

# DOS

# Dr Samer Arandi

***Lab2***

Shihnaz Nasser && Salsabeel Daraghmeh

In this lab ,we are going to improve lab1 by adding replication to the servers except the front end server, that receives the read requests(“info,search”) and store them in cache server for the first time then it check in the cache server before it sends request to one of the catalog replica .while the write requests is sent to the catalog and order servers trying to achieve consistency principle among these servers. Extra three books were added to our site , two replica for each order and catalog servers are added , in this lab we achieved the replication ,consistency and load balancing principles

**Replication &** **Consistency:**

We added two replicas for order server and catalog server the replication by making two copies of each of them having two order servers and tow catalog servers except the frontend server that doesn’t replicated we used cache server for the front-end server giving each server different port number.

To ensure that the data in each server is consistent, if any server updates the data in the database it sends REST request to the other servers to update their data and sending invalidate message to the front end server to delete the data stored in cache (if any )

**Load balance:**

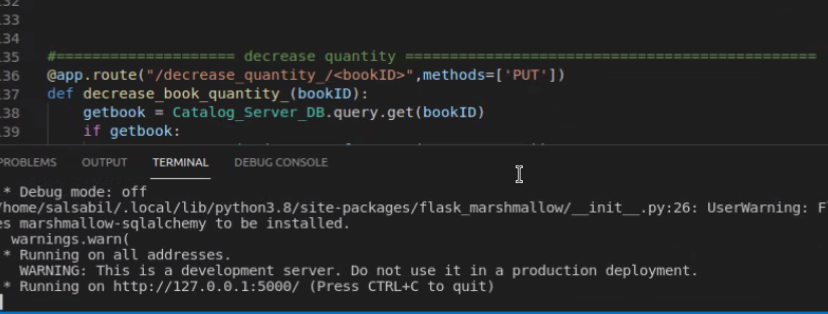
We used round robin algorithm to distribute the requests between replicas , each request is sent by the frontend server to one of the replicas using counter from 0 to 1 if it is 0 send request to the first catalog server then increase the counter by 1 , else if it is 1 then send request to the second catalog server then set the counter to 0 ,with this algorithm we implemented load balancing, the same idea is applied at the order server

**How does our program work?**

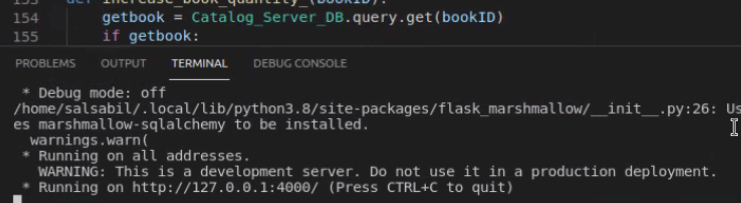
In order to run our program, the first step is to run the replicas and the cache server with different ports individually each one on its machine, then the client can use postman to send request to the front-end server that check the stored request in cache server using REST microservices ,if the requests exists then it used the cache server if not it ,will resend it to the catalog or order servers using load balancing algorithm.

**How to run our program?**

As we said above the first step is to run two catalog servers on the catalog server with two different ports and each server has its own database as shown below :

Catalog Server2 on port 5000:

Catalog Server1 on port 4000:

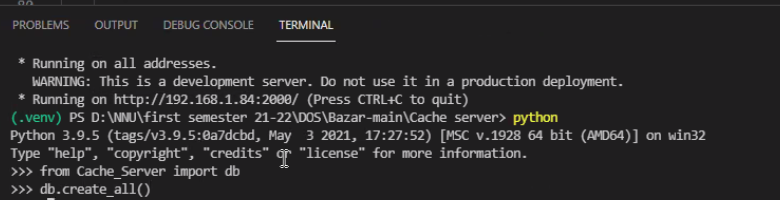
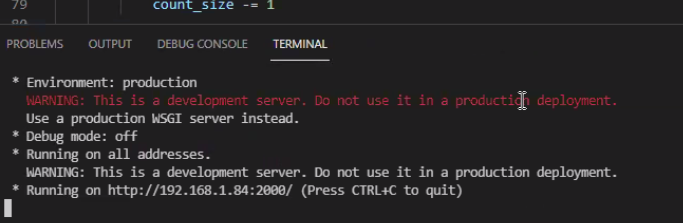


With the same concept we ran three replicas at the order server .the first with port number 5000,the second with port number 4000

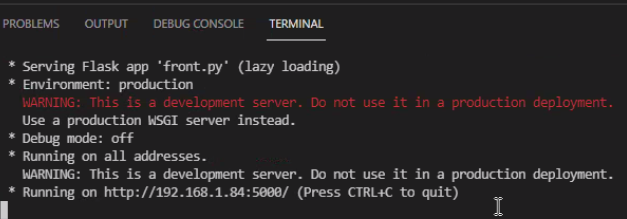
While the cache server with the same IP of the front-end server, and with different port number as shown below:

Cache Server on port 2000:

In the cache server we created the data base that we stored the requests in it



While The front-end Server on port 5000 :

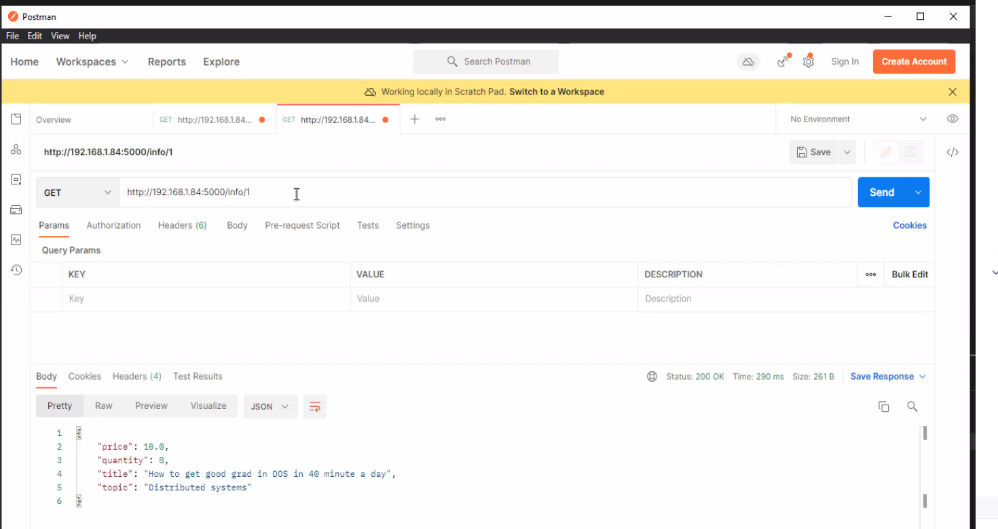


After that we used postman as client platform to do different requests like : info ,search ,update price, updating quantity…etc then the difference was clear between requesting read order for the first time “without cache” then “with cache”. Like in info and search requests

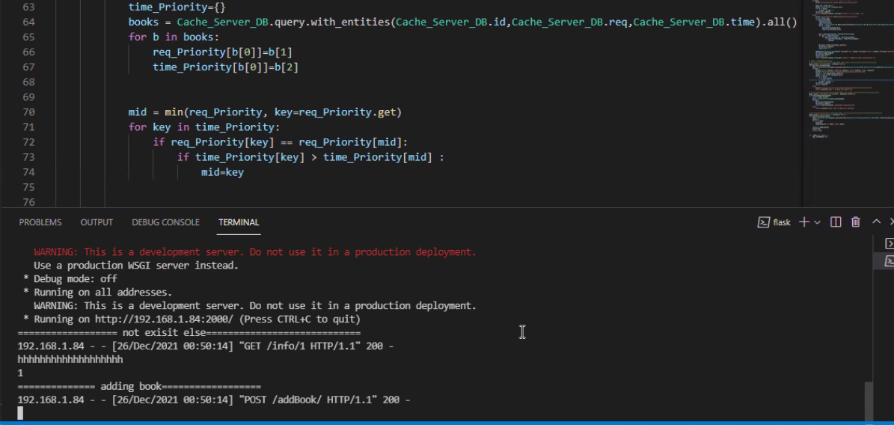
**Info request:**

The first try (without cache ):

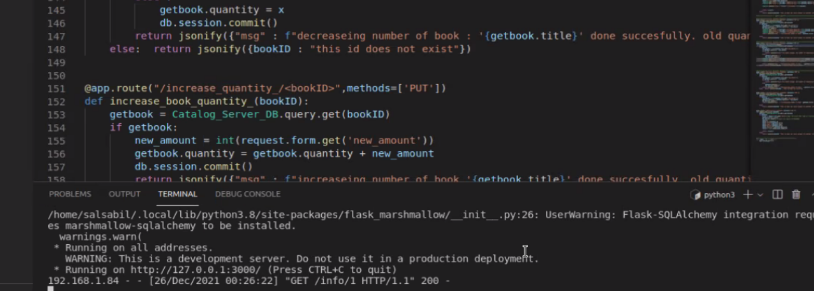
In the beginning the cache was empty, when the request” info/1”the front-end server sent the request to the cache server as its empty , it will resends it to a replica of the catalog server ”the first catalog in this case” using round robin algorithm as shown below:

this figure shows that it needs 290 ms

As this request doesn’t exists in the cache before it will be added to the cache server which has limited size if it full we used LRU algorithm to hold this problem .

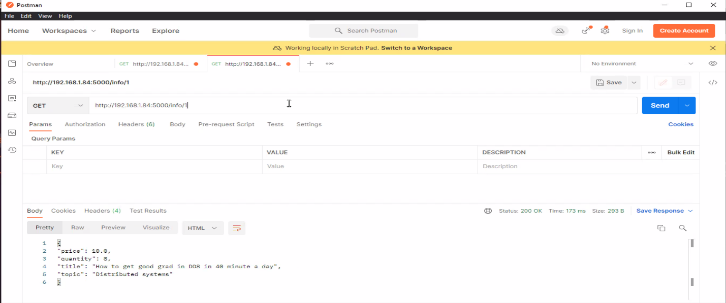


Then the frontend sends request to catalog server “1st” then it response with the results to the front end server as shown in the figure :



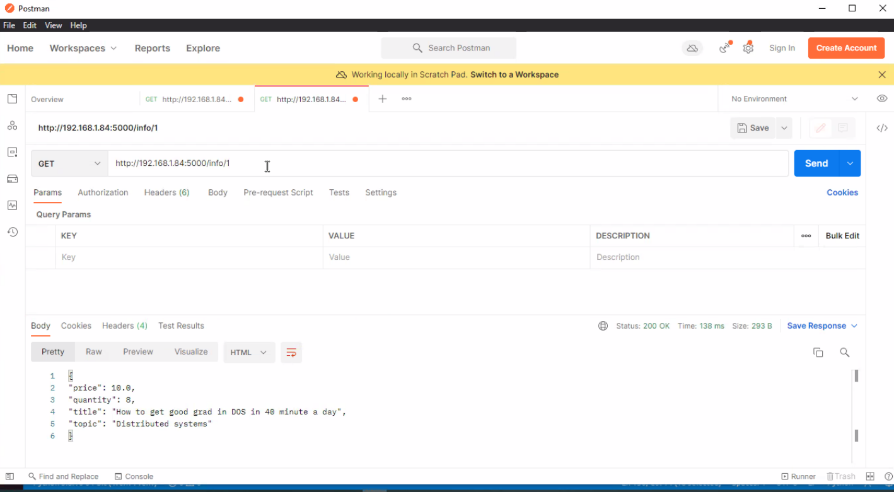
**The Second try( with cache )**  
as the result is stored in the cache after the frontend sent request , it finds that it is already in cache server then it does not send request to the catalog server having better performance .

This figure shows the result With cache the time is 173ms



**The third try with cache :**

This figure shows that the time decreased to 134ms and the front end does not send requests to the catalog server .

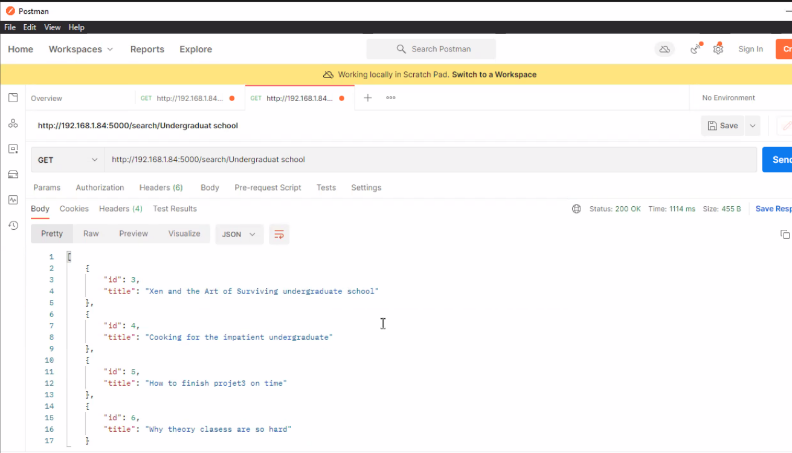


**Search request :**

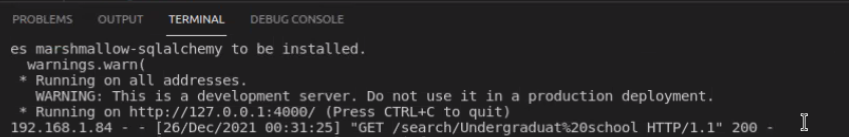
**The first try(without cache ):**

With the same concept, after the client sent request to the frontend server it checks in the cache server if it exists then it will send the result directly to the client, if not then it will re send the request to one of catalog replicas.as shown below:

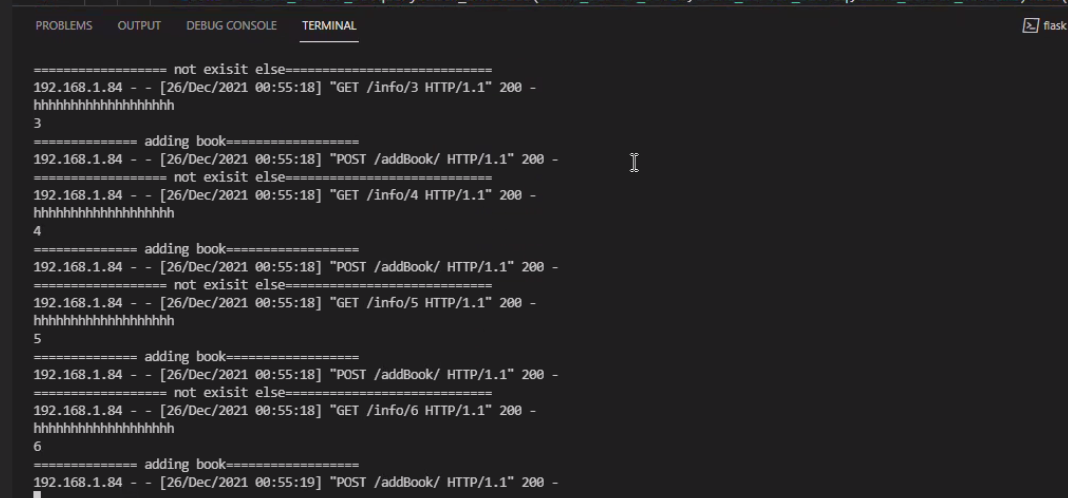
This figure shows that the response time is 1114ms



The front-end server sends request to catalog server using round robin algorithm that it will sends back the result to frontend then to the client

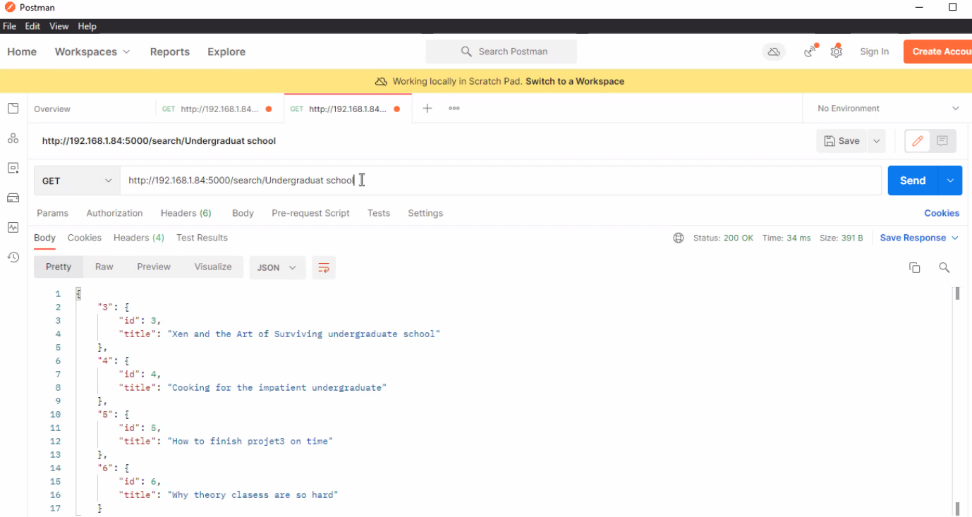


The cache server added the results to its database as shown in the figure below:



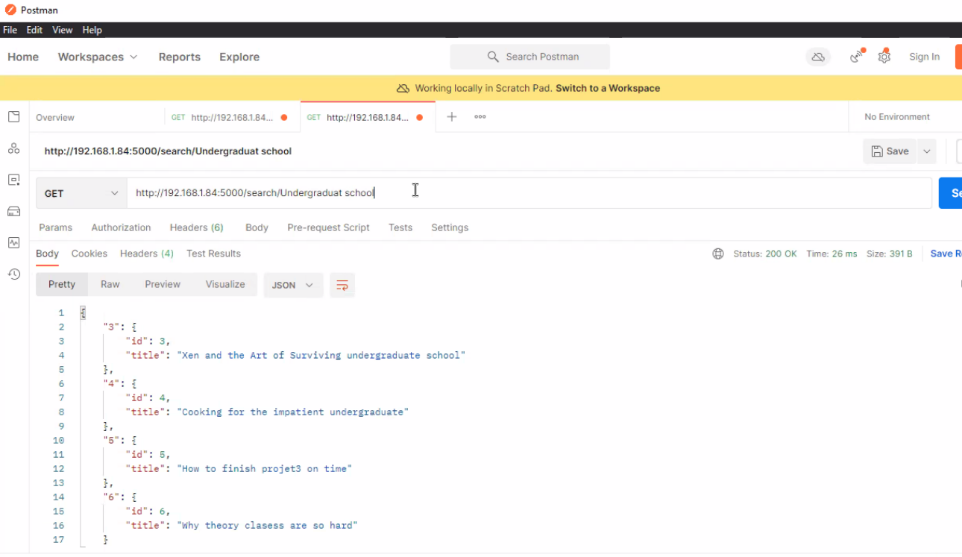
**The second try (with cache) :**

the search with cache has better performance, and that its clear in this figure that shows the time is 34 ms :



**The third try with cache :**

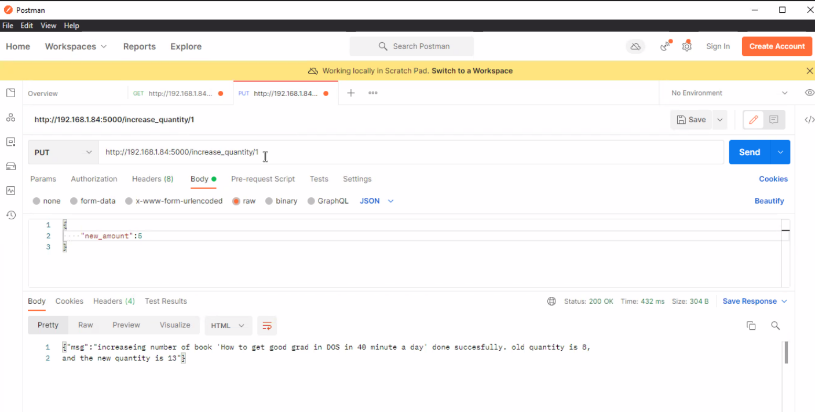
This figure shows how is the performance improves well with caching (time needed is 26ms)

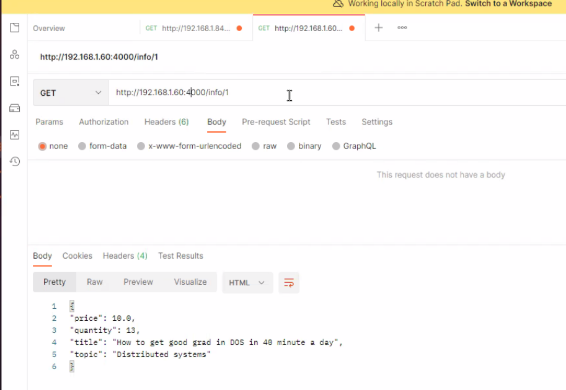
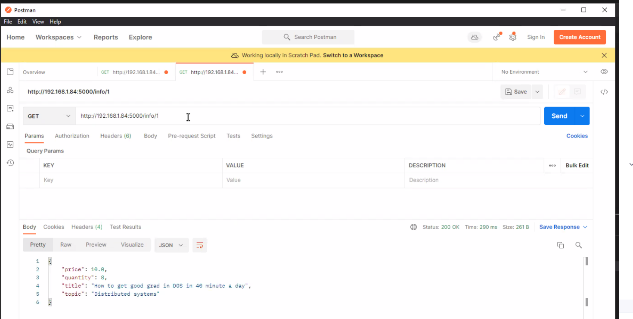


**Updating quantity:**

Increasing quantity:

When we increased the quantity in the catalog database ,using put request ,the same value is updated among the replicas with this concept we achieved consistency, as every replicas sent to the other replica that the value changed ,sending invalidate message to the front-end server that sends to its cache to delete if there is any data ,in this example we increased the quantity by 5



Then we ensured that the value is the same in the two replicas:

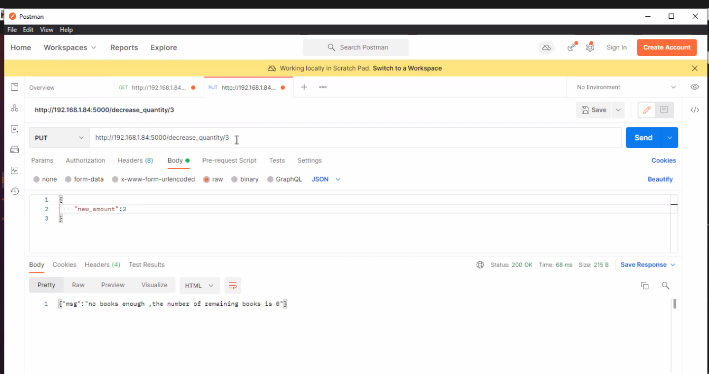
Then we deleted the data of this book from the cache sending invalidate request



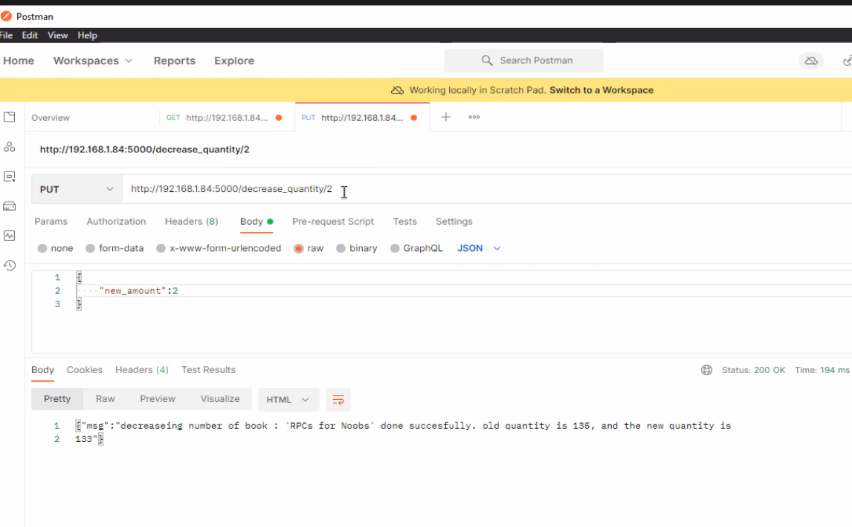
**Decrease Quantity:**

With the same idea, we decreased the quantity of books in the two servers, as shown in figure below:

If the books count is not enough to decrease it sends a message that there is not enough books .

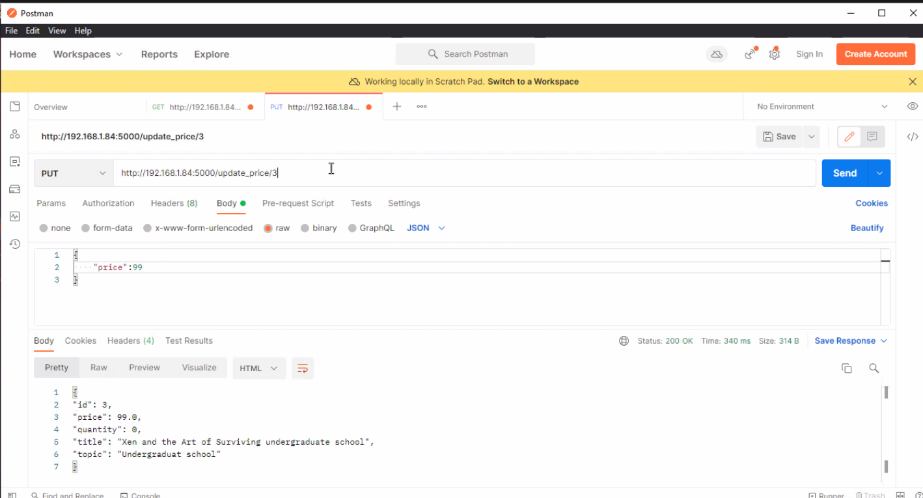


if the number of books is enough, then it decreases the quantity in the catalog replicas, with the same value



**Updating price**

With the same idea, we updated book price with new value at the two servers, as shown in the figure below:



-

|  |  |  |
| --- | --- | --- |
| Info request | Time without cache | Time with cache |
| 1st try | 290ms | 173ms |
| 2nd try | 286ms | 134ms |
| Average Response time | 288ms | 153.5ms |
| Performance | =288/153.5 = 1.8 | |

Note that Cache decreases the latency time, so improving performance

|  |  |  |
| --- | --- | --- |
| Search request | Time without cache | Time with cache |
| 1st try | 1114ms | 34ms |
| 2nd try | 1110ms | 26ms |
| Average Response time | 1112ms | 30 |
| Performance | =1112/30 = 37.06  Enhanced the performance 37 times | |

How can we improve the performance?

We can improve the security of the design using authentication to handle that not anyone making changes in our databases.