Distributed Operating Systems

Bazar.com

HW #1

هبة ياسر أحمد غانم

11716087

Program Design

The program is built using a two-tier web design, a frontend, and a backend. Each tier has microservices, the frontend is one component while the backend consists of two components: a catalog server and an order server.

The components are distributed, each one runs on a different machine and they are connected through the network and REST API.

To build the program I used Flask, which is a lightweight micro web framework written in Python, and I ran every server file on a separate virtual machine instance. All the instances have an Ubuntu OS. To test the whole design, I built a simple python CLI and run it on the host machine.

For the database, I used simple two CSV files, one for the catalog, which resides on the catalog server, and the other for the orders and it resides on the order server.

Frontend Server

The first server is called Frontend, which is the middleware between the client and other backend services. When a client wants to perform an operation, a request is sent from the client to the frontend server, then it is forwarded to the appropriate server. After that, a response is sent from this server to the frontend server then it is forwarded back to the client. All details happen transparently and away from the client which is talking to the frontend server only.

Catalog Server

The second server is the Catalog server, which is responsible for managing and updating the catalog database, which has all the books in the stock and all information about them.

Order Server

The order server is responsible for recording the purchase orders that it receives and checking if the ordered book is available in the stock by communicating with the catalog server.

There are 3 main operations which a client can perform:

1. **Search:**

When a client wants to fetch the books that have a certain subject providing the request with the subject of the books. The request method for this operation is GET while the URL is as follows:

http://FRONTEND_IP/search/<book_subject>

The request is sent to the frontend server then a request with GET method is sent to the catalog server, to get the list of the books:

http://CATALOG_IP/search/<book_subject>

If the subject exists in the catalog CSV file, a response will be sent to the frontend then forwarded to the client with the list of the books as an array of JSON objects and a status code of 200. But if the database has no books having this subject, a response of *"No books with this subject were found"* message will be sent back, with a status code equals 404.

2. **Info:**

When a client asks for information about a certain book providing the request with the id of the book. The request method for this operation is GET while the URL is as follows:

http://FRONTEND_IP/info/<book_id>

The request is sent to the frontend server then a request with GET method is sent to the catalog server, to get the information of the book:

http://CATALOG_IP/info/<book_id>

If the id exists in the catalog CSV file, then a response will be sent to the frontend then forwarded to the client with the information of the book as a JSON object and status code of 200. However, if the id is not valid, a response of *"Book not found"* message will be sent back, with a status code equal to 404.

3. **Purchase:**

When a client wants to purchase a book with a certain id. The request method for this operation is PUT while the URL is as follows:

http://FRONTEND_IP/purchase/<book_id>

The request is sent to the frontend server then a request with PUT method is sent to the order server:

http://ORDER_IP/purchase/<book_id>

The order server first sends a GET method request to the catalog to get the information of the book and check if it is available in the stock.

http://CATALOG_IP/info/<book_id>

Then if it is available, it will send another request of PUT method to the catalog server to update the quantity of this book.

http://CATALOG_IP/update/<book_id>

It will receive a response of *"The book was bought successfully"* message and it will be sent back to the frontend server that will send it back to the client with a status code of 200.

The order server will add a new item to the orders list holding the information of the purchased book.

But if the book is not available in the stock, order server will return a response of *"The book is out of stock"* message with a status code of 404.

Design Improvements

When a client wants to purchase a book, it sends a request to the frontend server, then it is forwarded to the order server, the order server sends a request to the catalog server to check the availability of the book in the stock. If it is available, it sends another request to the catalog server to modify the quantity of the book. This long process can be improved as the following scenario. When a client asks to purchase a book, a request to the frontend server is sent, then it will be forwarded to the catalog server immediately, which is responsible for checking the availability of the book and updating its quantity. If the book is purchased, a request holding the book information will be sent from the catalog server to the order server which will add a new order to the orders database, but if it is not available then no need to interfere with the order server as no new order will be added to the orders list. This can reduce the number of requests sent to the order server and respond to the client faster.

Figure 1 and Figure 2 represent the system without improvement, in the case of unavailability and availability of the book respectively.

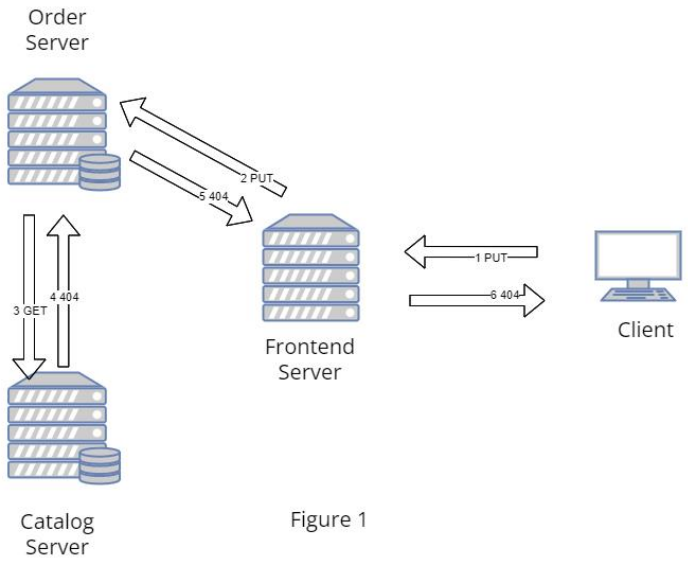


Figure 1

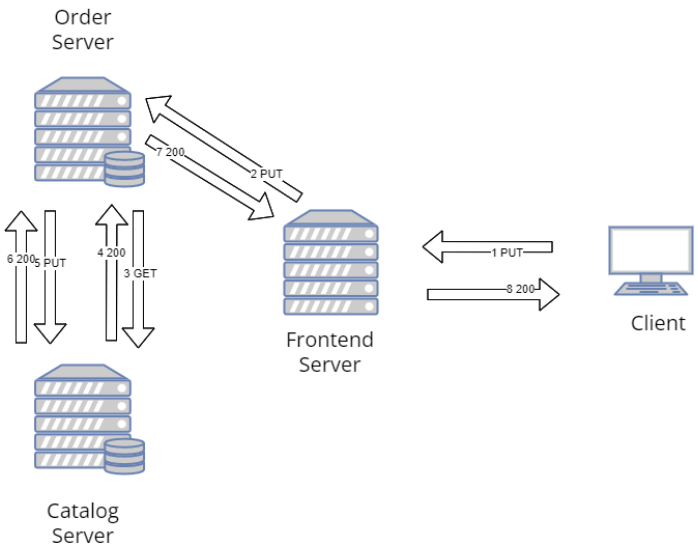


Figure 2

Figure 3 and Figure 4 represent the system with improvement, in the case of unavailability and availability of the book respectively.

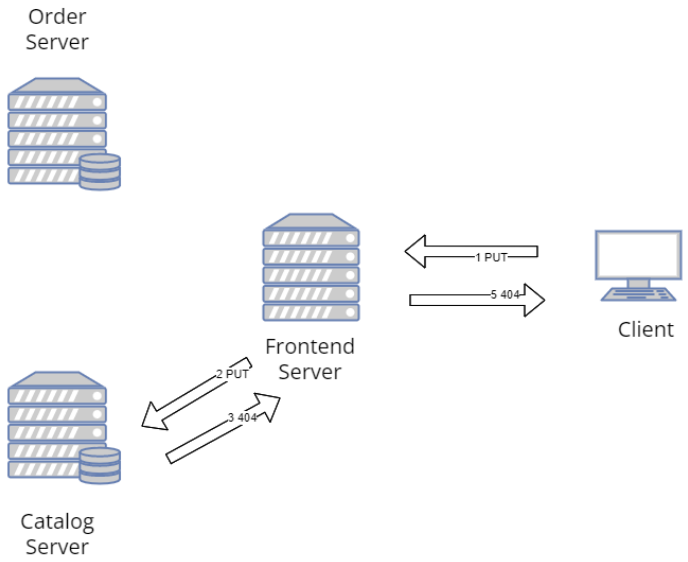


Figure 3

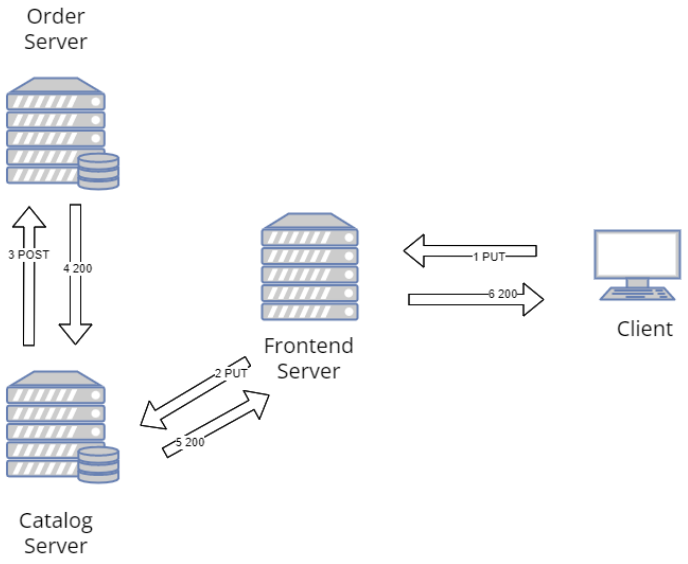
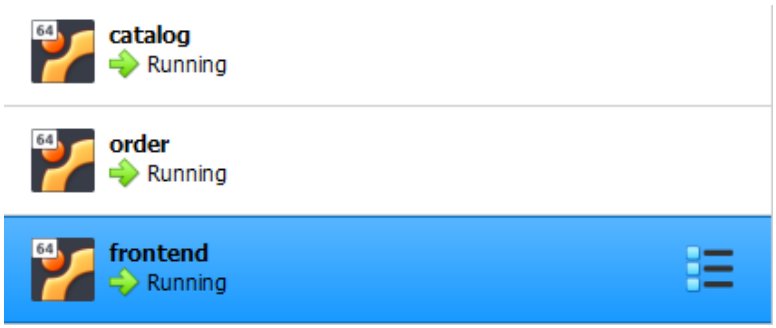


Figure 4

Another modification that can be added is making the catalog dynamic, meaning that there is an admin responsible for adding, deleting and modifying the books without the need of doing this manually.

How to run the program?

1. Create 3 virtual machine instances and install python and flask on each one.



2. Get the Ip address of each instance.

   Catalog Server:

```
hghanim@hghanim-VirtualBox:~$ hostname -I
192.168.1.207
```

   Frontend Server:

```
hghanim@hghanim-VirtualBox:~$ hostname -I
192.168.1.176
```

   Order Server:

```
hghanim@hghanim-VirtualBox:~$ hostname -I
192.168.1.35
```

3. Put each server file and database file on a separate virtual machine instance.
4. Change the Ip addresses in the following files:

   client.py

```
3
4   FRONTEND_IP = 'http://192.168.1.176:5000'
5
```

   frontendServer.py

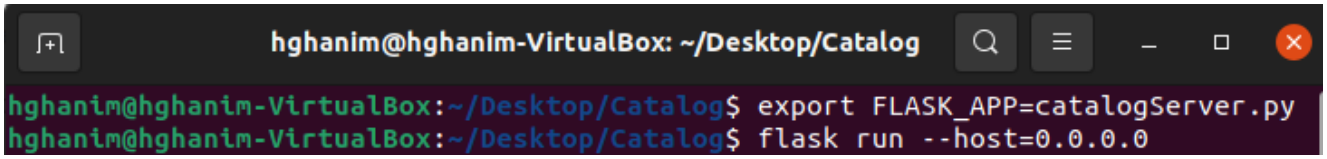
```
4   CATALOG_IP = 'http://192.168.1.207:5000'
5   ORDER_IP = 'http://192.168.1.35:5000'
```

   orderServer.py

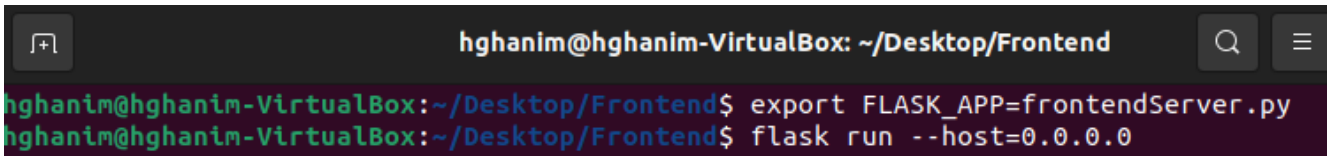
```
5
6   CATALOG_IP = 'http://192.168.1.207:5000'
7
```

5. Run all the servers using the following commands.

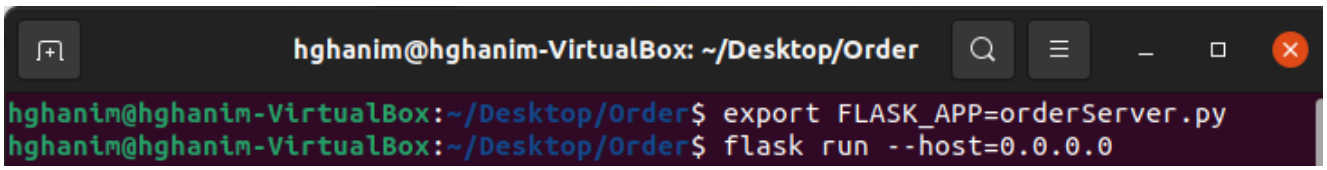
   Catalog Server:



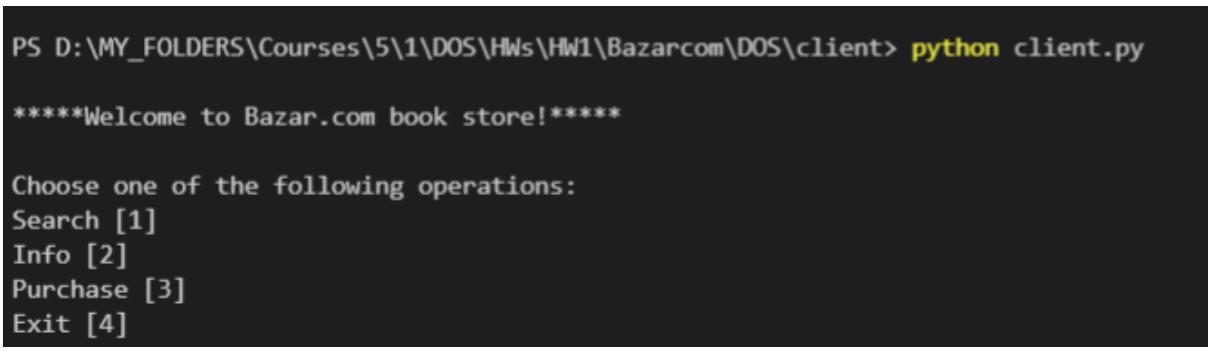
   Frontend Server:



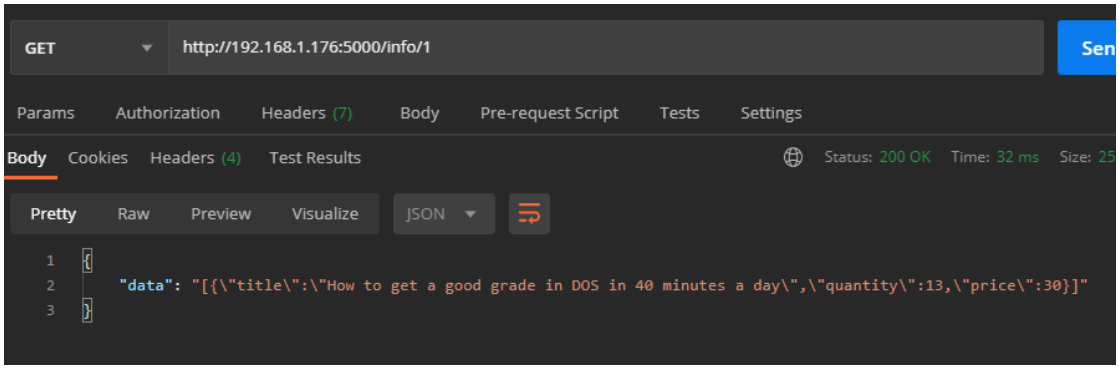
   Order Server:

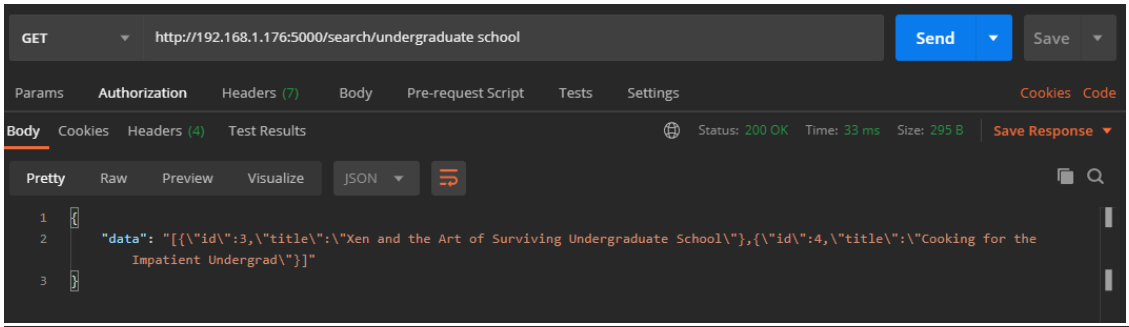


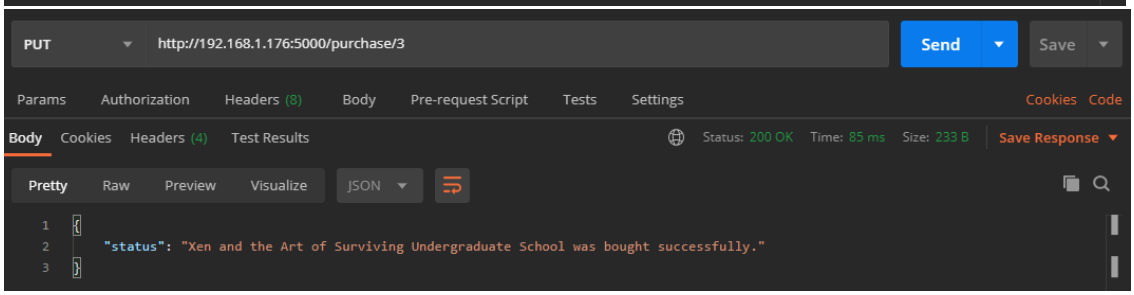
6. Start testing the system by running the client file on the host machine.



7. You can test the program using postman.




GET
http://192.168.1.176:5000/search/undergraduate school
Send
Save
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 33 ms
Size: 295 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"data": "[{\"id\":3,\"title\":\"Xen and the Art of Surviving Undergraduate School\"},{\"id\":4,\"title\":\"Cooking for the Impatient Undergrad\"}]"
}


PUT
http://192.168.1.176:5000/purchase/3
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 85 ms
Size: 233 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"status": "Xen and the Art of Surviving Undergraduate School was bought successfully."
}