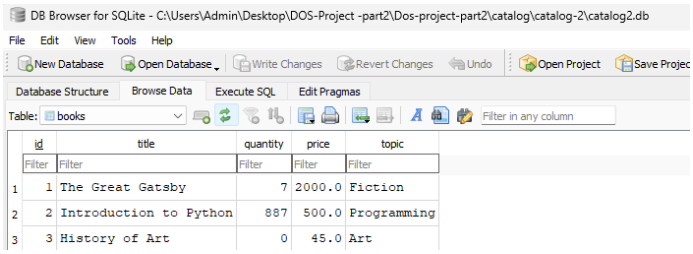


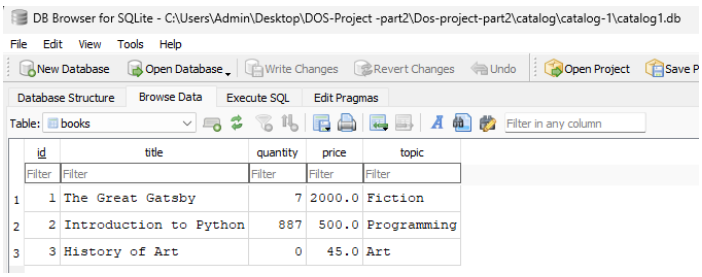


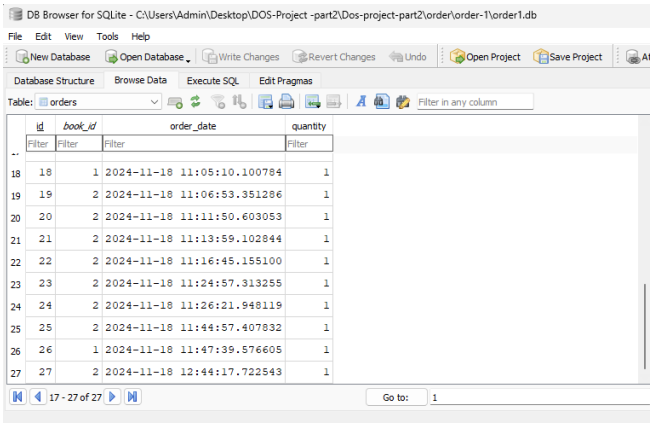
Test api purchase from order server:

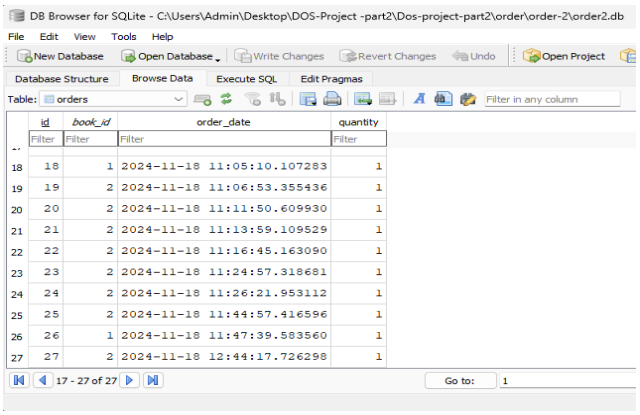


The effect on all database

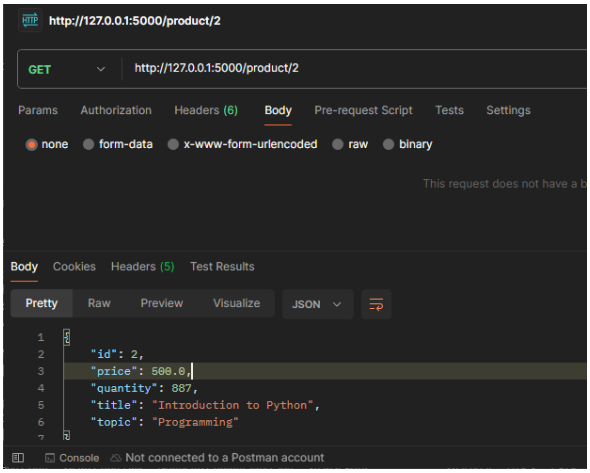








In Front and server code:



First time run api :

```
Using database
the catalog replica: c:\Users\Admin\Desktop\DOS-Project -part2\Dos-project-part2\front-and-server\../catalog/catalog-2/catalog2.db
The time: 0.0020020008087158203
127.0.0.1 - - [18/Nov/2024 12:52:15] "GET /product/2 HTTP/1.1" 200 -
```

Second time run api will get from cache:

```
Using cache
The time: 0.0013659000396728516
127.0.0.1 - - [18/Nov/2024 12:52:48] "GET /product/2 HTTP/1.1" 200 -
```

Explain some point:

Replica server catalog and catalog:

Any request from server it will get data from its database

And any trying to modify on database it will to change and update all other database.

Cache:

The first time get request it will get from database.

The second time run same request it will get from cache with less time than first time.

About Consistency with cache:

I will remove the data from cache then set it again in cache.

This code:

```
cache.delete(f'products_{id}')
productD =dict(product2)
cache.set(f'product_{id}', {"product": productD}, timeout=60)
```

CONCLUSION

In summary, the implemented replication, caching, load balancing, and consistency mechanisms have notably improved system performance and fault tolerance. However, challenges arise during operations requiring cache consistency, leading to increased overhead and latency for updates. Despite these challenges, the system demonstrates enhanced reliability and efficiency in scenarios where the benefits of replication and caching outweigh the complexities introduced.