



Bazar Multi-tier Online Bookstore Part -1-
Distributed Operating Systems
10636456

1st. Nov. 2022

Masa koni 11820155

Lama Darawsheh 11819574

Program Design:

We have created three servers: catalog server, order server and frontend server. Each of them was running as a separate service.

The frontend server was running on the original pc on port 5000, the catalog server was running on a virtual machine with ubuntu OS on port 3000, and the order server was running on the virtual machine with ubuntu OS on port 3001. We originally wanted to put order server on another instance in virtualbox but our laptops could not function with 3 machines.

The **frontend server**, is the server that the client communicates with and sends the requests through it to either order or catalog server.

The **catalog server**, is the only server that is allowed to read/modify on the database, it performs three main operations:

1. Search for book in a specific category by the topic.
2. Get the information for a certain book by the id.
3. Edit the stock or the price of a book by the id.

The **order server**, handles purchase operations. It sends a query to catalog server searching for the availability of a book in the database, then sends an update to the catalog with the new quantity.

technologies: Node.js & express, mongoDB, Virtual box with Ubuntu running on it.

On MongoDB we made a database with a collection of: `_id`, `itemNumber`, `name`, `cost`, `topic` and `numberOfItems`.

How it works:

- 1- The client only talks to the frontend server, and sends requests to localhost on port 5000 using a browser or postman.
<http://localhost:5000/>
- 2- The frontend server redirects the request to one of the two servers depending on the request type.
- 3- If a search, information or update request is sent, the frontend server redirects it to Catalog server, which is running on the VM with IP address: 192.168.56.101:3000.
- 4- If a purchase request is sent, the frontend server redirects it to Order server, which is running on the VM with IP address: 192.168.56.101:3001.
Then sends a query to catalog server to check the availability of an item, if not then the operation isn't valid, otherwise, the order server sends an update to the catalog server to decrement the quantity of that item.

Here is a list of the available requests and their responses:

Get information by id: Request: localhost:5000/books/info/[id]

- Requesting information of book 2

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:5000/books/info/2
- Status:** 200 OK
- Time:** 90 ms
- Size:** 361 B
- Body (JSON):**

```
{  "id": "636044663c08ae8e513fab1d",  "name": "RPCs for Noobs",  "num": 2,  "numberinstock": 15,  "cost": 10,  "topic": "distributed Systems"}
```

- Requesting information of book 88 which is not in the library.

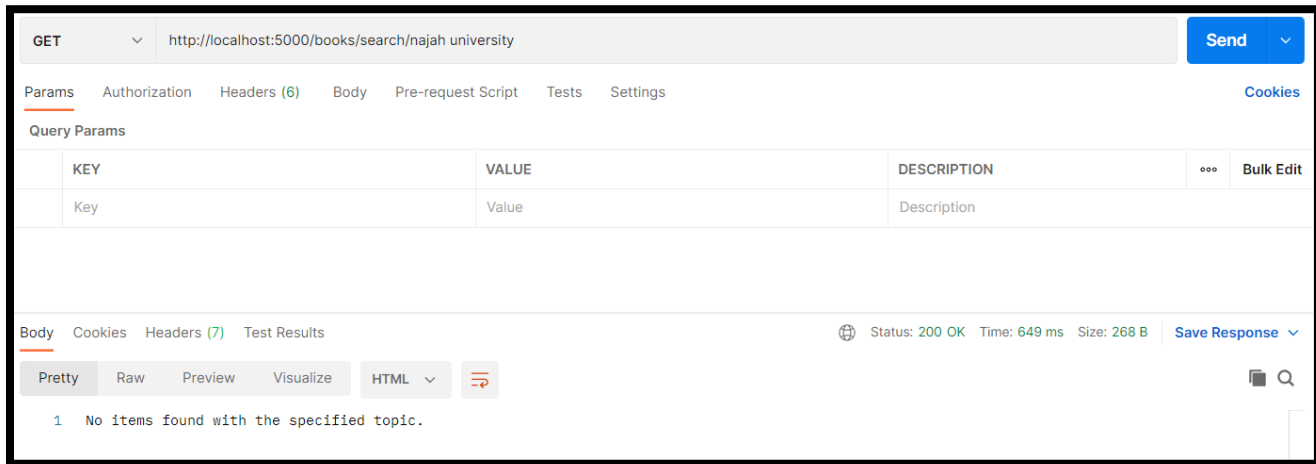
The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:5000/books/info/88
- Status:** 200 OK
- Time:** 138 ms
- Size:** 242 B
- Body (HTML):**

```
1 Invalid Item Id
```

Search by topic: Request: localhost:5000/books/search/[topic]

- Searching with a non-existent topic



GET ▼ http://localhost:5000/books/search/najah university Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

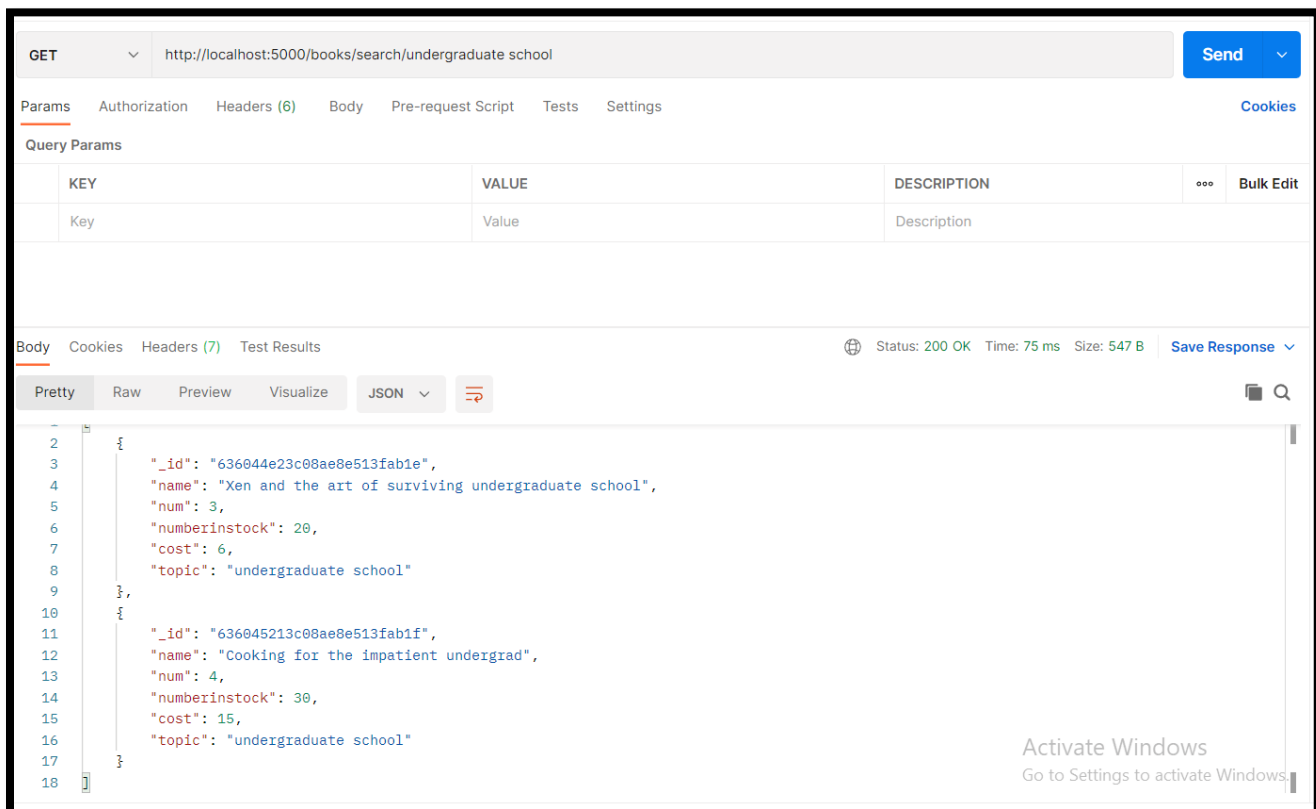
KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results 🌐 Status: 200 OK Time: 649 ms Size: 268 B Save Response ▼

Pretty Raw Preview Visualize HTML ▼ 🔍

1 No items found with the specified topic.

- Searching for a book under title undergraduate school



GET ▼ http://localhost:5000/books/search/undergraduate school Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results 🌐 Status: 200 OK Time: 75 ms Size: 547 B Save Response ▼

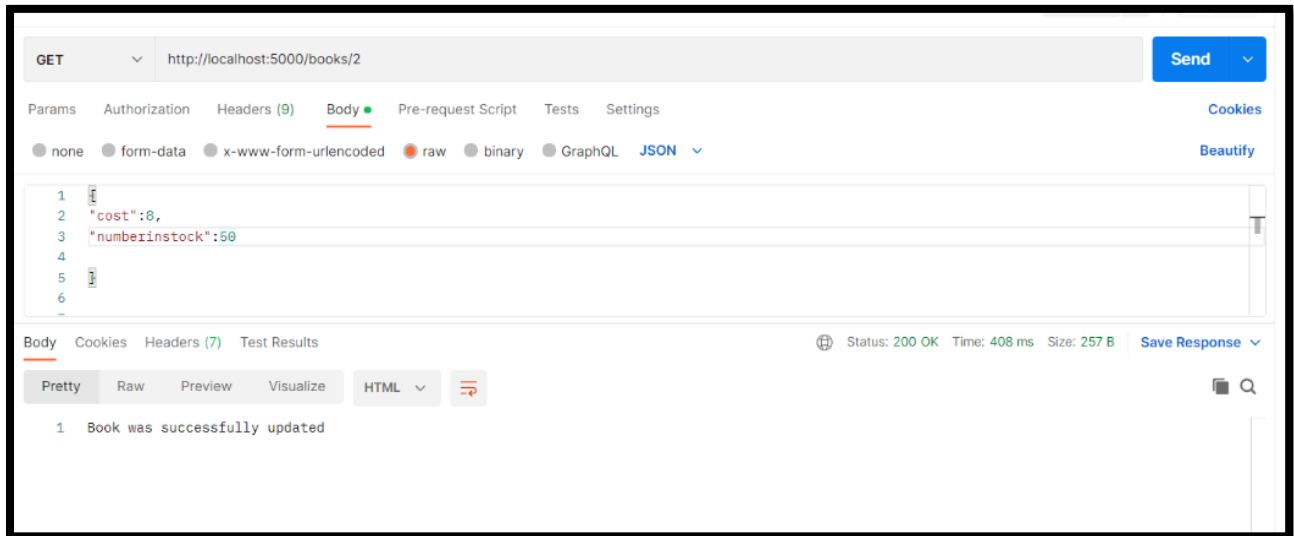
Pretty Raw Preview Visualize JSON ▼ 🔍

```
2 {
3   "_id": "636044e23c08ae8e513fab1e",
4   "name": "Xen and the art of surviving undergraduate school",
5   "num": 3,
6   "numberinstock": 20,
7   "cost": 6,
8   "topic": "undergraduate school"
9 },
10 {
11   "_id": "636045213c08ae8e513fab1f",
12   "name": "Cooking for the impatient undergrad",
13   "num": 4,
14   "numberinstock": 30,
15   "cost": 15,
16   "topic": "undergraduate school"
17 }
18 }
```

Activate Windows
Go to Settings to activate Windows.

Update cost: Request: localhost:5000/books/[id]

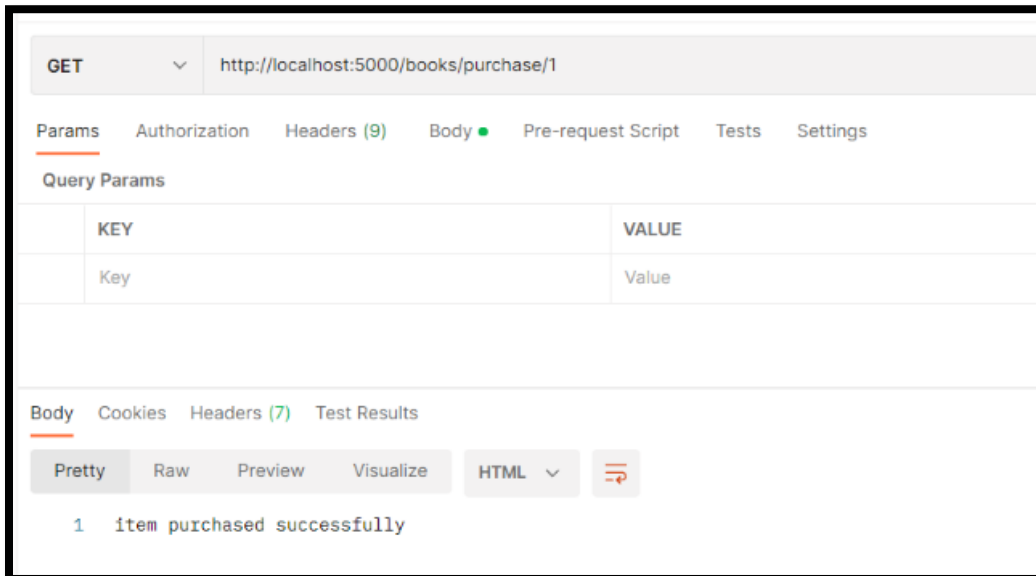
- Update cost of book 2



Purchase book: Request: `localhost:5000/books/purchase/[id]`
Before sending the purchase request.

```
1 {
2   "_id": "63604273d14eede0085885a4",
3   "name": "how to get a good grade in DOS in 40 minutes",
4   "num": 1,
5   "numberinstock": 9,
6   "cost": 5,
7   "topic": "distributed Systems"
8 }
```

- Sending a request to buy book 1



After sending the purchase request.

```
1 {
2   "_id": "63604273d14eede0085885a4",
3   "name": "how to get a good grade in DOS in 40 minutes",
4   "num": 1,
5   "numberinstock": 8,
6   "cost": 5,
7   "topic": "distributed Systems"
8 }
```

Design tradeoffs:

There may be an overhead due to the communication between servers.

Improvements and extensions:

Adding an admin side to manage books.

How to run Program:

1. Install node.js and Express framework on the two machines.
2. Build a database on monogodb with a collection containing the 4 books.
3. Open the frontend terminal and start the server by typing: npm start
4. Open the catalog terminal and start the server by typing: npm start
5. Open the order terminal and start the server by typing: npm start
6. Start sending requests from postman/browser on localhost:5000.

Code:

Frontend server: <https://github.com/MasaKni/front-end>

Catalog server: <https://github.com/lamadarawsheh/CatalogServer>

Order server: <https://github.com/lamadarawsheh/orderServer>