

# Distributed Operation Systems

BAZAR.COM: A MULTI-TIER ONLINE BOOK STORE

OLA JAMAL SAADEH SAADEH | 11612344

## INTRODUCTION:

The objective is to create Bazar.com, a online bookstore employing a two-tier web architecture featuring microservices at each tier. The store showcases a selection of four books, and the system comprises a front-end microservice, along with two back-end components: a catalog server and an order server.

The front-end microservice is responsible for receiving and processing user requests. Concurrently, the catalog server oversees the maintenance of the catalog, while the order server manages the processing of purchase orders. This system adheres to a client-server model integrated with a microservices architectural approach.

## OPERATIONS:

- **FRONT-END OPERATIONS:**

1. **search(topic):** Returns entries belonging to a specified topic.
2. **info(item\_number):** Returns details for a given item number.
3. **purchase(item\_number):** Initiates a purchase request.

All routes forward user requests to the catalog and order servers, retrieving the servers' responses, and subsequently, these responses are relayed back to the user as the ultimate response.

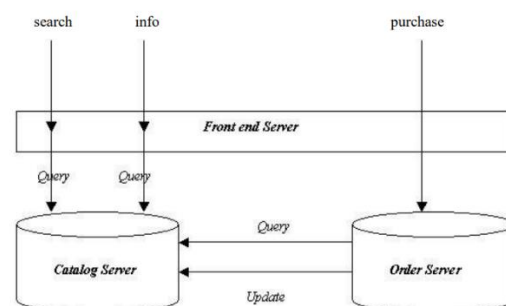
- **CATALOG SERVER OPERATIONS:**

This server maintains a comprehensive database containing information about all books. The server is responsible for managing this database, making it a crucial component. The other servers depend on this central server to obtain various details about the books and to perform edits.

1. **query(subject):** Returns all matching entries based on the specified topic.
2. **query(item\_number):** Returns relevant details for a specified item.
3. **update:** Allows updating the cost or stock of an item.

- **ORDER SERVER OPERATIONS:**

1. **purchase(item\_number):** Verifies item availability and processes the purchase request. In this request, the server establishes a connection with the catalog server to retrieve information about the book, verifying its stock availability. Following that, it generates a new record in the orders database to represent the order.

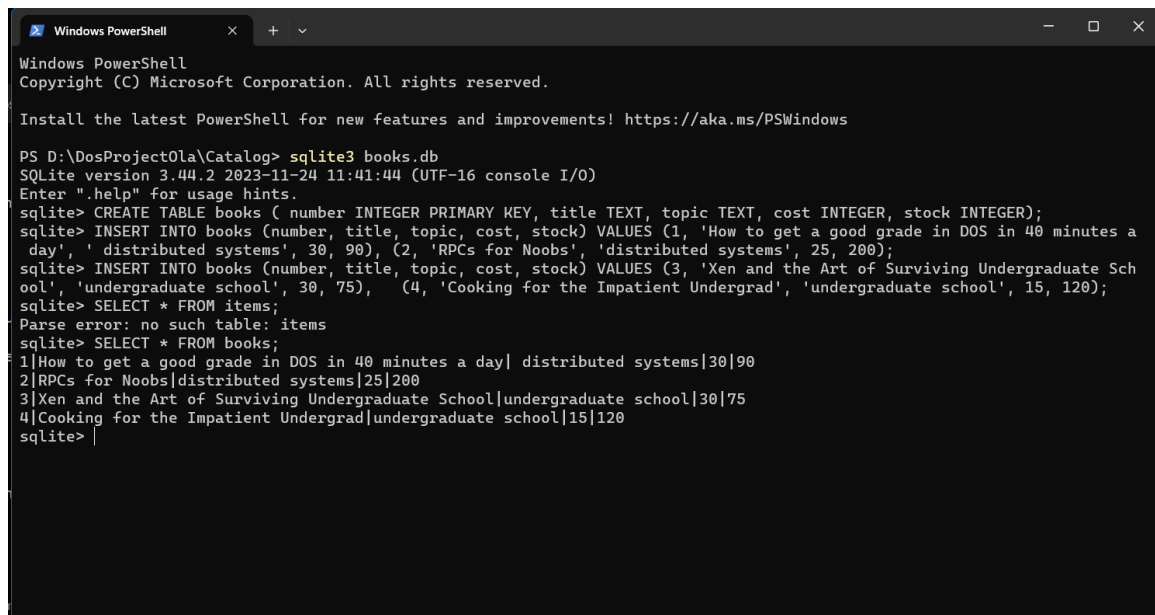


We utilized Docker in our system design, deploying instances of each server independently within containers. This approach provides isolation and flexibility, allowing each container to run its services seamlessly. Docker's containerization technology enhances system scalability, simplifies deployment, and ensures efficient resource utilization

## RUN PROGRAM:

For the implementation of Bazar.com servers, we utilized Express.js, a node.js framework specifically designed for developing REST backend applications and microservices. In conjunction with Express.js, we employed SQLite as the database solution for each server. The choice of Express.js was motivated by its straightforward setup and user-friendly nature, offering convenient features for managing routes and handling requests. Additionally, Node.js provided a wealth of libraries, and we made use of knex.js to streamline our database operations effectively.

### 1. Creating a database of books using sqlite3.

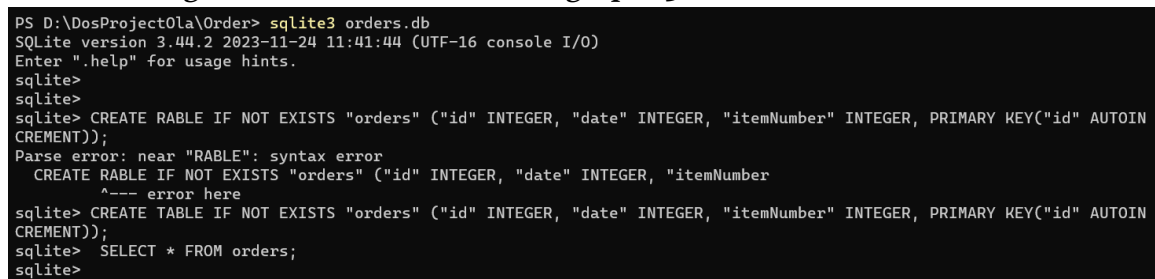


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\DosProject01a\Catalog> sqlite3 books.db
SQLite version 3.44.2 2023-11-24 11:41:44 (UTF-16 console I/O)
Enter ".help" for usage hints.
sqlite> CREATE TABLE books ( number INTEGER PRIMARY KEY, title TEXT, topic TEXT, cost INTEGER, stock INTEGER);
sqlite> INSERT INTO books (number, title, topic, cost, stock) VALUES (1, 'How to get a good grade in DOS in 40 minutes a day', 'distributed systems', 30, 90), (2, 'RPCs for Noobs', 'distributed systems', 25, 200);
sqlite> INSERT INTO books (number, title, topic, cost, stock) VALUES (3, 'Xen and the Art of Surviving Undergraduate School', 'undergraduate school', 30, 75), (4, 'Cooking for the Impatient Undergrad', 'undergraduate school', 15, 120);
sqlite> SELECT * FROM items;
Parse error: no such table: items
sqlite> SELECT * FROM books;
1|How to get a good grade in DOS in 40 minutes a day|distributed systems|30|90
2|RPCs for Noobs|distributed systems|25|200
3|Xen and the Art of Surviving Undergraduate School|undergraduate school|30|75
4|Cooking for the Impatient Undergrad|undergraduate school|15|120
sqlite>
```

### 2. Creating a database for orders using sqlite3.



```
PS D:\DosProject01a\Order> sqlite3 orders.db
SQLite version 3.44.2 2023-11-24 11:41:44 (UTF-16 console I/O)
Enter ".help" for usage hints.
sqlite>
sqlite>
sqlite> CREATE RABLE IF NOT EXISTS "orders" ("id" INTEGER, "date" INTEGER, "itemNumber" INTEGER, PRIMARY KEY("id" AUTOINCREMENT));
Parse error: near "RABLE": syntax error
CREATE RABLE IF NOT EXISTS "orders" ("id" INTEGER, "date" INTEGER, "itemNumber
^--- error here
sqlite> CREATE TABLE IF NOT EXISTS "orders" ("id" INTEGER, "date" INTEGER, "itemNumber" INTEGER, PRIMARY KEY("id" AUTOINCREMENT));
sqlite> SELECT * FROM orders;
sqlite>
```

### 3. By docker build images for three servers and run containers

#### ○ Express-order-app

```
PS D:\DosProject01a\Order> docker build -t express-order-app .
[+] Building 89.8s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 161B
=> [internal] load .dockerignore
=> => transferring context: 90B
=> [internal] load metadata for docker.io/library/node:18
=> [1/5] FROM docker.io/library/node:18@sha256:a17842484dd30af97540e5416c9a62943c709583977b
=> [internal] load build context
=> => transferring context: 102.55kB
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY package.json .
=> [4/5] RUN npm install
```

#### ○ Express-frontend-app

```
PS D:\DosProject01a\FrontEnd> docker build -t express-frontend-app .
[+] Building 93.7s (10/10) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 90B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 161B
=> [internal] load metadata for docker.io/library/node:18
=> [1/5] FROM docker.io/library/node:18@sha256:a17842484dd30af97540e5416c9a62943c709583977b
=> CACHED [2/5] WORKDIR /app
=> [internal] load build context
=> => transferring context: 91.56kB
=> [3/5] COPY package.json .
=> [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:85ab86ce030c7e59628733026ba132fdeb5110b3c5f77a759dcc9636345f6
=> => naming to docker.io/library/express-frontend-app

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
```

#### ○ Express-order-app

```
PS D:\DosProject01a\Order> docker build -t express-order-app .
[+] Building 89.8s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 161B
=> [internal] load .dockerignore
=> => transferring context: 90B
=> [internal] load metadata for docker.io/library/node:18
=> [1/5] FROM docker.io/library/node:18@sha256:a17842484dd30af97540e5416c9a62943c709583977b
=> [internal] load build context
=> => transferring context: 102.55kB
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY package.json .
=> [4/5] RUN npm install
=> [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:2e38847f0ec535c71b802e43f3e17608795d38c56e9d52abc01932acbd3d6e6d
=> => naming to docker.io/library/express-order-app













What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS D:\DosProject01a\Order> docker run --name express-order-app-container -d -p 5000:5000 exp
22bc8627fc0f8978ff6cb4384a5a8d9f1054023ee5980284801689f14a893297
PS D:\DosProject01a\Order>
```

#### 4. Show three containers

```
PS D:\DosProject01a\FrontEnd> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8e42d3a5270f	express-frontend-app	"docker-entrypoint.s..."	2 seconds ago	Up 2 seconds	0.0.0.0:6000->6000/tcp	express-frontend-app
22bc8627fc0f	express-order-app	"docker-entrypoint.s..."	5 minutes ago	Up 5 minutes	0.0.0.0:5000->5000/tcp	express-order-app-co
ab29a2d3d7ac	express-catalog-app	"docker-entrypoint.s..."	12 minutes ago	Up 12 minutes	0.0.0.0:4000->4000/tcp	express-catalog-app-

Name	Image	Status	CPU (%)	Port(s)	Actions
 <a href="#">express-808d711d</a>	<a href="#">express-808d711d</a>	Running	0%	<a href="#">5000:5000</a>	  
 <a href="#">express-25f94683</a>	<a href="#">express-25f94683</a>	Running	0.03%	<a href="#">4000:4000</a>	  
 <a href="#">express-540aa0c9</a>	<a href="#">express-540aa0c9</a>	Running	0%	<a href="#">6000:6000</a>	  

And if I run images ls these three images:

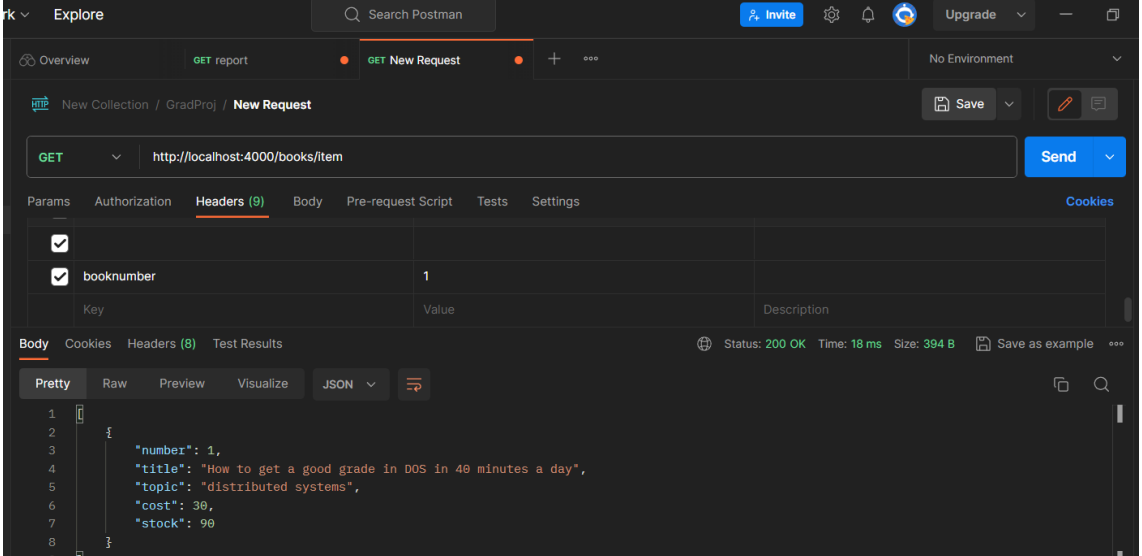
```
PS D:\DosProject01a\FrontEnd> docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
express-frontend-app	latest	85ab86ce030c	15 minutes ago	1.13GB
express-order-app	latest	2e38847f0ec5	16 minutes ago	1.13GB
express-catalog-app	latest	f36ee310fce7	18 minutes ago	1.13GB
catalog-node-app	latest	8d2e88c13061	23 hours ago	1.13GB

## RESULTS:

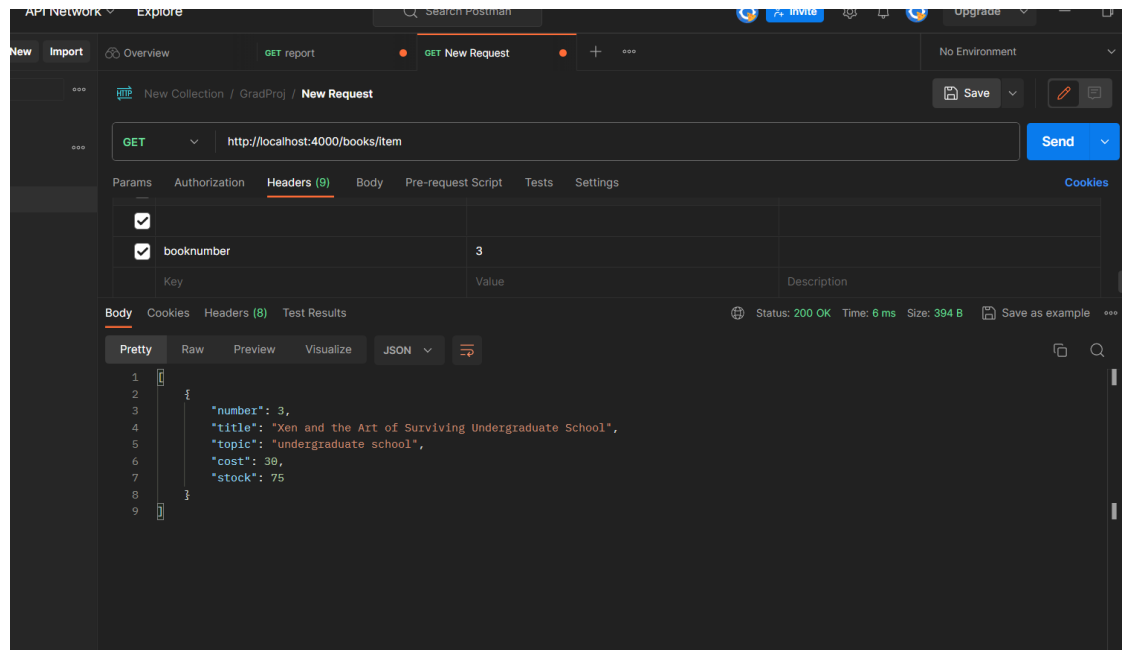
We used the Postman application for thorough testing of all REST services provided by the servers.

#### 5. From catalog server find book by booknumber

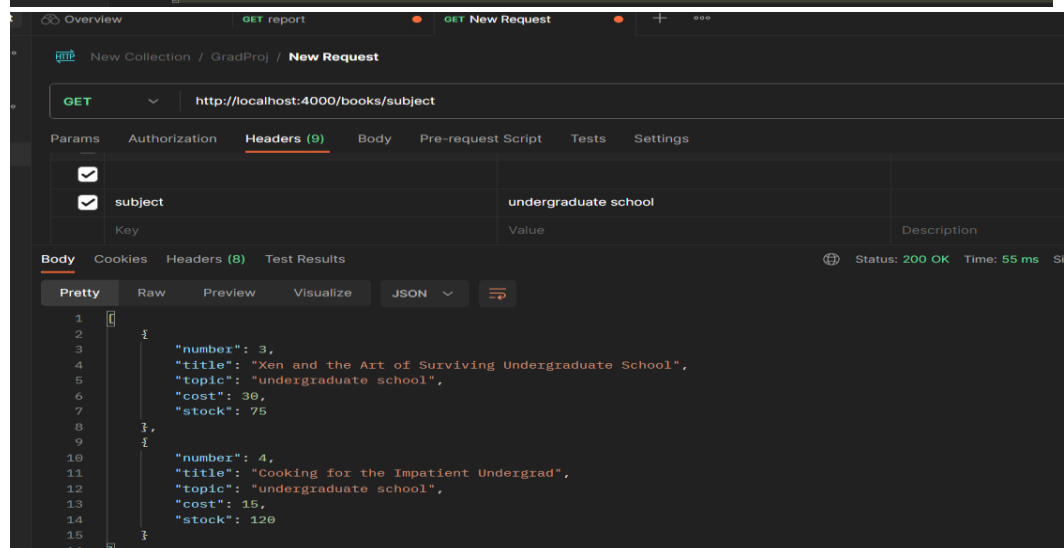
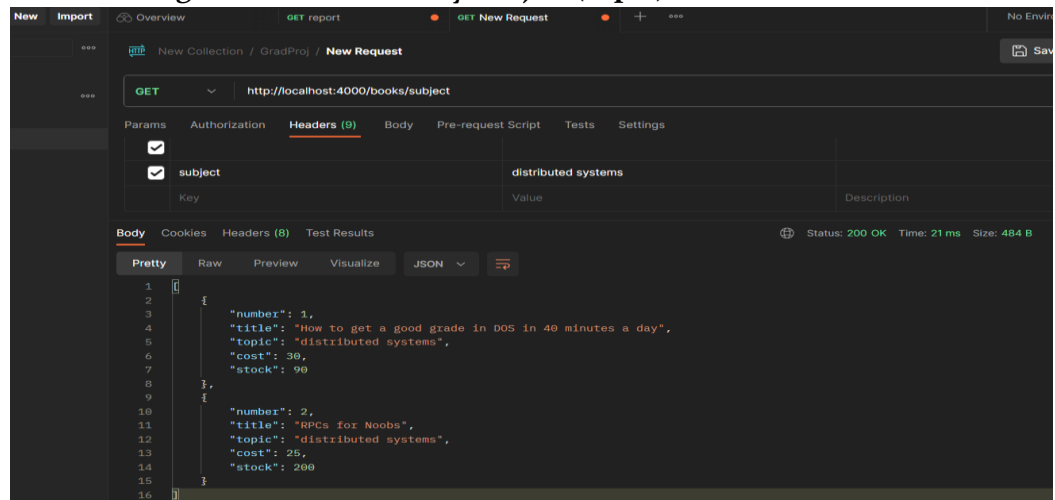


The screenshot shows the Postman interface with a GET request to `http://localhost:4000/books/item`. The 'Headers' tab is active, showing a header `booknumber` with the value `1`. The 'Body' tab is also active, showing the JSON response:

```
{  "number": 1,  "title": "How to get a good grade in DOS in 40 minutes a day",  "topic": "distributed systems",  "cost": 30,  "stock": 90}
```

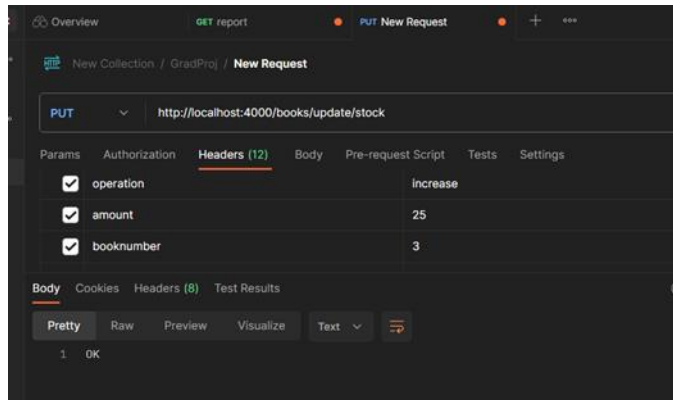


## 6. From catalog server find book by subject(topic)

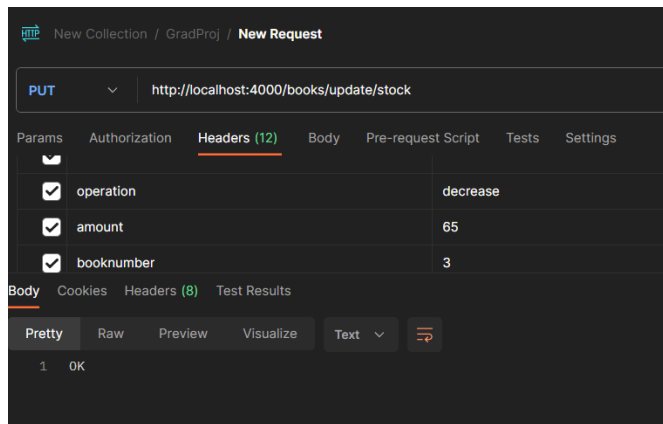


## 7. By catalog server increase or decrease the stock

### ○ Decrease



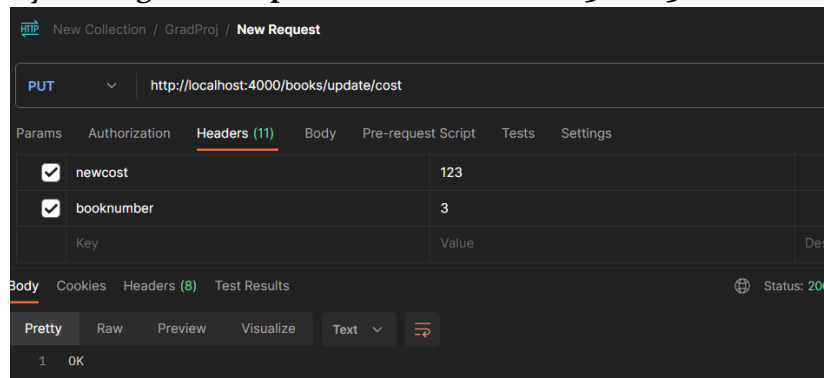
### ○ Increase



### ○ Result: log after updating stock for book3

```
=> "Server is running on port 4000" --- (Catalog\index.js:91)
=> "Updating item 3 stock by increase amount to 25" --- (Catalog\index.js:58)
=> "The new value is: 100" --- (Catalog\index.js:64)
=> "Updated successfully" --- (Catalog\index.js:66)
=> "Updating item 3 stock by increase amount to 25" --- (Catalog\index.js:58)
=> "The new value is: 125" --- (Catalog\index.js:64)
=> "Updated successfully" --- (Catalog\index.js:66)
=> "Updating item 3 stock by decrease amount to 65" --- (Catalog\index.js:58)
=> "The new value is: 60" --- (Catalog\index.js:64)
=> "Updated successfully" --- (Catalog\index.js:66)
```

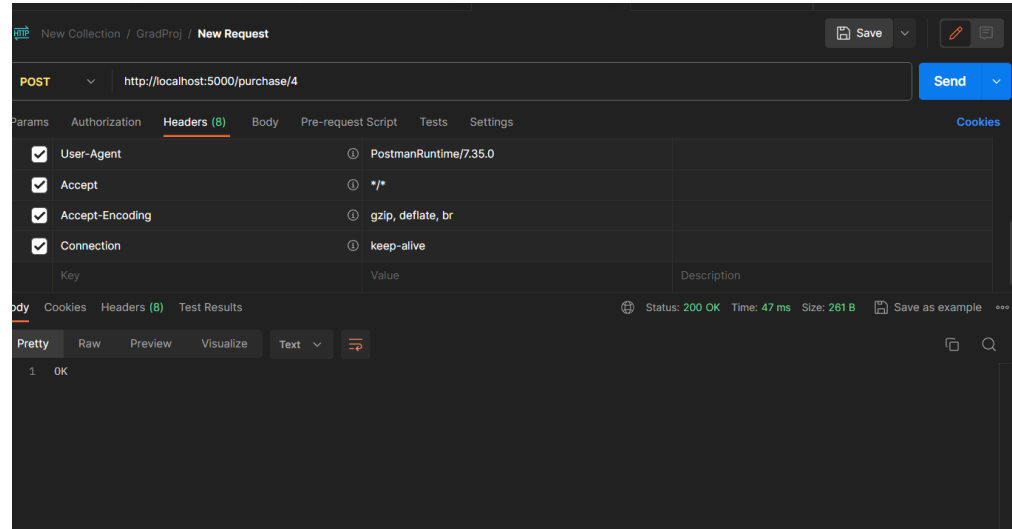
## 8. By catalog server update the cost of book 3 to 123



- **Result :**The cost of book<sub>3</sub> was updated as shown:

```
sqlite> SELECT * FROM books;
1|How to get a good grade in DOS in 40 minutes a day|distributed systems|30|90
2|RPCs for Noobs|distributed systems|25|200
3|Xen and the Art of Surviving Undergraduate School|undergraduate school|123|60
4|Cooking for the Impatient Undergrad|undergraduate school|15|120
sqlite>
```

## 9. On order server made a purchase



- **Result : logs**

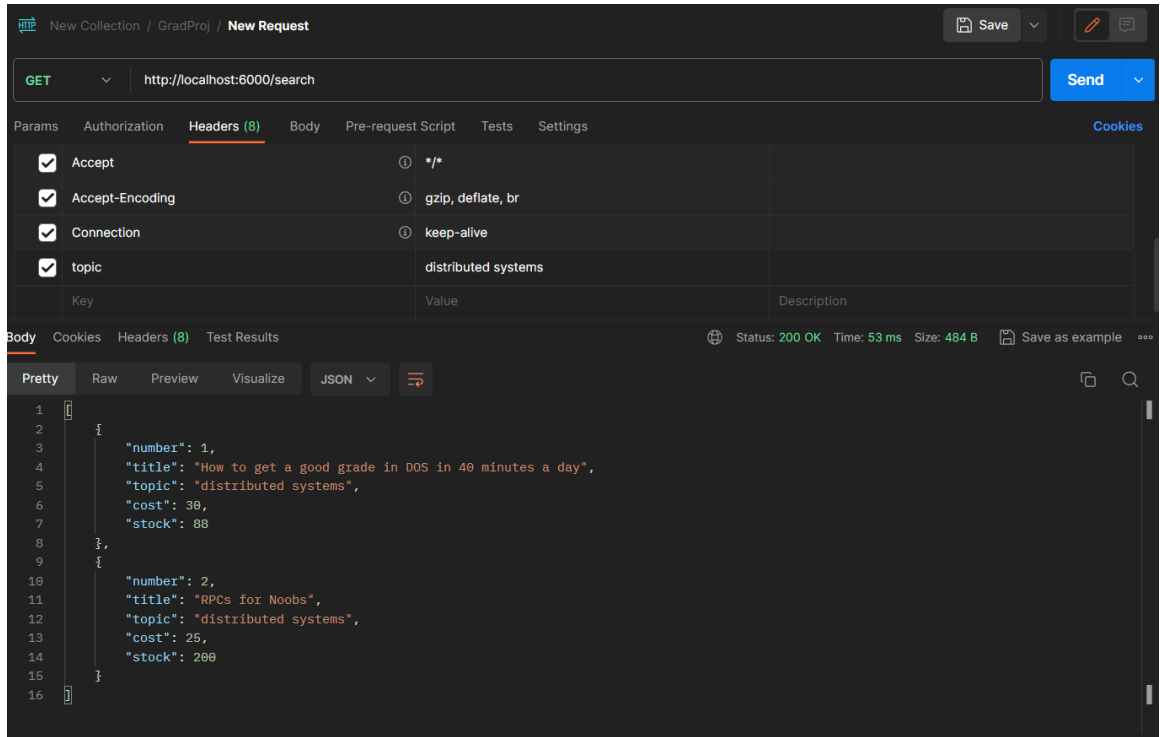
```
[INFO] 27-11-2023 12:58:08:686 ==> "server is running on port 5000" --- (Order\index.js:70)
[INFO] 27-11-2023 12:58:09:562 ==> "new order to buy item: 4" --- (Order\index.js:29)
[INFO] 27-11-2023 12:58:09:649 ==> "item data is: [{\"number\":4,\"title\": \"Cooking for the Impatient Undergrad\", \"topic\": \"undergraduate school\", \"cost\":15, \"stock\":108}]" --- (Order\index.js:34)
[INFO] 27-11-2023 12:58:09:650 ==> "the number of books is now 108 and its decreased to 107" --- (Order\index.js:42)
[INFO] 27-11-2023 12:58:09:676 ==> "update the stock of the item" --- (Order\index.js:53)
[INFO] 27-11-2023 12:58:09:691 ==> "result of update: OK" --- (Order\index.js:55)
```

- **Database of order**

```
sqlite> SELECT * FROM orders;
1|1701081219214|4
2|1701081384344|4
3|1701081747988|4
4|1701081756589|4
sqlite>
```



## 10. By front end server search by item and info(topic)



New Collection / GradProj / New Request

GET  Send

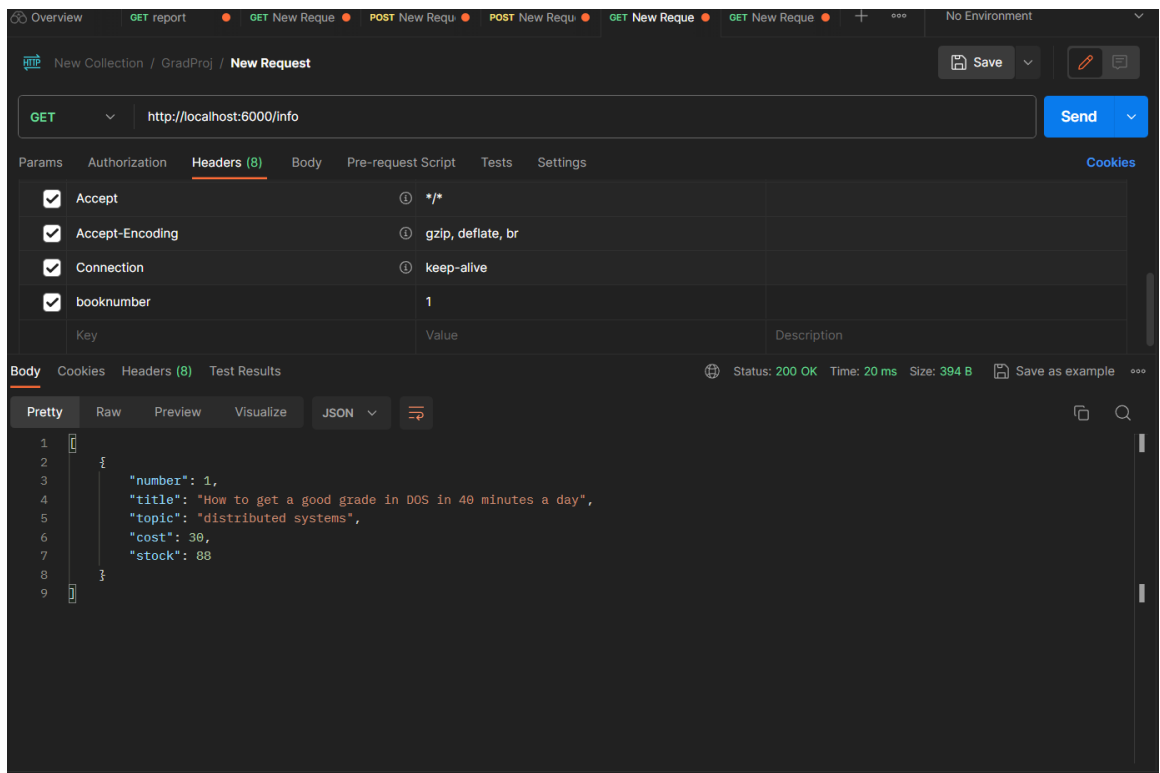
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Key	Value	Description
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
topic	distributed systems	

Body Cookies Headers (8) Test Results Status: 200 OK Time: 53 ms Size: 484 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "number": 1,
4     "title": "How to get a good grade in DOS in 40 minutes a day",
5     "topic": "distributed systems",
6     "cost": 30,
7     "stock": 88
8   },
9   {
10    "number": 2,
11    "title": "RPCs for Noobs",
12    "topic": "distributed systems",
13    "cost": 25,
14    "stock": 200
15  }
16 }
```



Overview GET report GET New Reque POST New Reque POST New Reque GET New Reque GET New Reque No Environment

New Collection / GradProj / New Request

GET  Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

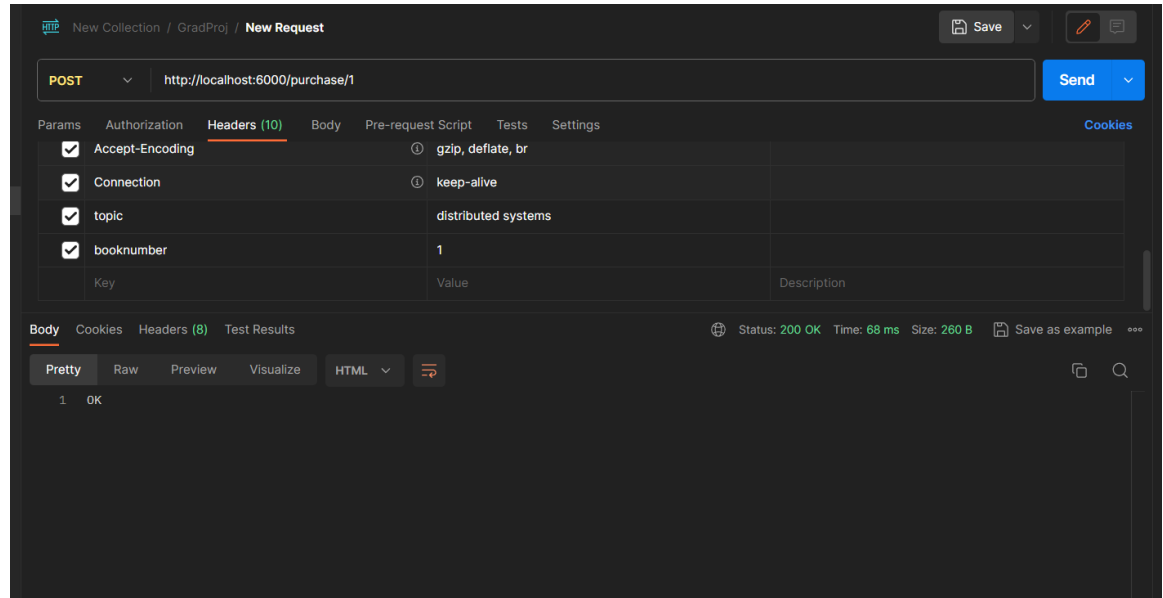
Key	Value	Description
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
booknumber	1	

Body Cookies Headers (8) Test Results Status: 200 OK Time: 20 ms Size: 394 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "number": 1,
4     "title": "How to get a good grade in DOS in 40 minutes a day",
5     "topic": "distributed systems",
6     "cost": 30,
7     "stock": 88
8   }
9 }
```

## 11. And purchase on frontend server



### ○ Result: logs

```
=> "server is running on port 6000" --- (FrontEnd\index.js:105)
=> "*****" --- (FrontEnd\index.js:86)
=> "Start a purchase order for the book with number: 1" --- (FrontEnd\index.js:89)
=> "Order processing on the server: http://localhost:5000" --- (FrontEnd\index.js:90)
=> "Successful payment for the selected item." --- (FrontEnd\index.js:94)
```

## CONCLUSION:

In conclusion, the development of Bazar.com, a microservices-based online book store, has been successfully undertaken following the guidelines provided. The system architecture, consisting of a front-end microservice, catalog server microservice, and order server microservice, demonstrates the application of a distributed and microservices design.