

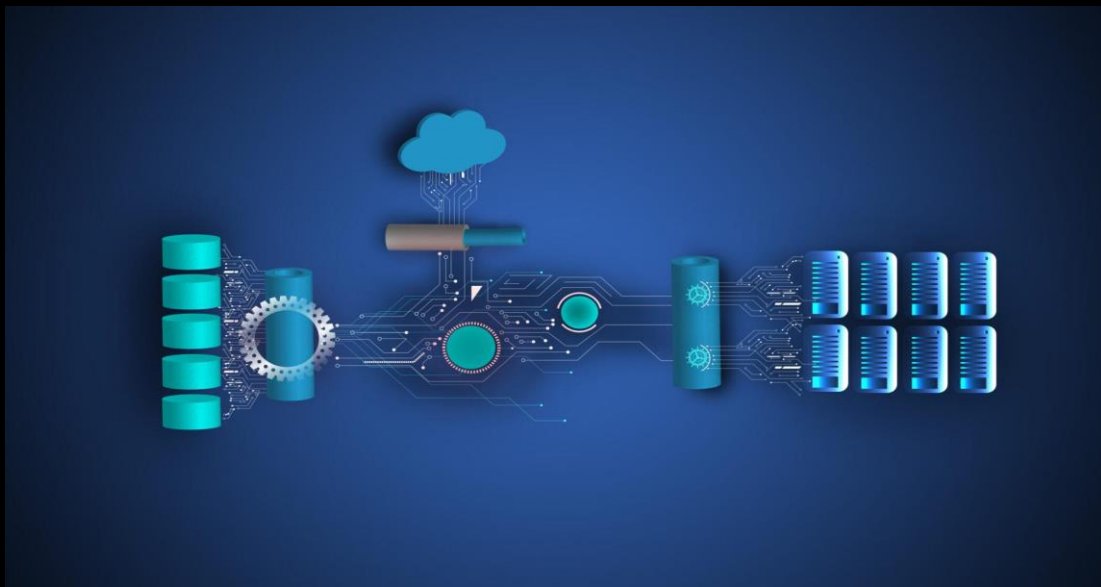
An-Najah national University
Faculty of Engineering and IT



Distributed Operation Systems (10636456)

Bazar.com: A Multi-tier Online Book Store

Part1



Eyad Al Hasan | 11924095

Eman Khaled | 12028272

In this project, we construct a microservice to establish an electronic store for the purpose of selling books. This store has been developed utilizing three microservices:

- **Catalog Service:** This service talks directly to the database and shows resources for searching and editing book information.
- **Order Service:** This service manages all user orders. It checks if the book is available in the catalog and has sufficient stock.
- **Front-end Service:** This service is what users interact with to use Bazar. It provides search and view options that connect to the catalog service, and a purchase option that connects to the order service.



The flask microservice was used to create this website. The front-end "user interface" is displayed on the page, allowing users to access the various features available in the e-shop:

- For Catalog Service **5001:**
This service provides two endpoints for other services to utilize: query and update. The query endpoint can handle two types of GET requests: query/item and query/topic. The query/topic request takes a search text in the URI and searches for it in the database. It returns a list of books that match the topic. If no books are found, an empty list is returned. On the other hand, the query/item request takes the ID of a book in the URI and returns the book if it exists. If the book is not found, a 404 NOT FOUND status code is returned.
The update endpoint accepts PUT requests to modify book entries. Through this endpoint, any field of a book (except the ID) can be modified by passing them in a JSON object. If a field is not passed, it remains unmodified. The book ID is received in

the URI. If the book exists, the endpoint responds with the updated book fields in a JSON object. If the book is not found, a 404 NOT FOUND status code is returned.

➤ For Order Service **5002**:

This platform only reveals a single endpoint, purchase. When a PUT request is made with the book ID in the URI, it first verifies if the ID is numeric. If not, the request is rejected. Next, it communicates with the catalog service to retrieve the book details linked to that ID. If the book is located, it examines the quantity available. If there are enough copies, it asks for a decrease in the quantity by one. If not, it informs that the book is currently out of stock.

➤ For Front-End Service **5000**:

This service provides the endpoints that the user can use to search, find information, and make purchases. The search endpoint allows the user to search for a specific topic by sending a GET request with the topic included in the URI. This request is then forwarded to the catalog service as a query for that topic. The lookup endpoint, on the other hand, allows the user to find a book by sending a GET request with the book ID included in the URI. Before forwarding the request to the catalog server as a query for that item, it checks if the ID is numeric. Lastly, the buy endpoint functions similarly to the lookup endpoint, but it requires the use of the PUT method for requests. These requests are then forwarded to the buy endpoint on the order service.

All the books and related information are stored in a sqlite database in the catalog server, like this:

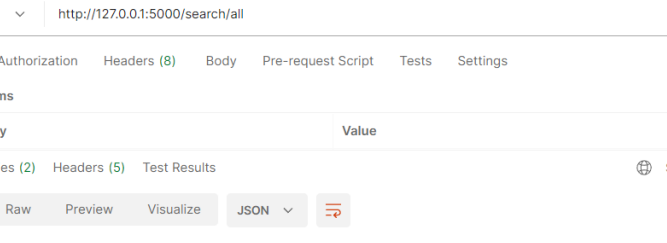
| | id INTEGER PRIMARY KEY | book_id INTEGER NOT NULL REFERENCES books(id) | + |
|---|------------------------|---|---|
| 1 | 1 | 1 | |
| 2 | 2 | 1 | |
| 3 | 3 | 1 | |
| 4 | 4 | 1 | |
| 5 | 5 | 1 | |
| 6 | 6 | 1 | |
| 7 | 7 | 1 | |

There were multiple servers, each running on a separate machine. The first server was operating on Windows, while the other two were running on Ubuntu. These servers interacted with each other through the network by utilizing the HTTP protocol.

We test the functionality of the three servers by using Postman to send requests back and forth. The front end server is where all the requests start, since it's where the client interacts.

➤ **Search:**

URL: <http://127.0.0.1:5000/search/all>



The screenshot shows a web browser window with the address bar displaying 'http://127.0.0.1:5000/search/all'. The page title is 'bazar_front / search'. The main content area shows a search results page. At the top, there's a navigation bar with links for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. Below this, there's a 'Query Params' section. The main content area is divided into two parts: 'Body' and 'Cookies (2)'. The 'Body' section is active, showing a JSON response. The JSON is displayed in a 'Pretty' view, showing an array of two objects. The first object represents a book with id 1, price 12.99, quantity 14, title 'Book 1', and topic 'Topic 1'. The second object represents a book with id 2, price 15.99, quantity 3, title 'Book 2', and topic 'Topic 2'.

| id | price | quantity | title | topic |
|----|-------|----------|--------|---------|
| 1 | 12.99 | 14 | Book 1 | Topic 1 |
| 2 | 15.99 | 3 | Book 2 | Topic 2 |

URL: <http://127.0.0.1:5000/search/Topic 1>

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

- URL:** `http://127.0.0.1:5000/search/Topic 1`
- Method:** GET
- Status:** 200 OK
- Time:** 5 ms
- Size:** 276 B
- Response Body (JSON):**

```
[{"id": 1, "price": 12.99, "quantity": 14, "title": "Book 1", "topic": "Topic 1"}]
```

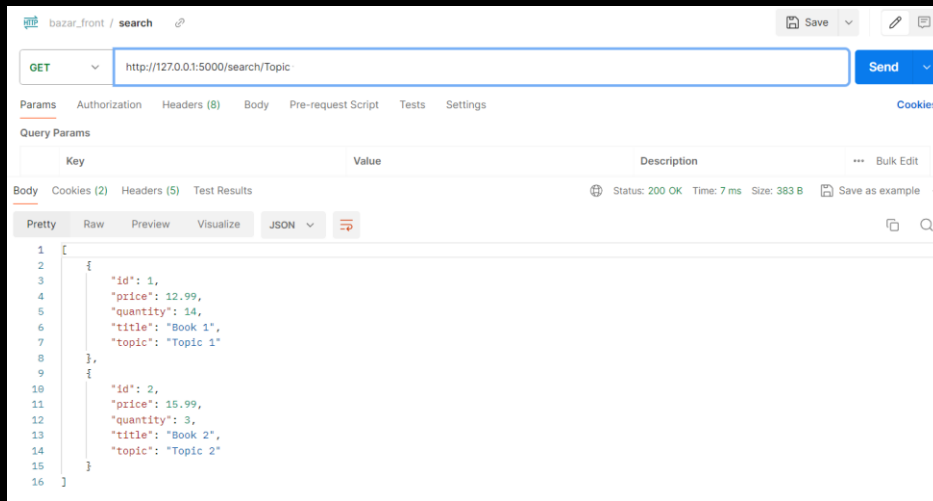
URL: <http://127.0.0.1:5000/search/Topic 2>

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

- URL:** `http://127.0.0.1:5000/search/Topic 2`
- Method:** GET
- Status:** 200 OK
- Time:** 7 ms
- Size:** 275 B
- Response Body (JSON):**

```
[{"id": 2, "price": 15.99, "quantity": 3, "title": "Book 2", "topic": "Topic 2"}]
```

URL: <http://127.0.0.1:5000/search/Topic>



bazar_front / search

GET <http://127.0.0.1:5000/search/Topic> Send

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

Query Params

| Key | Value | Description | Bulk Edit |
|-----|-------|-------------|-----------|
|-----|-------|-------------|-----------|

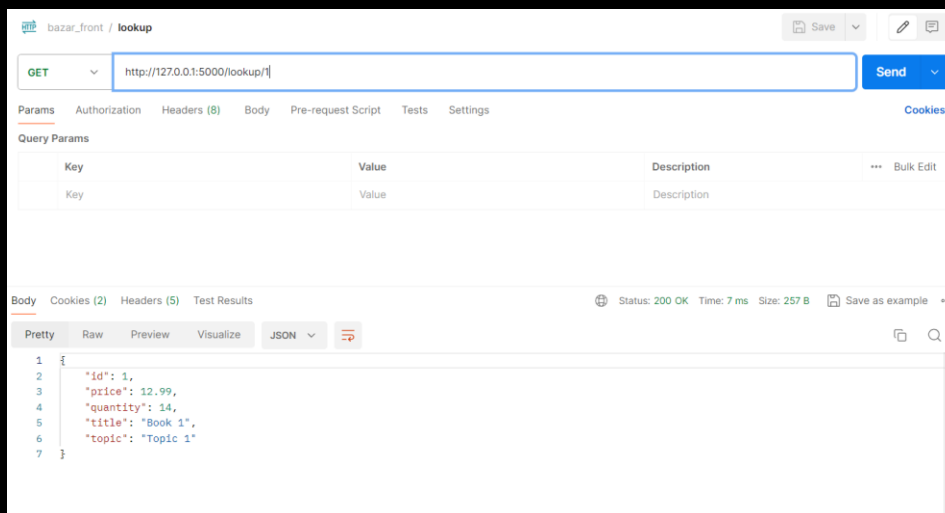
Body Cookies (2) Headers (5) Test Results Status: 200 OK Time: 7 ms Size: 383 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 1,
4     "price": 12.99,
5     "quantity": 14,
6     "title": "Book 1",
7     "topic": "Topic 1"
8   },
9   {
10    "id": 2,
11    "price": 15.99,
12    "quantity": 3,
13    "title": "Book 2",
14    "topic": "Topic 2"
15  }
16 }
```

➤ Lookup:

URL: <http://127.0.0.1:5000/lookup/1>



bazar_front / lookup

GET <http://127.0.0.1:5000/lookup/1> Send

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

Query Params

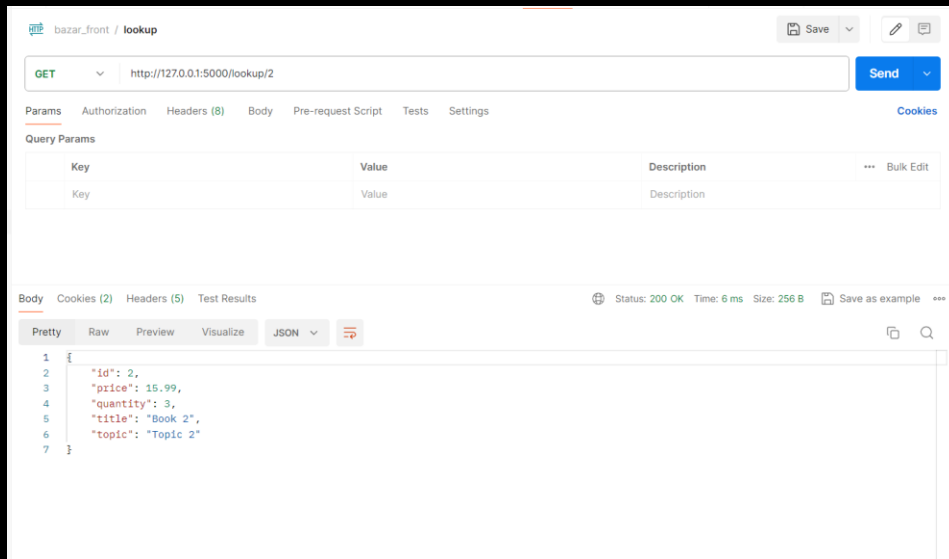
| Key | Value | Description | Bulk Edit |
|-----|-------|-------------|-----------|
|-----|-------|-------------|-----------|

Body Cookies (2) Headers (5) Test Results Status: 200 OK Time: 7 ms Size: 257 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "price": 12.99,
4   "quantity": 14,
5   "title": "Book 1",
6   "topic": "Topic 1"
7 }
```

URL: <http://127.0.0.1:5000/lookup/2>



bazar_front / lookup

GET http://127.0.0.1:5000/lookup/2 Send

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

Query Params

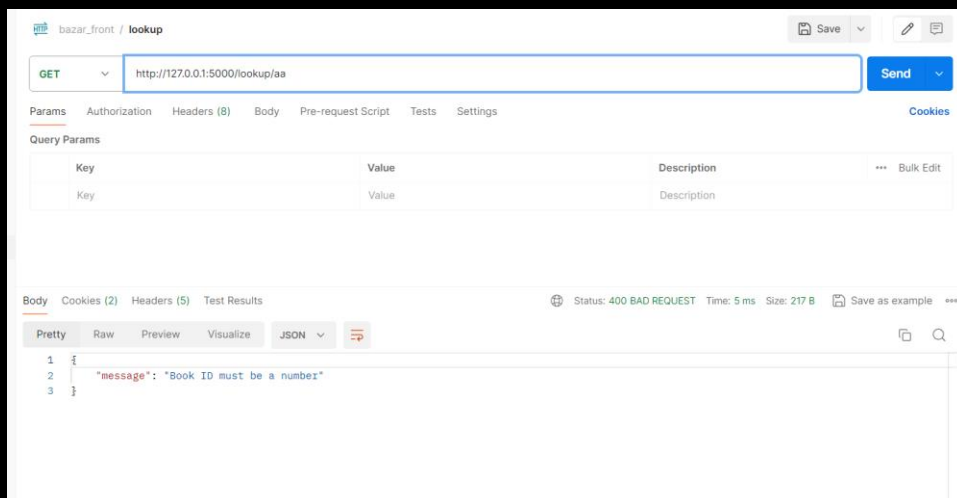
| Key | Value | Description | Bulk Edit |
|-----|-------|-------------|-----------|
| Key | Value | Description | |

Body Cookies (2) Headers (5) Test Results Status: 200 OK Time: 6 ms Size: 256 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "price": 15.99,
4   "quantity": 3,
5   "title": "Book 2",
6   "topic": "Topic 2"
7 }
```

URL: <http://127.0.0.1:5000/lookup/aa>



bazar_front / lookup

GET http://127.0.0.1:5000/lookup/aa Send

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

Query Params

| Key | Value | Description | Bulk Edit |
|-----|-------|-------------|-----------|
| Key | Value | Description | |

Body Cookies (2) Headers (5) Test Results Status: 400 BAD REQUEST Time: 5 ms Size: 217 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Book ID must be a number"
3 }
```

➤ Query:

URL: <http://127.0.0.1:5001/query/item/1>

bazar-catalog / get-by-item

GET <http://127.0.0.1:5001/query/item/1> Send

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

Query Params

| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | | |

Body Cookies (2) Headers (5) Test Results Status: 200 OK Time: 8 ms Size: 257 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "price": 12.99,
4   "quantity": 14,
5   "title": "Book 1",
6   "topic": "Topic 1"
7 }
```

URL: <http://127.0.0.1:5001/query/item/2>

bazar-catalog / get-by-item

GET <http://127.0.0.1:5001/query/item/2> Send

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

Query Params

| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | | |

Body Cookies (2) Headers (5) Test Results Status: 200 OK Time: 8 ms Size: 255 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 2,
3   "price": 1.99,
4   "quantity": 0,
5   "title": "Book 2",
6   "topic": "Topic 2"
7 }
```


URL: <http://127.0.0.1:5001/query/item/6>

bazar-catalog / get-by-item

GET <http://127.0.0.1:5001/query/item/6> Send

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

Query Params

| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | | |

Body Cookies (2) Headers (5) Test Results Status: 404 NOT FOUND Time: 7 ms Size: 203 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": "Book not found"
3 }
```

URL: <http://127.0.0.1:5001/query/item/aa>

bazar-catalog / get-by-item

GET <http://127.0.0.1:5001/query/item/aa> Send

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

Query Params

| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | | |

Body Cookies (2) Headers (5) Test Results Status: 400 BAD REQUEST Time: 5 ms Size: 217 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Book ID must be a number"
3 }
```

URL: <http://127.0.0.1:5001/query/topic/Topic 1>

bazar-catalog / get-by-topic

GET <http://127.0.0.1:5001/query/topic/Topic 1> Send

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

Query Params

| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | | |

Body Cookies (2) Headers (5) Test Results Status: 200 OK Time: 7 ms Size: 276 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "price": 12.99,
5     "quantity": 14,
6     "title": "Book 1",
7     "topic": "Topic 1"
8   }
9 ]
```

➤ Update:

URL: <http://127.0.0.1:5001/update/1>

bazar-catalog / update

PUT <http://127.0.0.1:5001/update/1> Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

1 { "price": 9.99, "quantity": 12 }

Body Cookies (2) Headers (5) Test Results Status: 200 OK Time: 12 ms Size: 256 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "price": 9.99,
4   "quantity": 12,
5   "title": "Book 1",
6   "topic": "Topic 1"
7 }
```

➤ Purchase:

URL: <http://127.0.0.1:5002/buy/1/>

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

- Method:** PUT
- URL:** http://127.0.0.1:5002/buy/1/
- Status:** 200 OK
- Time:** 23 ms
- Size:** 230 B
- Body (JSON):**

```
1 {
2   "message": "Book purchased successfully",
3   "success": true
4 }
```

URL: <http://127.0.0.1:5002/buy/2/>

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

- Method:** PUT
- URL:** http://127.0.0.1:5002/buy/2/
- Status:** 200 OK
- Time:** 14 ms
- Size:** 230 B
- Body (JSON):**

```
1 {
2   "message": "Book purchased successfully",
3   "success": true
4 }
```

URL: <http://127.0.0.1:5000/buy/1/>

bazar_front / buy

PUT

http://127.0.0.1:5000/buy/1/

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | | |

Body

Cookies (2)

Headers (5)

Test Results

Status: 200 OK

Time: 20 ms

Size: 230 B

Save as example

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"message": "Book purchased successfully",

3

"success": true

4

}

URL: <http://127.0.0.1:5000/buy/2/>

bazar_front / buy

PUT

http://127.0.0.1:5000/buy/2/

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

| Key | Value | Description | ... | Bulk Edit |
|-----|-------|-------------|-----|-----------|
| Key | Value | Description | | |

Body

Cookies (2)

Headers (5)

Test Results

Status: 400 BAD REQUEST

Time: 6 ms

Size: 230 B

Save as example

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"message": "Book out of stock",

3

"success": false

4

}

Improvements:

- Instead of inefficiently checking for availability, providing a response, and then sending an update request, a more streamlined approach would be to delegate the purchase process to the catalog server. In this setup, the order server would only need to query the availability, and if the quantity is not zero, proceed with the update operation directly.
- Currently, the purchase process is designed to buy a single item. To enhance user experience, it is recommended to allow users to specify the quantity of items they want to purchase. This can be achieved by including the number of copies in the request body while maintaining the existing verbs and communication flow between servers. The order server can then adjust its validation logic accordingly.