An-Najah National University

Faculty of Engineering

Computer Engineering Department

Distributed Operation Systems

Lab 1: Bazar.com: A Multi-tier Online Book Store

Sewar Aslan, Marwa AbuSaa

12028585, 12028718

Contents

Introduction

The store will apply a microservices approach at each tier of its two-tier web architecture, comprising a front-end and a backend where the front-end tier provides initial processing on the user requests. The back-end consists of two tiers: Catalog server and Order Server.

Materials

To complete this lab, we used the following:

- **Docker Desktop**
- **Docker Compose**
- **Node.js (Express)**
- **Postman (Test API)**

Architecture

1. Front-end:

   The front-end tier will accept user requests and perform initial processing. And coordinates the communication of the user interface with the Catalog service or Order service according to the users' requests. So it acts as an entry point for users' requests.

2. Catalog Server:

   It has a database of the books available including details such as title, cost, stock, and topic. It is a microservice responsible for maintaining and providing information on the available books. It implements a RESTful API that allows for operations such as updating book information and performing topic or item number queries for books, and it's implemented as an independent process.

3. Order Server:

   It’s responsible for managing customer orders. The OrderServer, being a microservice, talks to the front-end and catalog server in order to ease the purchasing process. This microservice provides a single operation endpoint via RESTful API and is designed to handle the purchase process efficiently.

Implementation

The project was developed using Express in conjunction with Node.js, which made efficient use of its ability to facilitate the creation of lightweight microservices. Express.js is a minimalist, flexible Node.js web application framework that provides an extremely powerful set of features for web and mobile applications. This project utilizes Axios to make HTTP requests between servers. Axios is a promise-based HTTP client for the browser and Node.js, which lets one make asynchronous HTTP requests to REST endpoints. And for the database we used DB.js file which contains a JavaScript object that represents a small collection of books. Each book entry includes an id, title, stock (number of copies available), cost (in some currency), and topic.

Setup and Installation

1. Clone the repository to your local machine: https://github.com/SewarAslan/Bazarcom.git
2. Navigate to the project directory: cd Bazarcom
3. Build the Docker images and start the containers: docker-compose up –build
4. The system should now be running on Docker containers.

Testing the APIs using API test tool

You can test APIs using tools like Postman. Here are the endpoints available:

- Frontend APIs:

URL	METHOD	DESCRIPTION
/Bazarcom/info/:id	GET	Get information about a book by its <ID>
/Bazarcom/search/:topic	GET	Search for books by their <topic>,return all matching books IDs and Titles
/Bazarcom/ purchase/:id	POST	Buy a book by its <ID>

- Catalog APIs:

URL	METHOD	DESCRIPTION
/catalogServer/query	GET	Allows users to search a book based on either topic or id
/CatalogServer/updateStock/:itemNumber	PUT	Update the stock of a specific book based on its (ID) when it has bought.

- Order APIs:

URL	METHOD	DESCRIPTION
/OrderServer/purchase/:itemNumber	POST	Handles the purchase of a specific book by its (ID)

Examples:

➢ From Frontend:

Search Books by Topic:

Endpoint: http://localhost:2000/Bazarcom/Search/:topic

Example: http://localhost:2000/Bazarcom/Search/ distributed systems

Get Book Information:

Endpoint: http://localhost:2000/Bazarcom/info/:id

Example: http://localhost:2000/Bazarcom/info/102

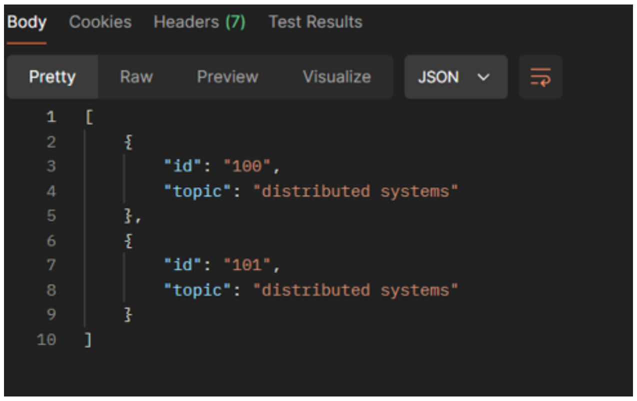
Purchase Book:

Endpoint: http://localhost:2000/Bazarcom/purchase/:id

Example: http://localhost:2000/Bazarcom/purchase/101

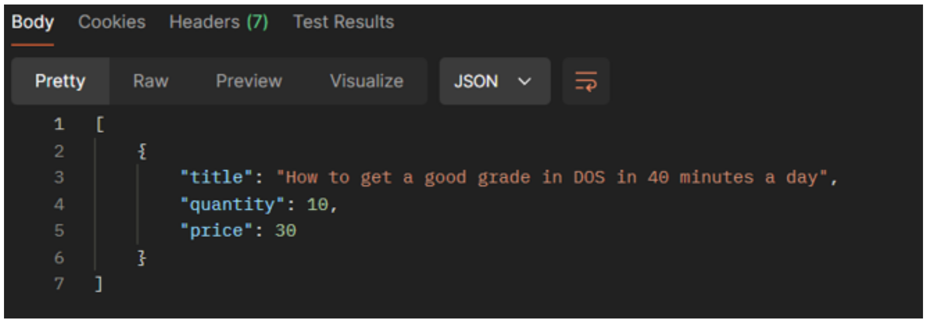
Output

Search Books by Topic:



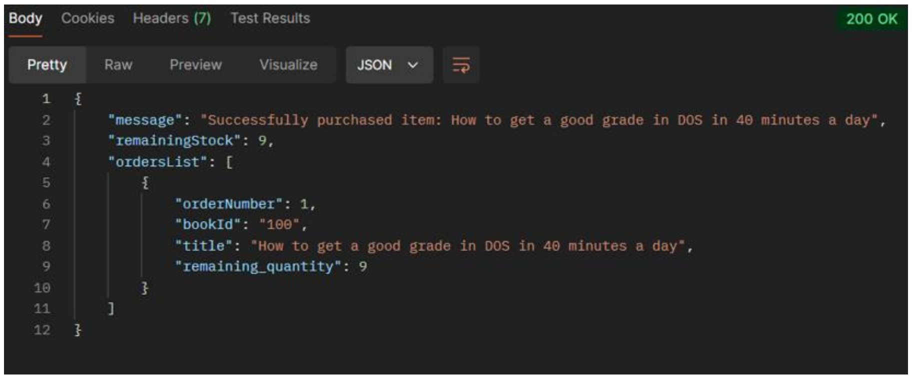
```
[
    {
        "id": "100",
        "topic": "distributed systems"
    },
    {
        "id": "101",
        "topic": "distributed systems"
    }
]
```

Get Book Information:



```
[
    {
        "title": "How to get a good grade in DOS in 40 minutes a day",
        "quantity": 10,
        "price": 30
    }
]
```

Purchase Book:



```
{
    "message": "Successfully purchased item: How to get a good grade in DOS in 40 minutes a day",
    "remainingStock": 9,
    "ordersList": [
        {
            "orderNumber": 1,
            "bookId": "100",
            "title": "How to get a good grade in DOS in 40 minutes a day",
            "remaining_quantity": 9
        }
    ]
}
```

Note: The rest of outputs are existed on our github repository:
https://github.com/SewarAslan/Bazarcom.git

Conclusion

In this lab, we developed a multitier online bookstore system using microservices architecture. We implemented a multitier online bookstore system by realizing the catalog and order servers independently as microservices, mediating all user interactions via a frontend. This made the entire platform scalable and efficient. The use of Docker containers developed an easy deployment process whereby each service can be deployed and managed independently. Additionally, testing via Postman allowed us to verify that all APIs went well and provided the necessary endpoints for book search, stock management, or order processing. We learned from this lab quite a great deal of how to develop distributed systems using state-of-the-art web technologies and witnessed the flexibility that microservices-based architecture allows for in handling complex processes such as online purchases.