**Distributed Operating Systems**

**Lab 2**

**Turning the Bazar into an Amazon: Replication, Caching and Consistency**

Walaa Dweikat

Aisha Marmash

As we submitted in Lab 1, the program is built using flask with sqlite database, it is a web REST API which contains three web micro-serves. Every server resides on different machine and they communicate with each other using the network by sending http requests form a server to another.

But now we add three new books to the Bazar.com, and because of the popularity of the book store we need to rearchitecting it to handle the higher in the workload.

We add replication, caching, load balance and consistency to improve request processing latency.

**Replication:**

We implemented the replication by copying the catalog server to two new servers and also copying the order server to two new servers, so we now have 3 catalogs and 3 orders, but the front end server is not replicated, each one runs in a different port.

**Consistency:**

We implemented consistency by using http rest calls from the server which have a change in its database to the other two replicas.

Cache consistency needs to be addressed whenever a database entry is updated by buy requests or arrival of new stock of books, so we send invalidate request to the in-memory cache, which will cause the data for that item to be removed from the cache.

**Load balance:**

We implement load balance using round robin algorithm by send each request from the front end server to one of the replicas by using 2 counters from 1 to 3 one for catalog server and the other one for order server. According to the counter value 1, 2 or 3 we send the request to a specific server. If catalog counter equal 1 we send the request to the first replica, but if it is 2 we send the request to the second replica and so on.

**Run Bazar.com:**

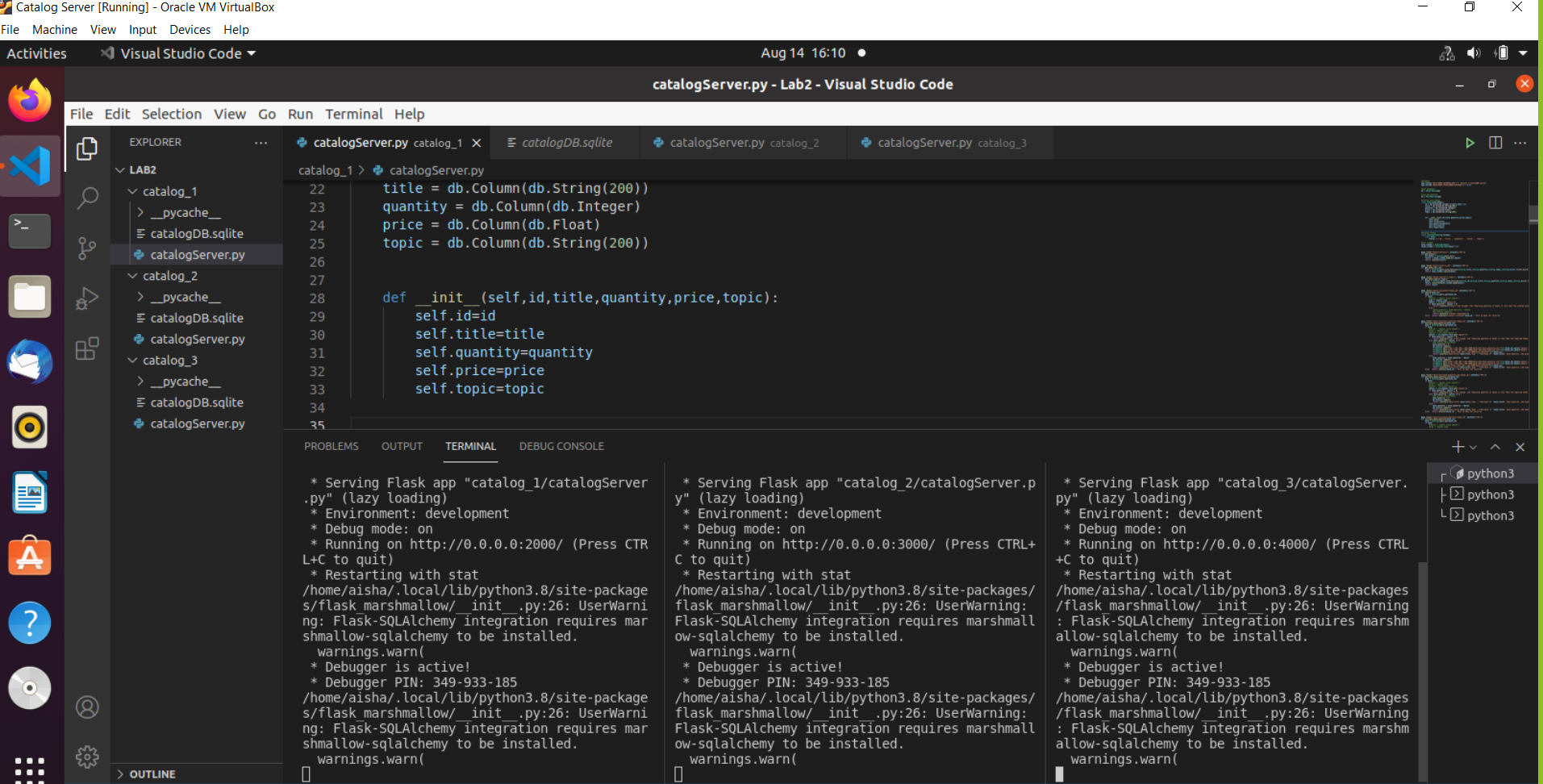
To run our program, we run all the servers each one on its machine on a different port of the other replica port and use the postman program as a client send request to the front end server, which will resend it to the needed server.

**How it is works**

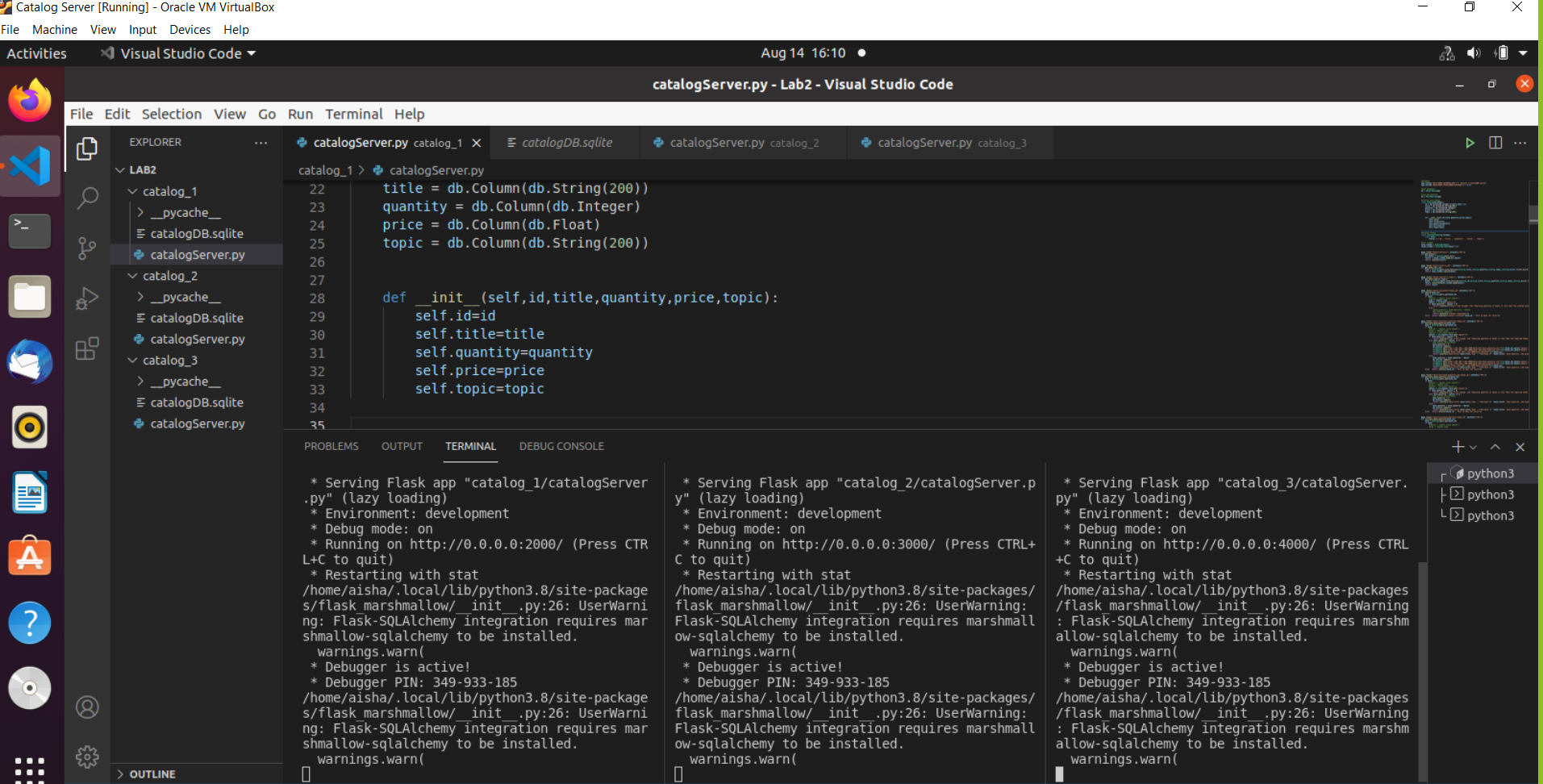
At first we ran 3 catalog servers on catalog server machine, each one on different port

3 replicas if catalog server

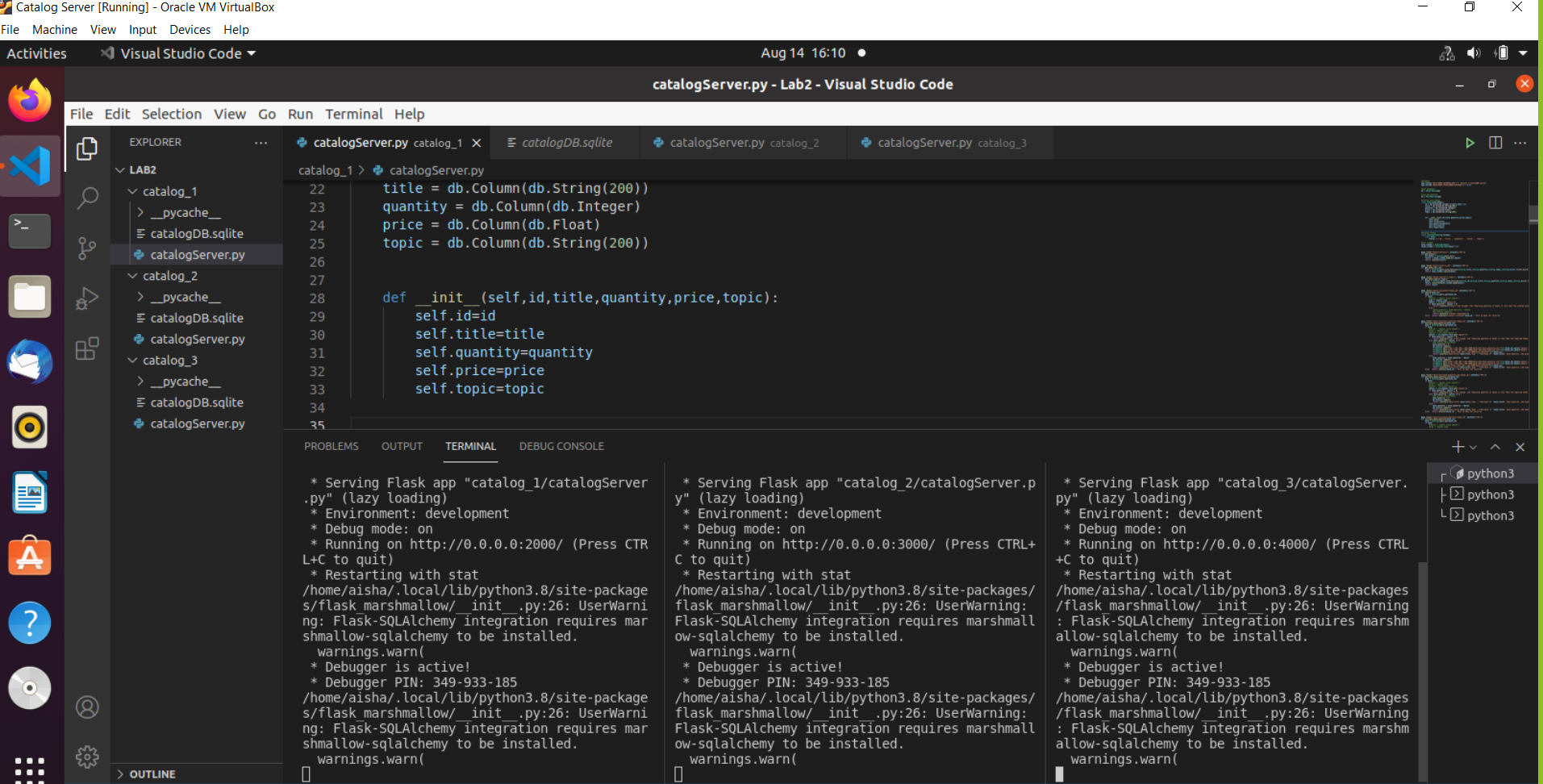
Catalog\_1 on port 2000



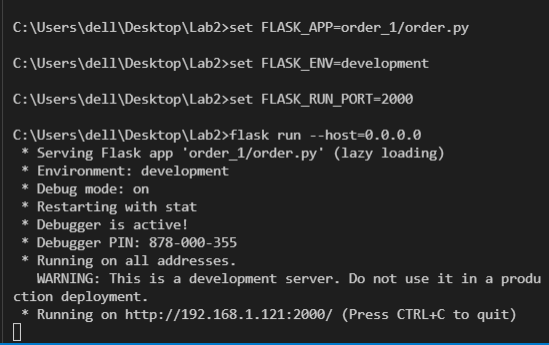
Catalog\_2 on port 3000



Catalog\_3 on port 4000

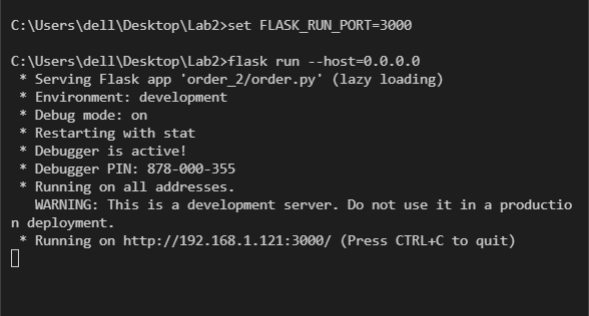


Ran 3 replicas of order server on their machine each one on a different port

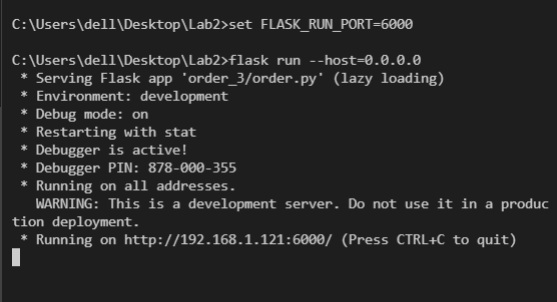
Order\_1 on port 2000

Order\_2 on port 3000

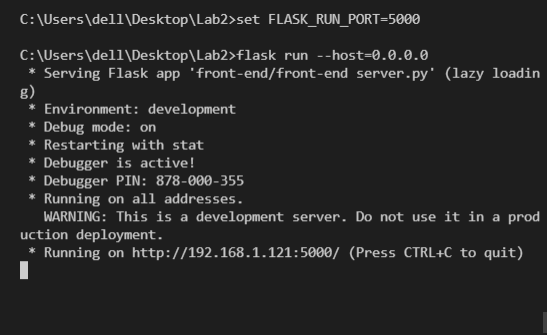
Order\_2 on port 3000



Order\_3 on port 6000

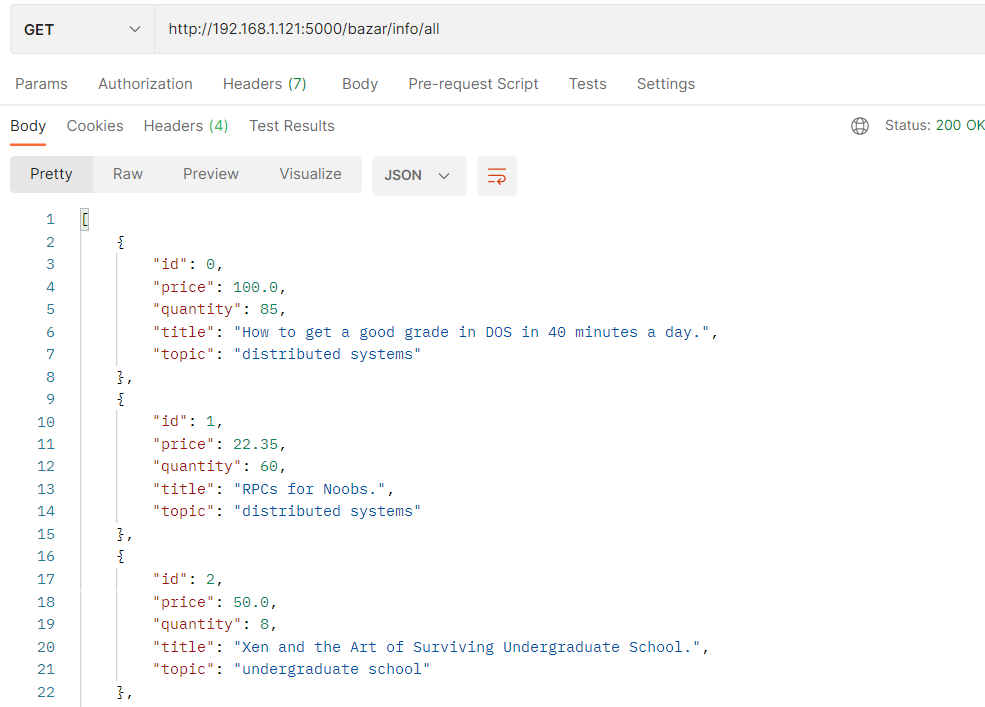


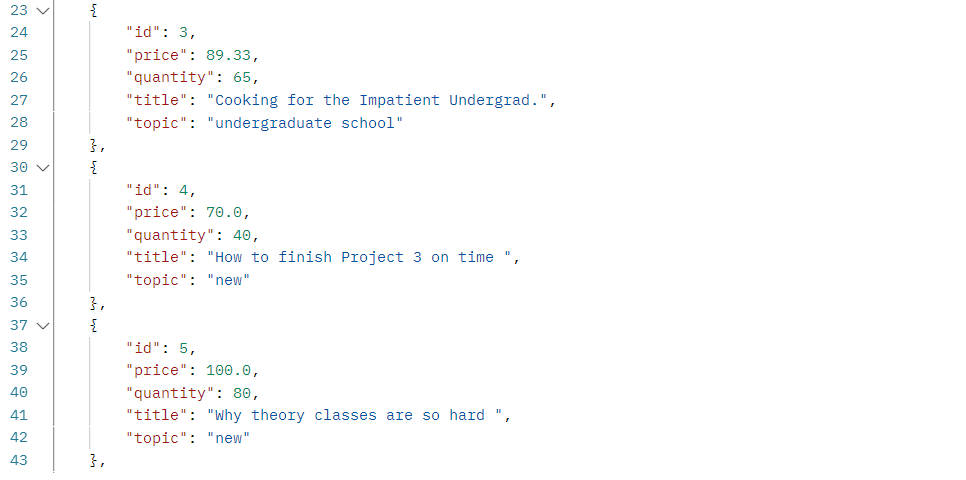
Ran one front end server on port 5000

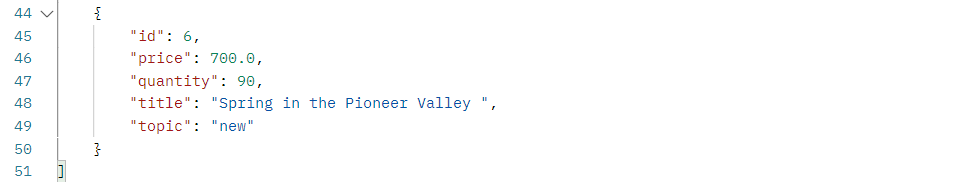


And we use postman as client.

* We add three new books to the store, so now we have 7 books

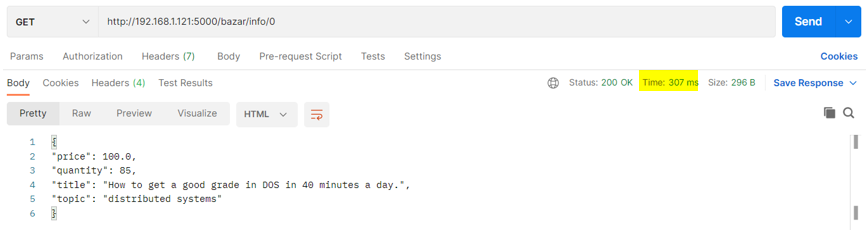




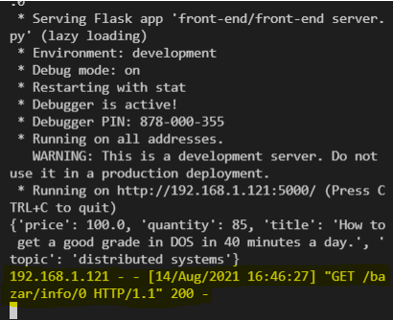


**Cache and load balance:**

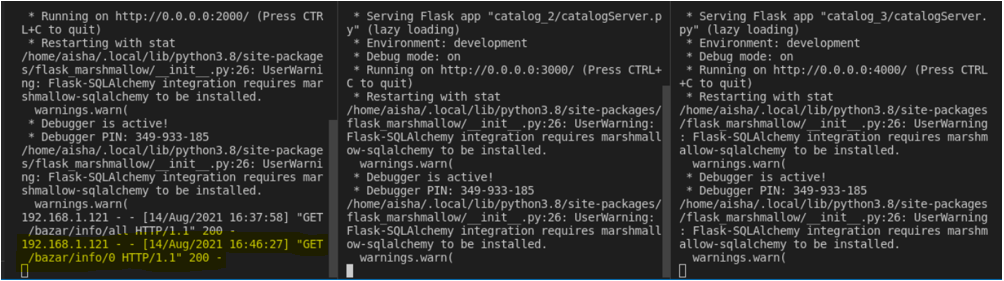
**Trial 1**



At first, we started with empty cache, so when we sent request for “info/0” the front end server resent the request to the one of the catalog replicas because the cache is empty and didn’t have this book information.

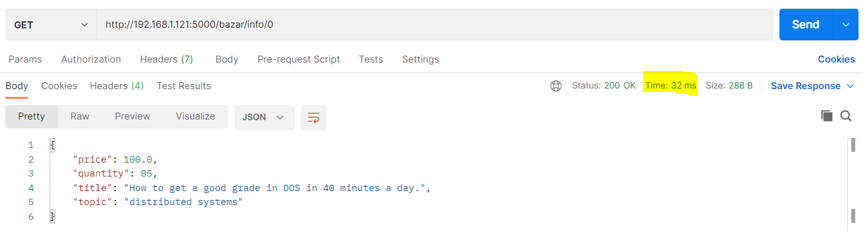
front-end server

the request sent to the catalog\_1 to read the information of the book with id = 0



The response returned from the catalog\_1 to the front end server then to the client and stored a copy of the information on the cache,so when the client sent this request for the first time it needed 307 ms but when he sent it again it spend 32 ms, because the needed information stored in the front end cash, and the front end server returnd it without sending request to the catalog server.

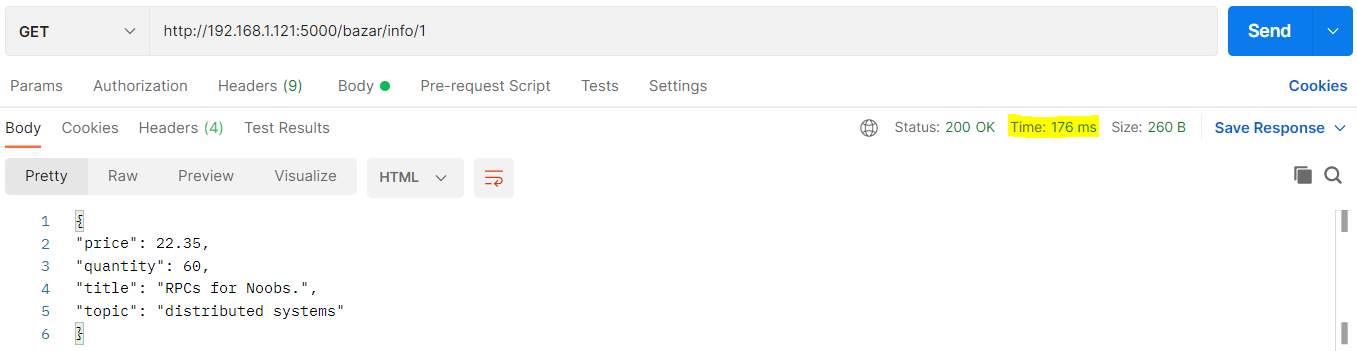
After cashing



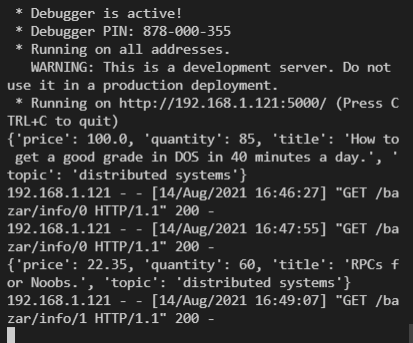
**Trial 2**

As prevouse trial when we asked for information not found in the cache, front end server brought it from the catalog server, but here we noticed load balance, where the front end server sent the request to the second replica of the catalog server, stored its response in the cache for the next requests.

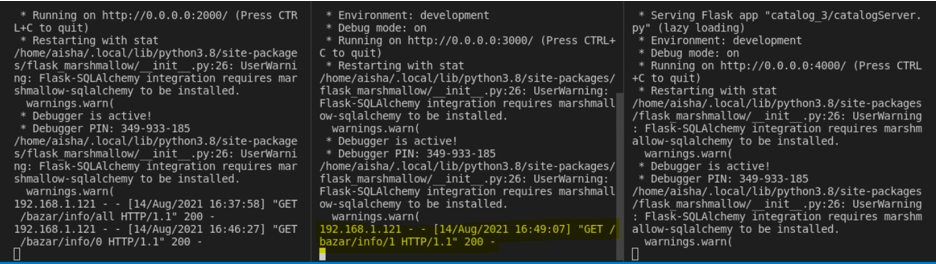
Without cache



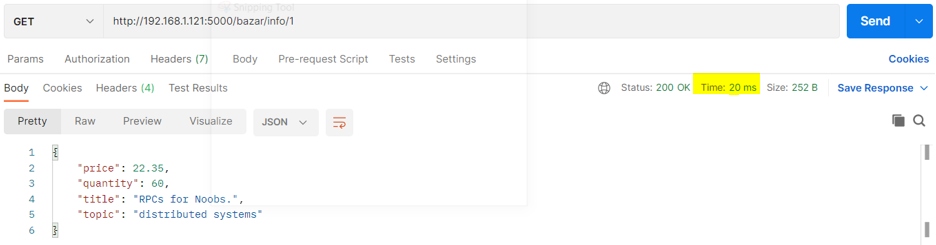
Front end server sent request to catalog\_2



The response brought from catalog\_2 (load balance) and stored in the cache memory

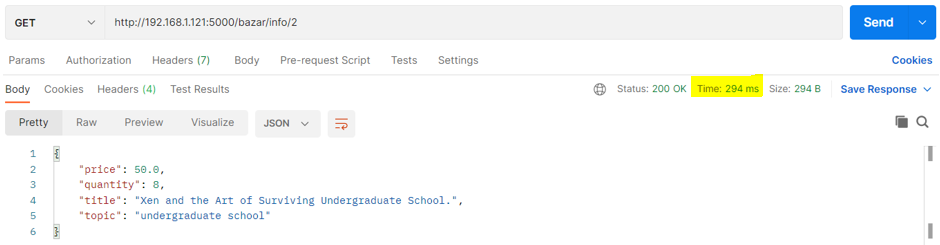


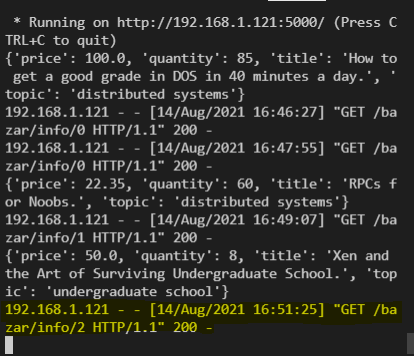
With cache



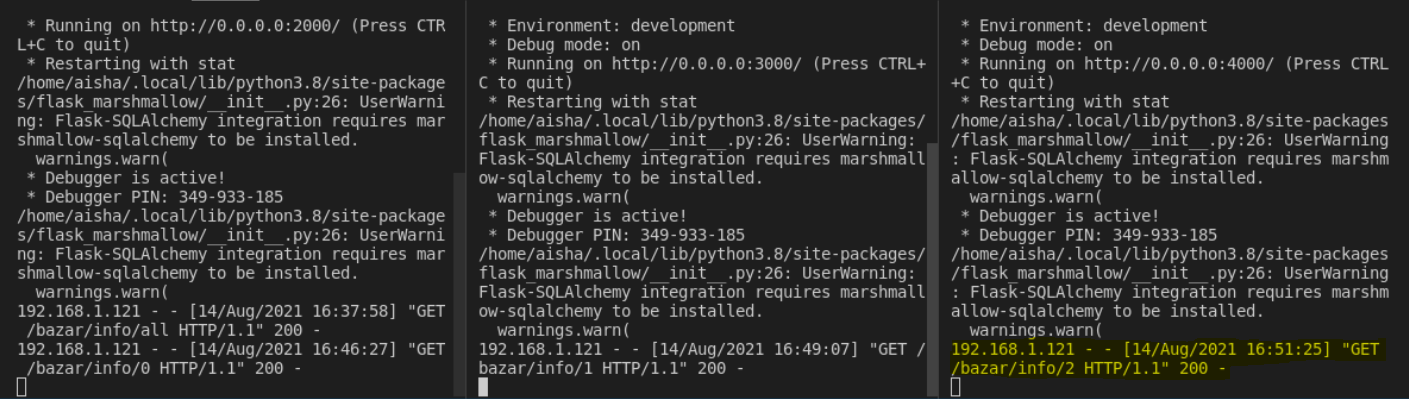
**Trial 3**

Here the request sent from client to the front end server, then to catalog\_3(load balance). Response returned from the catalog to the front end server, stored in the cache and returned the response to the client, and the time we needed was 294 ms.

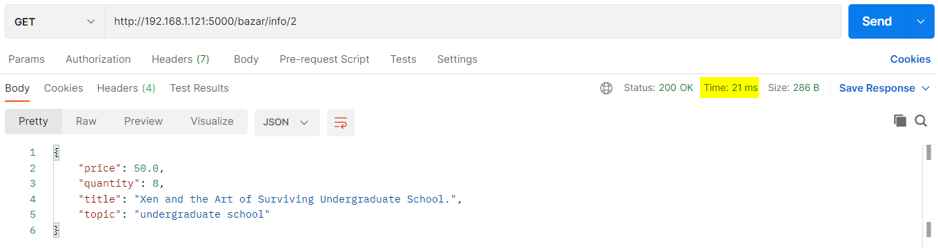


 Front end server

Sent to catalog\_3



Second request got the response from the cache with 21 ms.

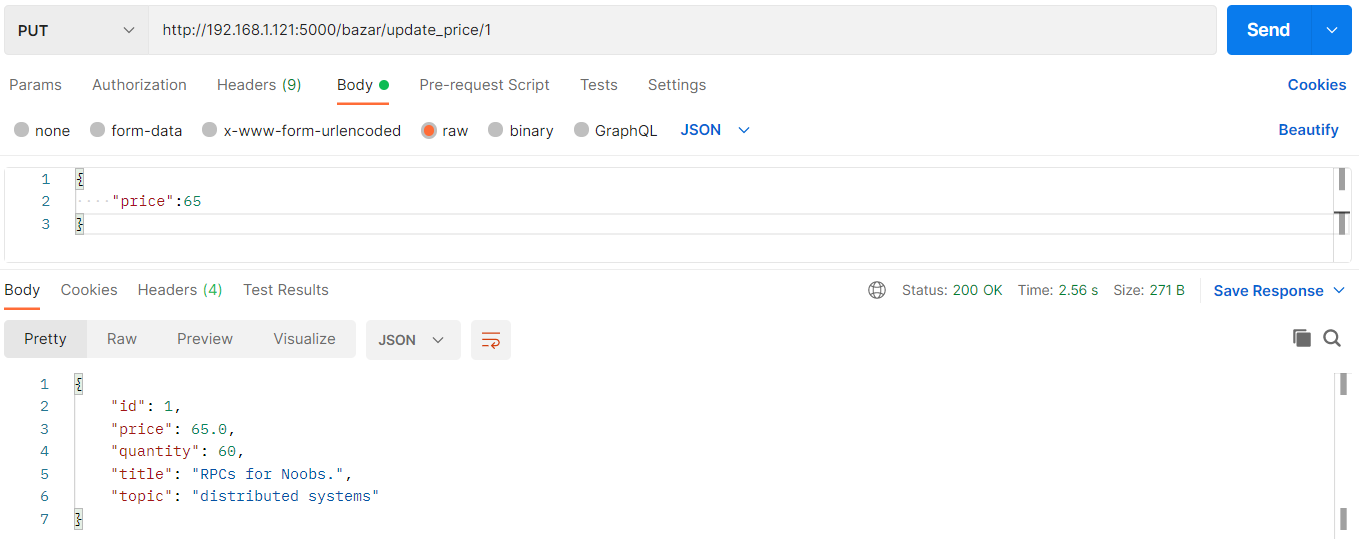


The cache has limited size, less than the catalog database size. So when the cache is full and we need to store new entry inside it we use LRU algorithm to replace it with one of the entries inside the cache.

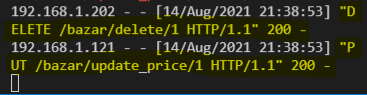
**Consistency**

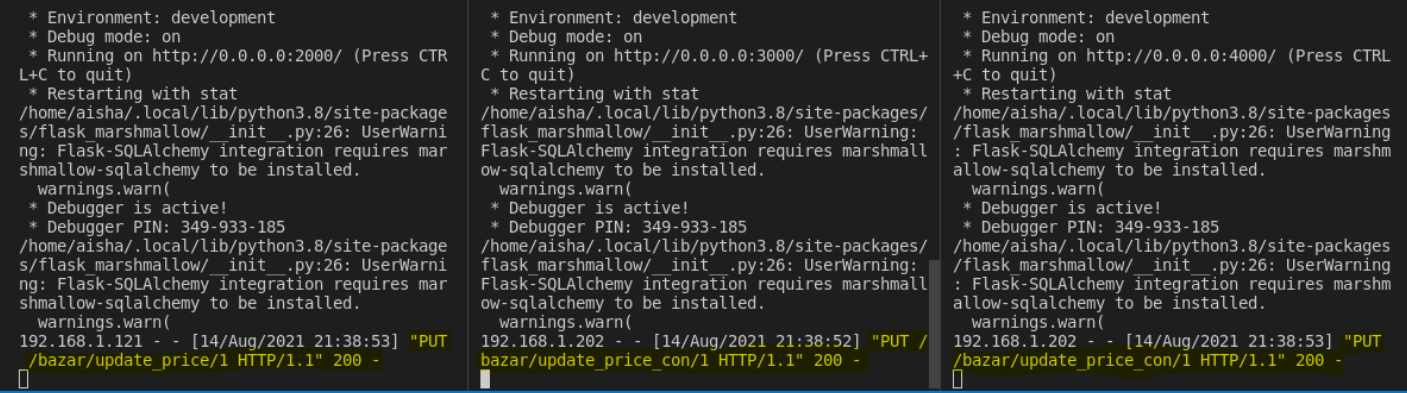
When we update information in the catalog database with “PUT” or “POST” url calls, the same update sent to the other replicas by using http rest calls, and we send invalidate request to the cache memory, which will cause to delete the information about the changed book.

Here we sent request to update price of book1 (write operation)



So we delete the entry of this book from the cache by sending invalidate request from the catalog to the cache



The request went to catalog\_1, but then requests sent from it to the other replicas to update the same information on their databases.

Here if we got the book 1 information from any catalog server, it will be the same.

And the same if we did buy operation on order server, it will store the new order in the all replicas databases, and update book quantity on all the catalog replicas.

- Compute the average response time (query/buy) of your new systems. What is the response time with and without caching? How much does caching help?

|  |  |  |
| --- | --- | --- |
|  | Time without cache | Time with cache |
| Trial 1 | 307ms | 32ms |
| Trial 2 | 176ms | 20ms |
| Trial 3 | 294ms | 21ms |

average response time without cache = (307+176+294)/3

= 777/3

= 259ms

average response time with cache = (32+20+21)/3

= 73/3

= 24ms

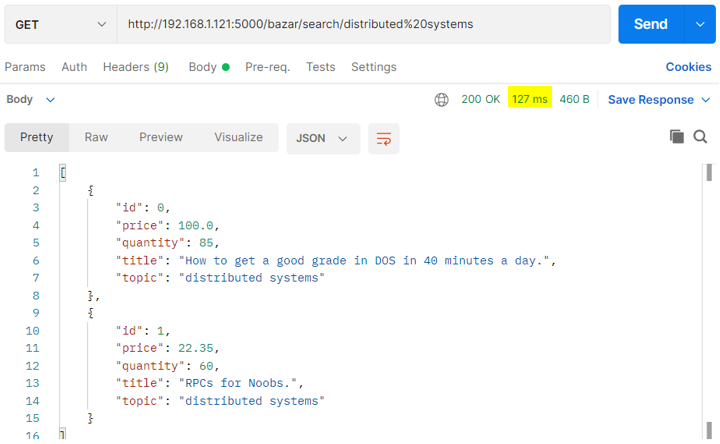
Performance :259/24.33 = 10.6 so the new performance 10.6\* old performance

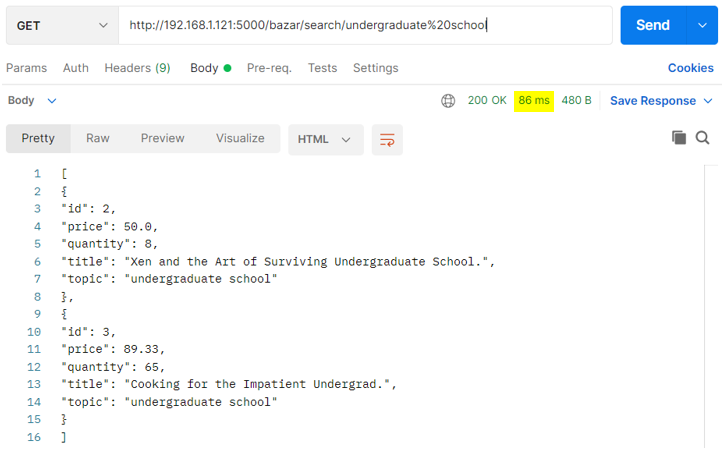
Cache increase decrease the latency time so we got the data faster.

**Search operation**

Also we use cache for search operation (read operation). For example, when I searched on a topic as a client and I got a response, returned the books from the catalog, and the books stored on the cache, so when I need to get information about one of them I got it from the cache memory with low latency. When the cache is full we will replace the new books with the least ordered book in the cache.

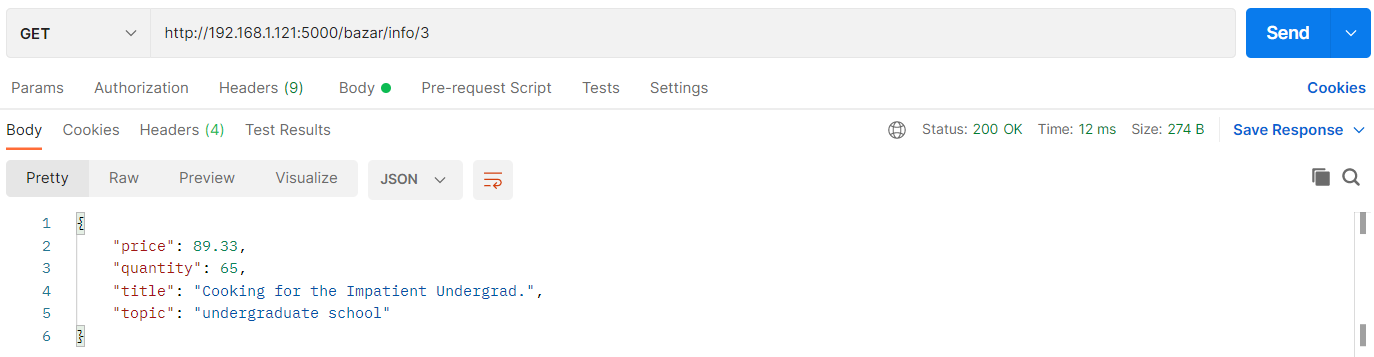
Starting with empty cache, we searched on “distributed systems” books, we got it from the catalog database and stored it in the cache database, it spent 127ms.

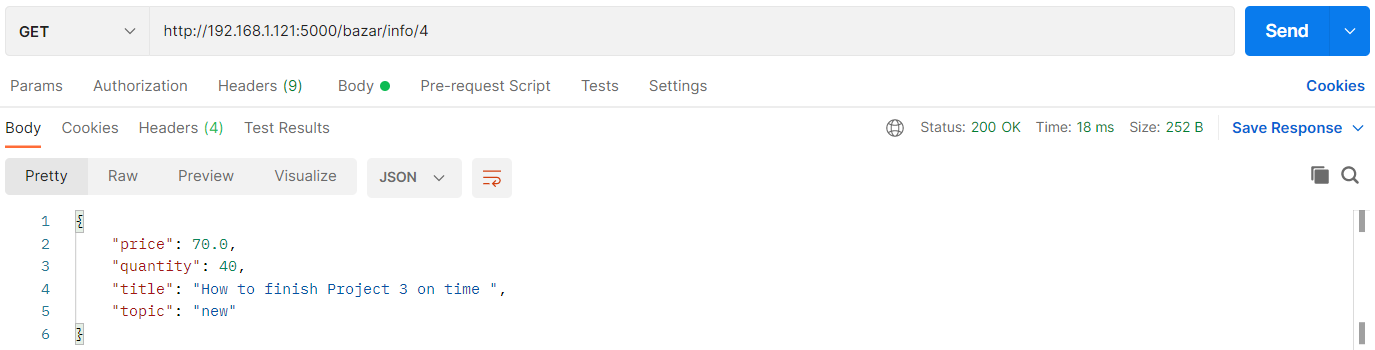


And we searched on “undergraduate school”

But the cache size is 5, and when we sent search requests of 2 different topics, cache now has 4 entries so there is a space for 1 book now and if I want to search of topic 3 “new” there are 3 new books come to the cache, so we replace the three new books with the least ordered ones.

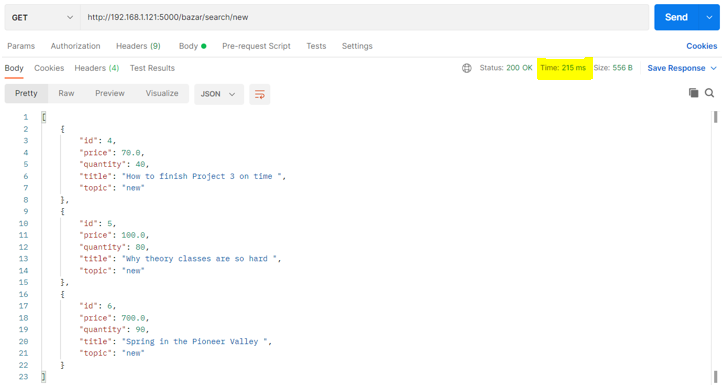
So if I do requests for books 3 and 4 again their will ordered more than others on the cache.





Notice the low latency, because we got this information from the cache.

Books 0, 1 and 2 are least ordered so when I search of “new” topic book it will replace them with 0, 1 and 2 books on the cache.



So now if we need book 0, 1 and 2 information the front end server will bring it for the client from the catalog server with low latency not from the cache.

* Construct a simple experiment that issues orders or catalog updates (i.e., database writes) to invalidate the cache and maintain cache consistency.

In the previous example of consistency, the time we needed was 2.56s to finish the update price operation, because we need to send delete request to the cache and update requests to the other catalog replicas, so it will spend a lot of time.

* What are the overhead of cache consistency operations?

Buy or update operation took long time, because it needs cache consistency.

* What is the latency of a subsequent request that sees a cache miss?

The latency increased because of cache miss, for example when we got the data from the cache it spent 20ms, but when there is a cache miss it spent 176ms to finish the operation.