Distributed Operating Systems Lab Report

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

Course: Distributed Operating Systems

Instructor: Samer Arandi

Student: Omar Raddad,Hala Atout

Date: October 28, 2024

Introduction

This lab assignment focuses on designing and implementing a simplified multi-tier online bookstore, **Bazar.com**, utilizing microservices and RESTful APIs. The project demonstrates core concepts of distributed systems, including microservices, RESTful communication, and data persistence. The bookstore, Bazar.com, carries only four books and operates on a two-tier architecture—front-end and back-end—with services designed to handle cataloging, ordering, and client interaction.

Problem Statement

The task was to develop a microservice-based online bookstore capable of processing three main operations for book transactions:

1. **Search by Topic** - allowing users to search for books by subject.
2. **Retrieve Book Information** - returning details on a book such as price and stock.
3. **Purchase** - enabling users to place an order, which updates stock information.

The bookstore comprises:

- **Front-End Service**: Accepts client requests and manages user interactions.
- **Catalog Service**: Manages book inventory details.
- **Order Service**: Processes purchase requests and updates stock.

Each component communicates through RESTful APIs and stores data persistently, using a lightweight storage solution.

Design and Implementation

Architecture

The application employs a **two-tier architecture** with separate microservices for each function:

1. **Front-End Microservice**: Acts as the main client interface.
2. **Catalog Microservice**: Provides catalog details through querying and updating stock and price.
3. **Order Microservice**: Handles book purchases, verifies stock, and modifies inventory records.

Each microservice is encapsulated within a Docker container, allowing for distributed deployment across multiple systems. Communication between microservices is managed via HTTP REST calls, supporting the following key endpoints:

- **/search/topic** - Queries the catalog by book topic.
- **/info/item_number** - Retrieves details for a specified book.
- **/purchase/item_number** - Processes a purchase order if stock is available.

RESTful API Implementation

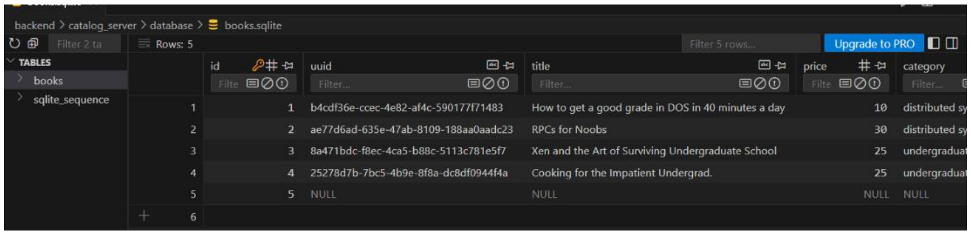
REST endpoints were implemented using [Framework, e.g., Flask for Python], with each API structured as follows:

- **GET /search/topic**: Returns books in the specified topic.
- **GET /info/item_number**: Provides detailed information on a specific book.
- **POST /purchase/item_number**: Executes a purchase, confirming stock availability and decrementing the quantity.

Data Persistence

For data storage, a simple text-based file or SQLite database was used:

- **Catalog Data**: Stores book information (title, stock quantity, price).
- **Order Data**: Records completed transactions for bookkeeping and inventory management.



	id	uuid	title	price	category
1	1	b4cdf36e-ccec-4e82-af4c-590177f71483	How to get a good grade in DOS in 40 minutes a day	10	distributed sy
2	2	ae77d6ad-635e-47ab-8109-188aa0aadc23	RPCs for Noobs	30	distributed sy
3	3	8a471bdc-f8ec-4ca5-b88c-5113c781e5f7	Xen and the Art of Surviving Undergraduate School	25	undergraduat
4	4	25278d7b-7bc5-4b9e-8f8a-dc8df0944f4a	Cooking for the Impatient Undergrad.	25	undergraduat
5	5	NULL	NULL	NULL	NULL
6					

The use of lightweight storage ensures minimal overhead while maintaining consistency across operations.

Testing and Evaluation

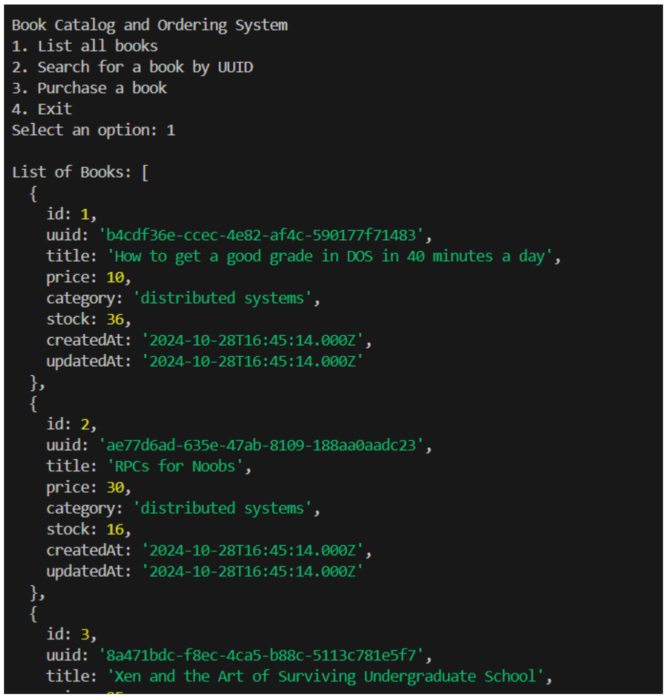
The system was deployed on three **Linux-based Docker containers**, each hosting one of the services. Tests were conducted on the following scenarios:

1. **Search Functionality**: Ensured accurate filtering by topic.
2. **Book Information Retrieval**: Confirmed that book details are correctly fetched based on item numbers.
3. **Purchase Process**: Verified stock check and update, handling out-of-stock scenarios with appropriate error responses.

Example Test Case Output:

- **Search Request**: Successful retrieval of books under "Distributed Systems."

- **Info Request**: Accurate book details displayed.



```
Book Catalog and Ordering System
1. List all books
2. Search for a book by UUID
3. Purchase a book
4. Exit
Select an option: 1

List of Books: [
  {
    id: 1,
    uuid: 'b4cdf36e-ccec-4e82-af4c-590177f71483',
    title: 'How to get a good grade in DOS in 40 minutes a day',
    price: 10,
    category: 'distributed systems',
    stock: 36,
    createdAt: '2024-10-28T16:45:14.000Z',
    updatedAt: '2024-10-28T16:45:14.000Z'
  },
  {
    id: 2,
    uuid: 'ae77d6ad-635e-47ab-8109-188aa0aadc23',
    title: 'RPCs for Noobs',
    price: 30,
    category: 'distributed systems',
    stock: 16,
    createdAt: '2024-10-28T16:45:14.000Z',
    updatedAt: '2024-10-28T16:45:14.000Z'
  },
  {
    id: 3,
    uuid: '8a471bdc-f8ec-4ca5-b88c-5113c781e5f7',
    title: 'Xen and the Art of Surviving Undergraduate School',
```

- **Purchase Request**: Stock decreases upon purchase; “Out of Stock” message appears when inventory is empty.

Design Trade-offs and Decisions

Trade-offs

- **Framework Choice**: A lightweight framework node express was chosen over more complex frameworks to prioritize simplicity and efficiency.
- **Data Storage**: SQLite or text files were selected for storage, balancing persistence needs with simplicity.

Conclusion

The Bazar.com project demonstrates a practical implementation of a multi-tier, microservice-based application with distributed system principles. Through RESTful APIs and persistent data handling, the bookstore fulfills all functional requirements while maintaining simplicity and modularity. The lab was an effective exercise in designing distributed applications, working with REST interfaces, and managing data across multiple services.
