بسم الله الرحمن الرحيم

AN-NAJAH NATIONAL UNIVERSITY

FACULTY OF ENGINEERING AND INFORMATION TECHNOLOGY

Computer Engineering Department

Online Book Store (Bazar.com)

Samah Sheeha & Eba Khalil

28 October 2024

Abstract

The Bazar.com project is a simple, multi-tier online bookstore application with just four books available for sale. It's designed using a microservices architecture, where each service operates independently to improve organization and scalability. The project is divided into three main services: the frontend, catalog server, and order server.

Introduction

In our fast-paced digital world, users expect applications that are not only functional but also easy to navigate and enjoyable to use. This project sets out to meet that demand by creating a user-friendly web application designed to connect users with various services seamlessly. Our application consists of three main parts: a Frontend that provides an engaging interface, a Catalog Server that organizes and manages product information, and an Order Server that handles user requests and transactions efficiently.

Our goal is to build a cohesive platform that enhances the user experience while maintaining strong performance and flexibility. By leveraging modern technologies and frameworks, we aimed to create a solution that not only addresses today's needs but can also evolve with future requirements.

In the following sections, we'll explore the design, implementation, and key features of each component, offering a detailed look at how our project came together and the impact it aims to have.

Services Overview

A. Frontend Service

Purpose: The frontend acts as the user-facing layer of the application, providing an API that allows users to search for books by category, view book details, and make purchases.

Implementation: This service is built using Java. It uses the requests library to communicate with the catalog and order servers.

Endpoints:

/search/<subject>: Sends a request to the catalog server to retrieve books by category.

/info/<item_number>: Fetches detailed information about a specific book by item number.

/purchase/<item_number>: Sends a purchase request to the order server.

B. Catalog Server

Purpose: The catalog server manages the book information, including stock levels, prices, and categories.

Implementation: This service is also developed in Java. Book data is stored in a straightforward CSV file (catalog.csv) that's loaded and queried as needed.

Endpoints:

/search/<subject>: Returns a list of books under a specific category (e.g., "Distributed Systems").

/info/<item_number>: Provides information on a book's title, available quantity, and price.

Persistence: The catalog data is stored in a CSV file that acts like a simple database, ensuring that information is saved across sessions.

C. Order Server

Purpose: The order server processes book purchase requests and manages stock levels by communicating with the catalog server.

Implementation: Like the other services, the order server is built in Java and uses the requests library to interact with the catalog server.

Endpoints:

/purchase/<item_number>: Checks if a book is in stock and decreases the inventory count if the purchase is successful.

Data Management: Purchase records are stored in a CSV file (orders.csv) as a simple transaction log. The server verifies stock by querying the catalog server before confirming any orders.

Docker and Deployment

Each of these services runs in its own Docker container, providing isolated environments that are easy to manage and deploy. We use Docker Compose to build and deploy all three services at once, with each service assigned a unique port for inter-service communication.

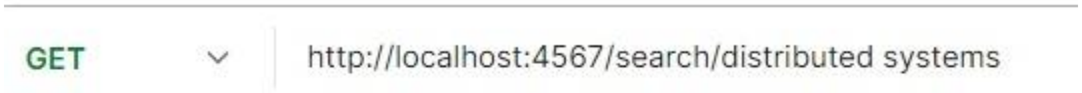
Practical steps

We have two text files: one for orders and the other for the catalog. Initially, the orders file is empty.

```
orderId,bookId,quantity,date
```

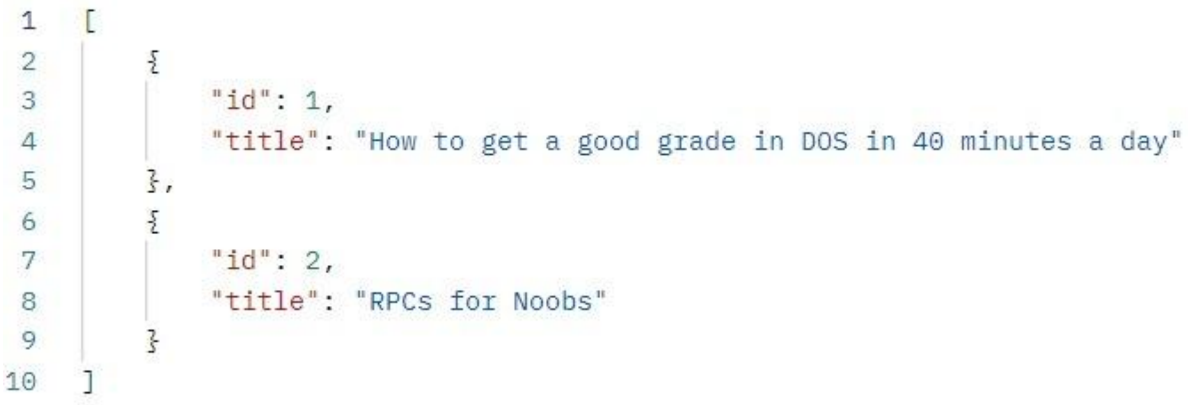
```
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a
day,distributed systems,2,20.0
2,RPCs for Noobs,distributed systems,3,15.0
3,Xen and the Art of Surviving Undergraduate
School,undergraduate,7,25.0
4,Cooking for the Impatient Undergrad,undergraduate,6,30.0
```

When using this command, the following output appeared, as shown in the image below.

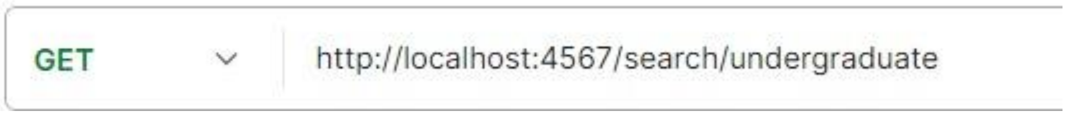


```
GET  http://localhost:4567/search/distributed systems
```

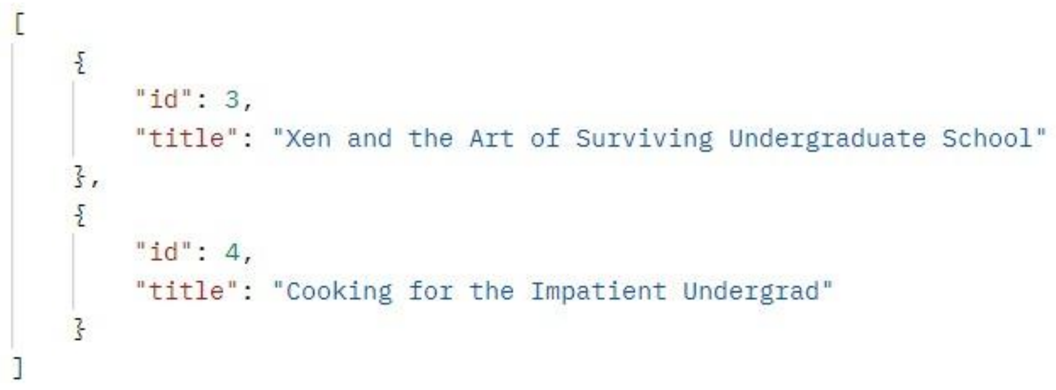
The output:-



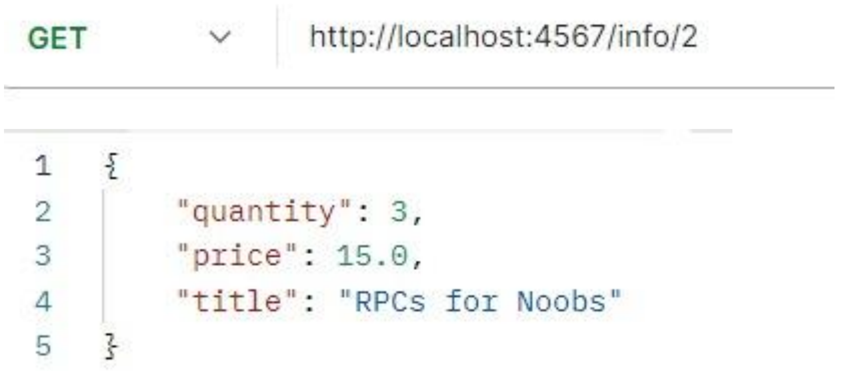
```
[
    {
        "id": 1,
        "title": "How to get a good grade in DOS in 40 minutes a day"
    },
    {
        "id": 2,
        "title": "RPCs for Noobs"
    }
]
```



```
GET  http://localhost:4567/search/undergraduate
```

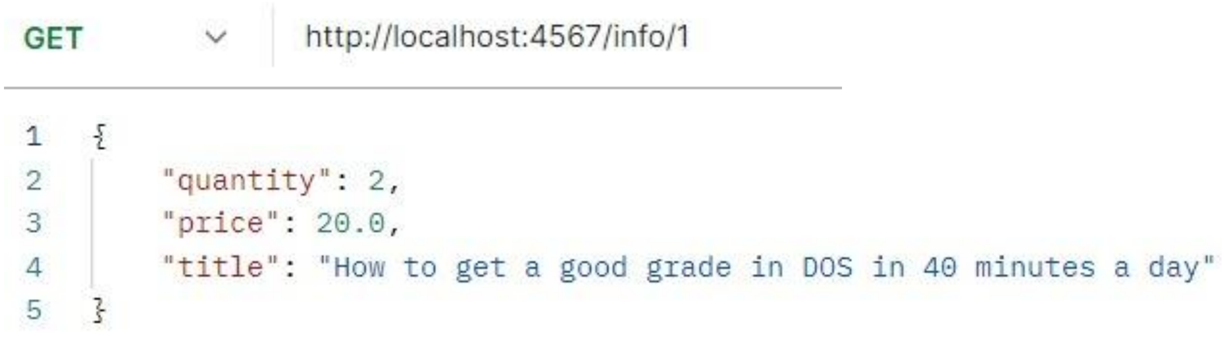


```
[
    {
        "id": 3,
        "title": "Xen and the Art of Surviving Undergraduate School"
    },
    {
        "id": 4,
        "title": "Cooking for the Impatient Undergrad"
    }
]
```



```
GET    http://localhost:4567/info/2
```

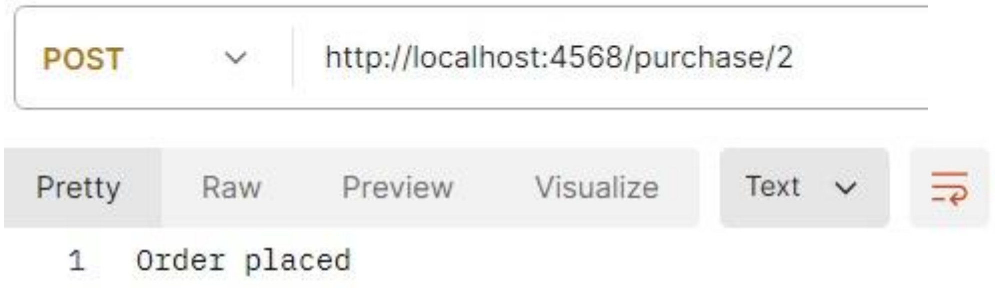
```
{
    "quantity": 3,
    "price": 15.0,
    "title": "RPCs for Noobs"
}
```



```
GET    http://localhost:4567/info/1
```

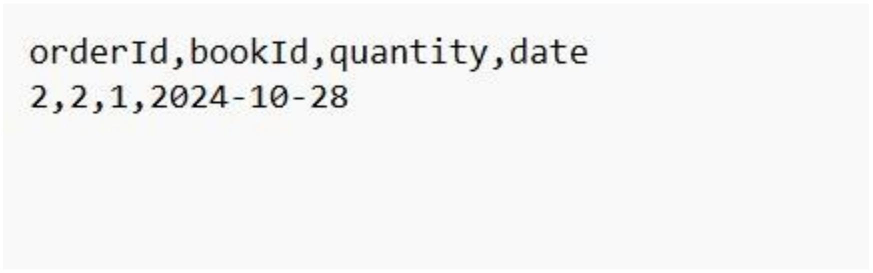
```
{
    "quantity": 2,
    "price": 20.0,
    "title": "How to get a good grade in DOS in 40 minutes a day"
}
```

After executing a purchase, the order file records that a book was bought. So, the order file will show an entry indicating that a book has been purchased



```
POST    http://localhost:4568/purchase/2
```

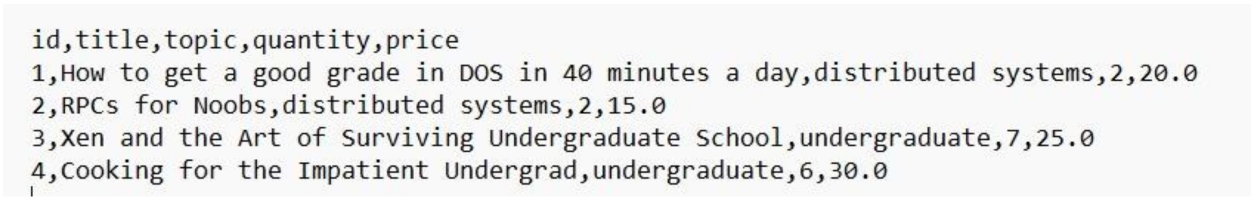
```
Order placed
```



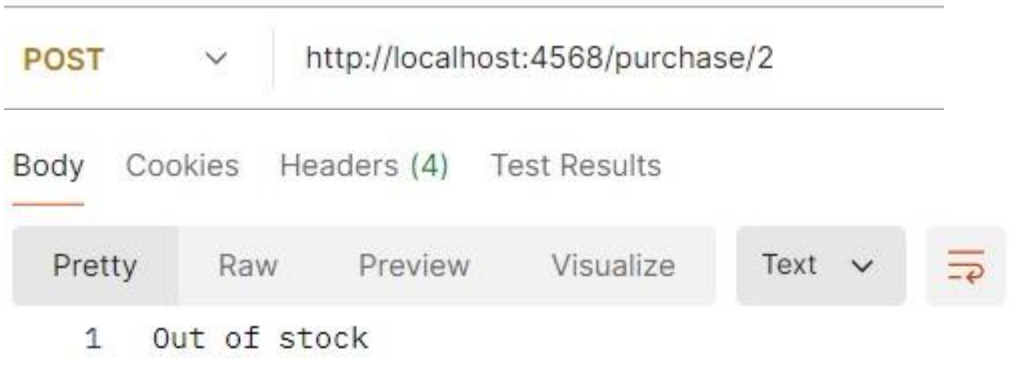
```
orderId,bookId,quantity,date
2,2,1,2024-10-28
```

In the image below: the quantity of the purchased book decreases in the catalog file.

Here, the quantity has decreased for the book with item number 2.



```
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,2,20.0
2,RPCs for Noobs,distributed systems,2,15.0
3,Xen and the Art of Surviving Undergraduate School,undergraduate,7,25.0
4,Cooking for the Impatient Undergrad,undergraduate,6,30.0
```



```
POST http://localhost:4568/purchase/2

Out of stock
```

If we execute a purchase operation and there is no stock left for the book (i.e., its quantity is zero), it will give an 'out of stock' message, indicating that it's unavailable.

Conclusion

Overall, the Bazar.com project highlights how useful a microservices approach can be for building organized and flexible applications. Breaking the app down into separate services—the frontend, catalog server, and order server—makes it easier to manage each part and allows us to handle user requests, inventory management, and order processing smoothly. With Docker, we've created a setup that's easy to deploy and ready for future growth. This project sets the stage for building more advanced distributed applications down the road, giving us a solid starting point to expand and improve.