Lab1 – Bazar.com: A multi-tier online book store

In this lab, we have designed a book store as multi-tier architecture. While the concept has been applied by building three servers: Catalog server: Which contains the DB (SQLite), initially the DB has only one table with 4 rows of data as described in the problem. The order server: maintains a list of all orders received for the books. The frontend is implemented as a single microservice, it has the URLs of the main APIs provided to the user to access through other servers and get data.

The frontend supports three main operations: search (topic), info (item-number) and purchase (item-number). The first two operations trigger queries on the catalog server directly, while the last operation triggers the request to the order server.

The catalog server supports only two operations: query and update, query is mainly two types: query-by-subject and query-by-item. The update operation allows the cost of an item to be updated or the number of items in the stock to be increased or decreased.

<u>How to tackle the problem?</u>

We used Flask in python web framework for the backend for each server and SQLite DB.

First of all, we create new virtual machines on VirtualBox, with Ubuntu OS. Each machine is for one server. The machines' network is connected as bridged network. Machines can communicate with each other by URL, specifying the appropriate API, IP addresses and port number of the process.

<u>Install the requirements:</u>

On the machine holds the catalog server, we need SQLite DB:

sudo apt install SQLite3

On each machine, install python3 virtual environment:

Sudo apt install python3-venv

Then we install git:

Sudo apt install git

Create a directory on each machine for creating project and writing code:

Mkdir Catalog_server   // Machine 1

Mkdir Frontend_server   // Machine 2

Mkdir Order_server   // Machine 3

Initializing virtual environment in each directory:

Python3 -m venv Catalog-Venv

Python3 -m venv Catalog-Venv

Python3 -m venv Catalog-Venv

Activate the venv:

. "name-of-venv/bin/activate"

After creating python virtual environment, we can use pip for python to install flask:

Pip install flask   // On each machine

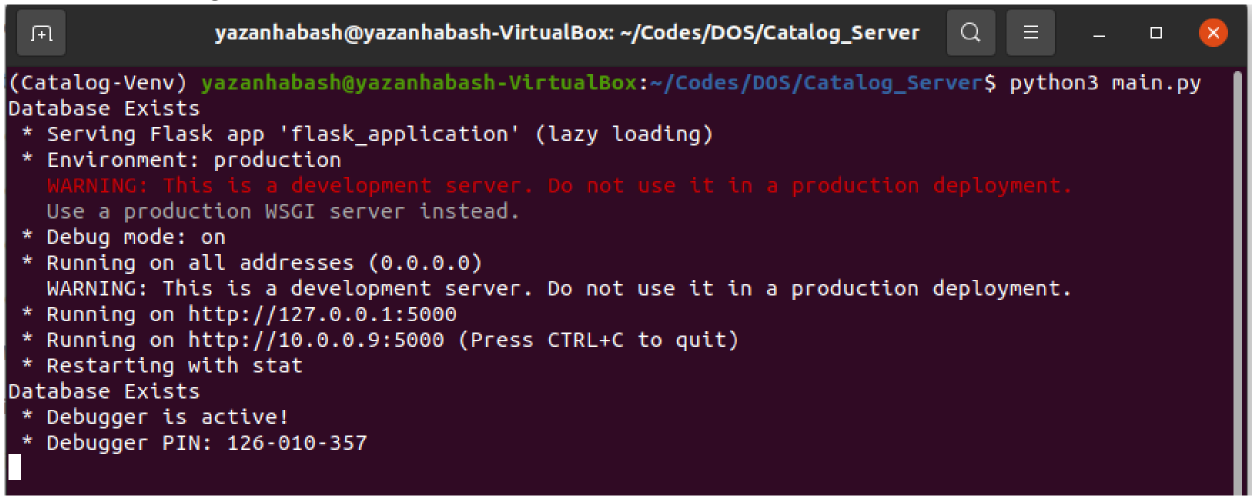
After creating files and writing code for each server, we have files called "flask_application.py" which contains the IP addresses and port numbers for needed another server on other machine. i.e., if the frontend needs to send request to order server, it needs the IP address and port number of the process running the app, etc.

In the main.py, we can specify by the argument **host** that the app can be accessed by any machine, we give it subnet mask as following below:

If __name__ == "__main__":

app.run(host="0.0.0.0", debug=True, port="some port number")

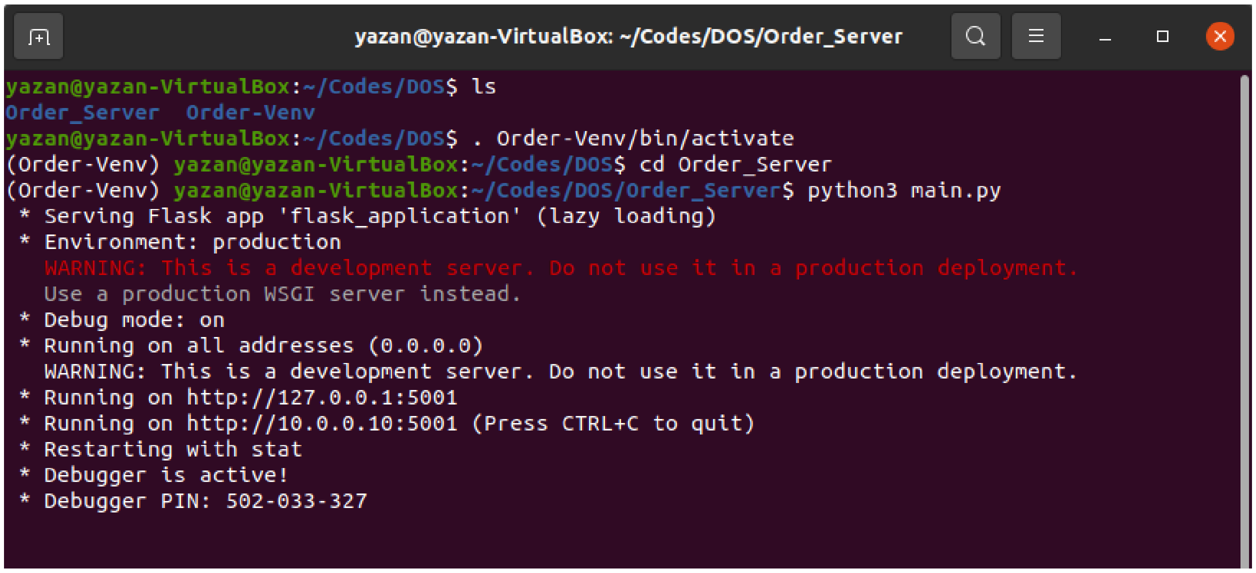
1) Run the catalog server; 1st machine:



yazanhabash@yazanhabash-VirtualBox: ~/Codes/DOS/Catalog_Server

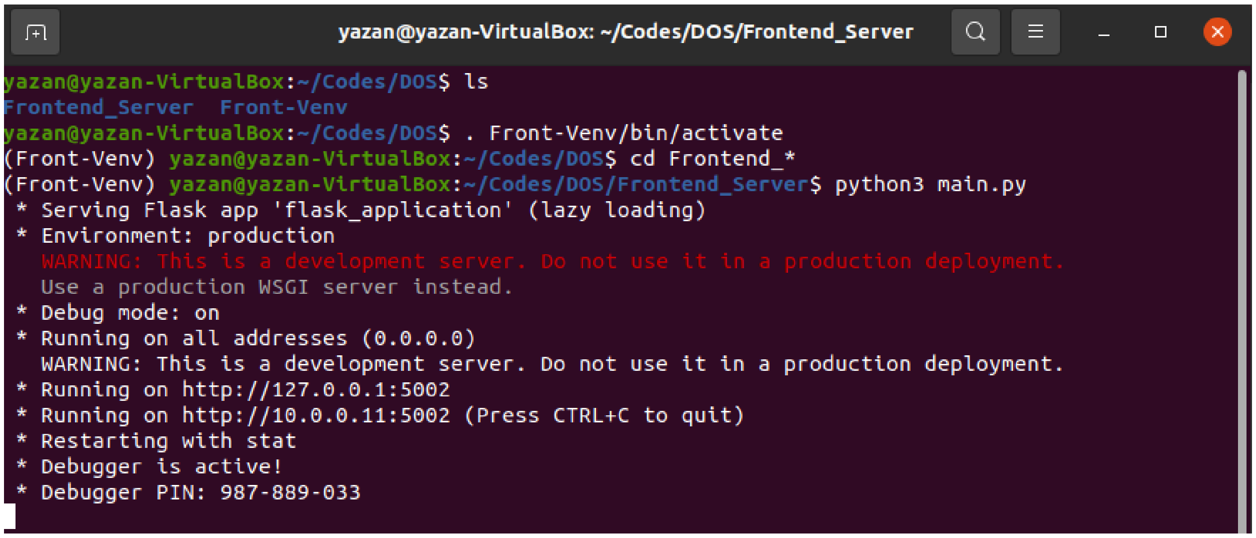
```
(Catalog-Venv) yazanhabash@yazanhabash-VirtualBox:~/Codes/DOS/Catalog_Server$ python3 main.py
Database Exists
 * Serving Flask app 'flask_application' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.0.9:5000 (Press CTRL+C to quit)
 * Restarting with stat
Database Exists
 * Debugger is active!
 * Debugger PIN: 126-010-357
```

2) Run the order server; 2nd machine:



```
yazan@yazan-VirtualBox:~/Codes/DOS$ ls
Order_Server  Order-Venv
yazan@yazan-VirtualBox:~/Codes/DOS$ . Order-Venv/bin/activate
(Order-Venv) yazan@yazan-VirtualBox:~/Codes/DOS$ cd Order_Server
(Order-Venv) yazan@yazan-VirtualBox:~/Codes/DOS/Order_Server$ python3 main.py
 * Serving Flask app 'flask_application' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:5001
 * Running on http://10.0.0.10:5001 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 502-033-327
```

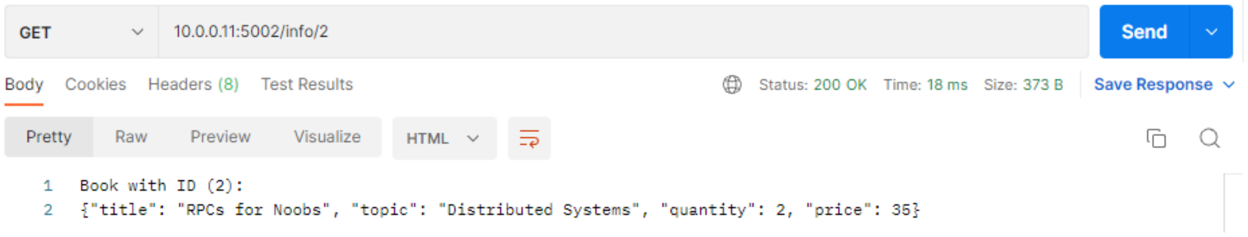
3) Run the frontend server; 3rd machine:



```
yazan@yazan-VirtualBox:~/Codes/DOS$ ls
Frontend_Server  Front-Venv
yazan@yazan-VirtualBox:~/Codes/DOS$ . Front-Venv/bin/activate
(Front-Venv) yazan@yazan-VirtualBox:~/Codes/DOS$ cd Frontend_*
(Front-Venv) yazan@yazan-VirtualBox:~/Codes/DOS/Frontend_Server$ python3 main.py
 * Serving Flask app 'flask_application' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:5002
 * Running on http://10.0.0.11:5002 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 987-889-033
```

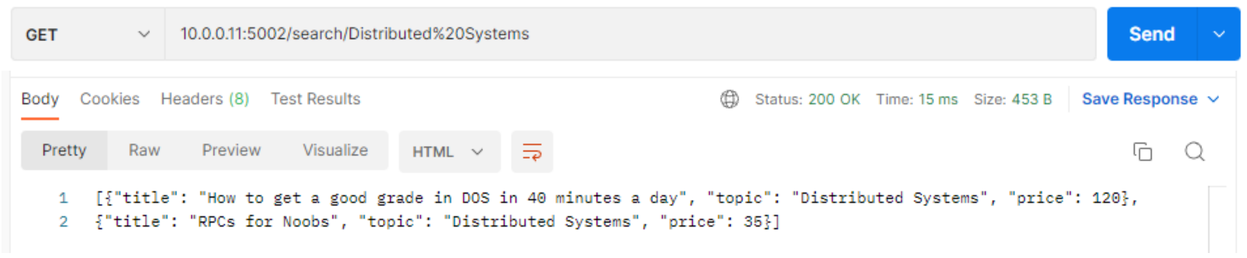
We will test all frontend APIs:

1) Get book by ID, assume client from Postman (Windows host OS will send to frontend server):



```
Book with ID (2):
{"title": "RPCs for Noobs", "topic": "Distributed Systems", "quantity": 2, "price": 35}
```

2) Get books by subject (Distributed Systems):

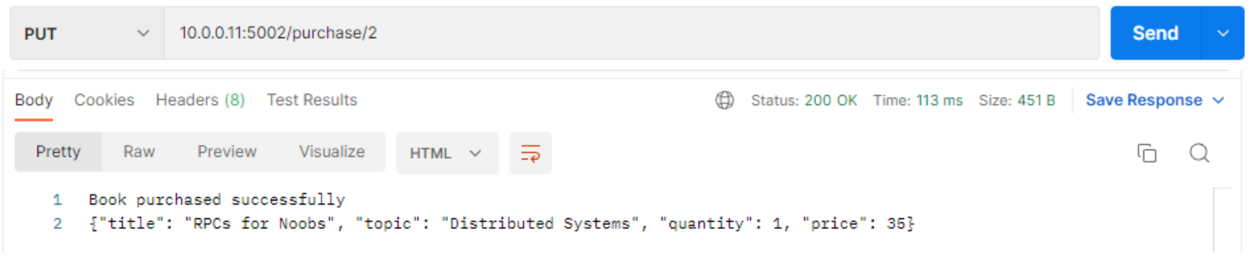


`GET` `10.0.0.11:5002/search/Distributed%20Systems`

Status: 200 OK Time: 15 ms Size: 453 B

```
[{"title": "How to get a good grade in DOS in 40 minutes a day", "topic": "Distributed Systems", "price": 120},
{"title": "RPCs for Noobs", "topic": "Distributed Systems", "price": 35}]
```

3) Purchase Book by ID:

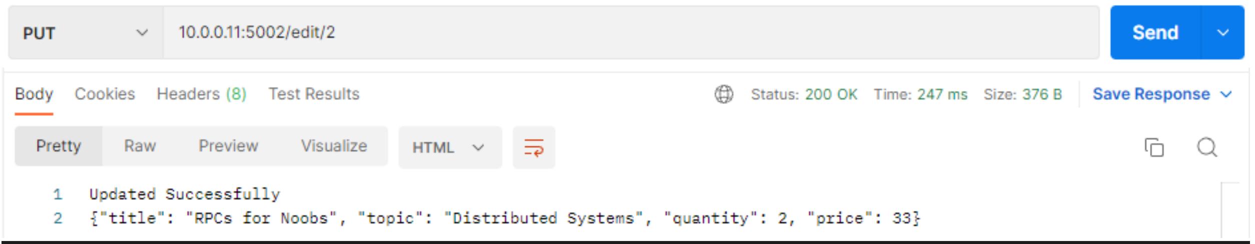


`PUT` `10.0.0.11:5002/purchase/2`

Status: 200 OK Time: 113 ms Size: 451 B

```
Book purchased successfully
{"title": "RPCs for Noobs", "topic": "Distributed Systems", "quantity": 1, "price": 35}
```

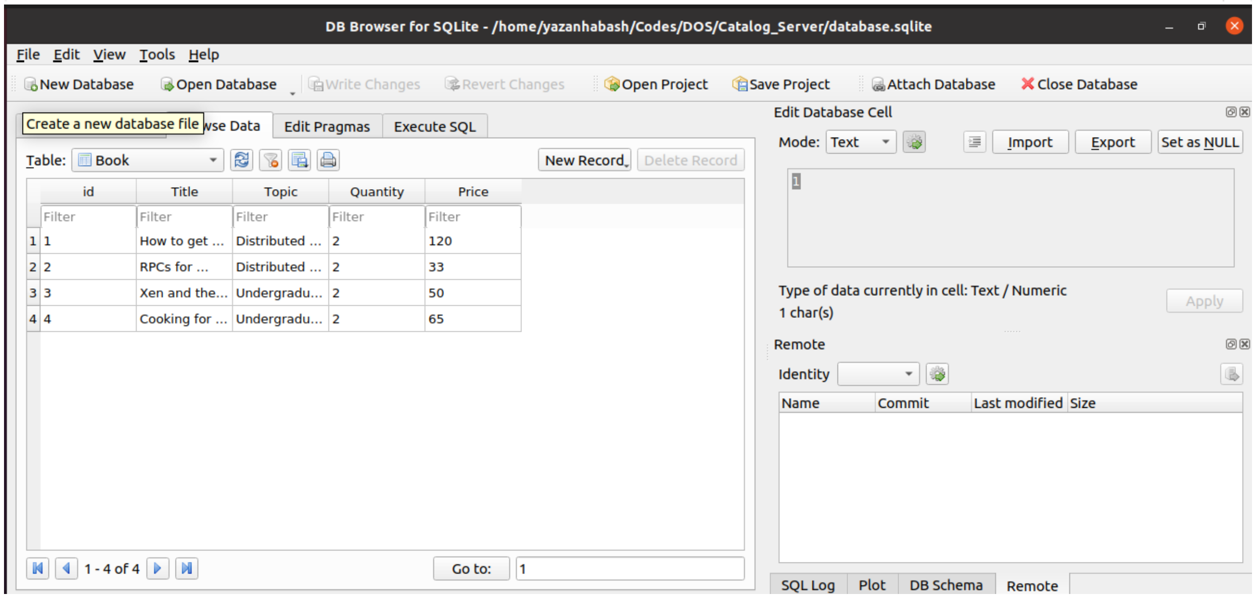
4) Edit the quantity and price of a book:



`PUT` `10.0.0.11:5002/edit/2`

Status: 200 OK Time: 247 ms Size: 376 B

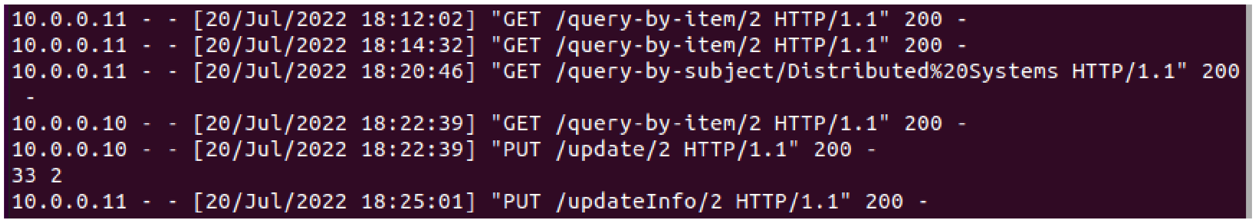
```
Updated Successfully
{"title": "RPCs for Noobs", "topic": "Distributed Systems", "quantity": 2, "price": 33}
```



	id	Title	Topic	Quantity	Price
	Filter	Filter	Filter	Filter	Filter
1	1	How to get ...	Distributed ...	2	120
2	2	RPCs for ...	Distributed ...	2	33
3	3	Xen and the...	Undergradu...	2	50
4	4	Cooking for ...	Undergradu...	2	65

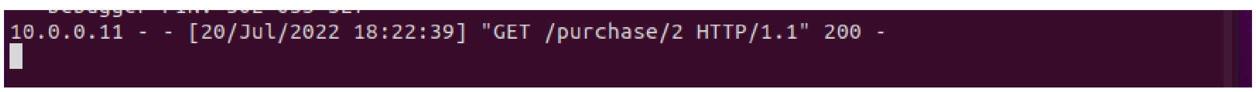
Type of data currently in cell: Text / Numeric
1 char(s)

Requests accepted from catalog server (previous steps):



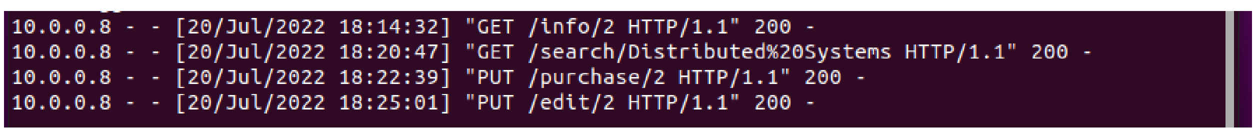
```
10.0.0.11 - - [20/Jul/2022 18:12:02] "GET /query-by-item/2 HTTP/1.1" 200 -
10.0.0.11 - - [20/Jul/2022 18:14:32] "GET /query-by-item/2 HTTP/1.1" 200 -
10.0.0.11 - - [20/Jul/2022 18:20:46] "GET /query-by-subject/Distributed%20Systems HTTP/1.1" 200
 -
10.0.0.10 - - [20/Jul/2022 18:22:39] "GET /query-by-item/2 HTTP/1.1" 200 -
10.0.0.10 - - [20/Jul/2022 18:22:39] "PUT /update/2 HTTP/1.1" 200 -
33 2
10.0.0.11 - - [20/Jul/2022 18:25:01] "PUT /updateInfo/2 HTTP/1.1" 200 -
```

Requests accepted from order server (previous steps):



```
10.0.0.11 - - [20/Jul/2022 18:22:39] "GET /purchase/2 HTTP/1.1" 200 -
```

Requests accepted from the frontend server (previous steps):



```
10.0.0.8 - - [20/Jul/2022 18:14:32] "GET /info/2 HTTP/1.1" 200 -
10.0.0.8 - - [20/Jul/2022 18:20:47] "GET /search/Distributed%20Systems HTTP/1.1" 200 -
10.0.0.8 - - [20/Jul/2022 18:22:39] "PUT /purchase/2 HTTP/1.1" 200 -
10.0.0.8 - - [20/Jul/2022 18:25:01] "PUT /edit/2 HTTP/1.1" 200 -
```

By: Yazan Habash & Ashraf Hab-Rumman