

*Distributed and Operating Systems*

*Dr. Samer Arandi*

*A Multi-tier Online Book Store*  
*(Replication, Caching and Consistency)*

*Leen Abu Shqair*

### Code Design:

In lab1 I have made three python files, for each server (catalog, order, front-end)

#### 1.Catalog server

I had the catalog server which handles the requests from the front end and the order servers and contains these functionalities in the first homework:

- Adding a book
- Deleting a book
- Updating a book by its id number
- Getting all books
- Getting a book by its id number(/lookup/id)
- Getting a book by its topic (/search/topic)

In the second homework I added another server which is the Replicated server of the catalog server

I added a function to update the catalog replication and updated the changes in (updating the book by its id)

- Update replication

#### 2.Order server

This server is to handle the buy requests from the front end server, which has buy functionality (/buy/id), In the second homework I added another server which is the Replicated server of the order server

#### 3.Front-end server

For the front-end server, it has most of the functionalities of the previous servers (search, lookup), and works as a connection between them, and the user deals with it without knowing about the order+catalog server. In the second homework I added a small cache in it using flask libraries (caching and load balancing)

By adding new servers for the replication the code tries to have consistency in it, if one database is updated the replicated one should be updated too.

By adding a small cache and studying the response time, the code aimed to apply the caching method.

### Code design:

Here is the basic database that we had. I used sqlite.

	id	quantity	cost	topic	name
	Filter	Filter	Filter	Filter	Filter
1	1	0	200	Dos	How to get a good grade in D...
2	2	5	200	Dos	RPCs for Dummies
3	3	5	200	Graduate school	Xen and the Art of Surviving ...
4	4	5	200	Graduate school	Cooking for the Impatient ...

The catalog server is running on the local host with this port number

```
* Running on http://localhost:5001/ (Press CTRL+C to quit)
```

Getting all the books from there:

GET
http://localhost:5001/books
Send

Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 519 ms
Size: 552 B
Save Response

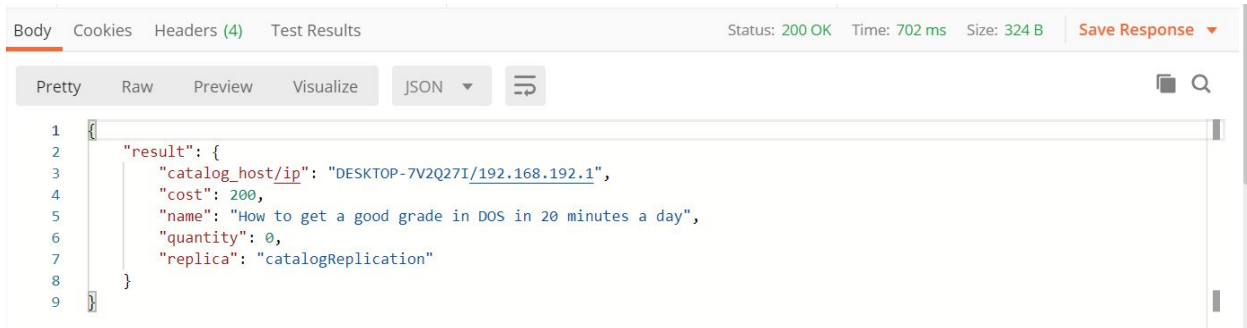
Pretty
Raw
Preview
Visualize
JSON

```

1  [
2    {
3      "cost": 200,
4      "id": 1,
5      "name": "How to get a good grade in DOS in 20 minutes a day",
6      "quantity": 0,
7      "topic": "Dos"
8    },
9    {
10     "cost": 200,
11     "id": 2,
12     "name": "RPCs for Dummies",
13     "quantity": 5,
14     "topic": "Dos"
15   },
16   {
17     "cost": 200,
18     "id": 3,
19     "name": "Xen and the Art of Surviving Graduate School",
20     "quantity": 5,
21     "topic": "Graduate school"
22   },
23 ]

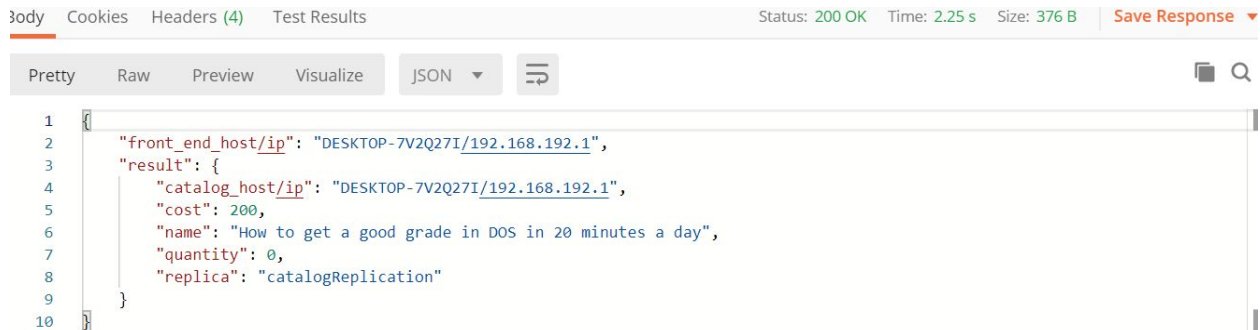
```

## lookup/id from the catalog server



```
Body Cookies Headers (4) Test Results Status: 200 OK Time: 702 ms Size: 324 B Save Response
Pretty Raw Preview Visualize JSON
1 {
2   "result": {
3     "catalog_host/ip": "DESKTOP-7V2Q27I/192.168.192.1",
4     "cost": 200,
5     "name": "How to get a good grade in DOS in 20 minutes a day",
6     "quantity": 0,
7     "replica": "catalogReplication"
8   }
9 }
```

## lookupp/id from the front-end server



```
Body Cookies Headers (4) Test Results Status: 200 OK Time: 2.25 s Size: 376 B Save Response
Pretty Raw Preview Visualize JSON
1 {
2   "front_end_host/ip": "DESKTOP-7V2Q27I/192.168.192.1",
3   "result": {
4     "catalog_host/ip": "DESKTOP-7V2Q27I/192.168.192.1",
5     "cost": 200,
6     "name": "How to get a good grade in DOS in 20 minutes a day",
7     "quantity": 0,
8     "replica": "catalogReplication"
9   }
10 }
```

And so on..

### Tradeoffs:

For the caching method, I ended up using Flask-Caching for in-memory caching. This doesn't allow us to specify a specific cache eviction algorithm. It also adds overhead to the system but on the long term requests it will give better and faster responses.

### Performance Calculations:

I updated the previous code of lab1, to measure some performance and time, I recorded the start time for each new generated request. Then after the request is received I recorded the time stamp and calculated the difference in milliseconds.

### Performance on some test cases:

I applied about 10 requests and had these results

	<u>Avg response time hw1</u>	<u>Avg response time hw2</u>
<u>lookup</u>	<u>64.4</u>	<u>42.2</u>
<u>search</u>	<u>58.73</u>	<u>50.61</u>
<u>buy</u>	<u>146.24</u>	<u>143.54</u>

Obviously, the second homework should have better performance because of the cache, but on a small number of requests it sometimes makes overhead on the system.

### Caching:

The first time requesting it gives around 158 ms, but after this, the cache works and gives around 10ms

Also when there is a cache miss there is latency around 10ms for every request.