HW DOS : Lab 1

Instructor's Name: Dr.Samer Arandi	**Semester: 2nd**
Student's Name : • Yaser Kharouf • Mohammed salha	**Registration Number:** **11924280** **12112617**
Date : 4/9/2025	**HW#2**

Lab Report: Bazar-Microservices Implementation

Objective

To design and implement a two-tier microservice-based web application called **Bazar.com** that simulates a bookstore system using Flask, Docker, and RESTful APIs.

1. Introduction

Implemented a three-tier microservices architecture for an online bookstore (Bazar.com) with:

1-Frontend service (Flask)

2-Catalog service (Flask + SQLite)

3-Order service (Flask + SQLite)

2. System Architecture

- Frontend Microservice – User interface and client API handler.
- Catalog Microservice – Handles book data (title, topic, quantity, price).
- Order Microservice – Handles purchase orders and logs.

```
graph TD
    A[Frontend:5000] -->|REST| B[Catalog:5001]
    A -->|REST| C[Order:5002]
    C -->|Query Stock| B
```

3. Services Implementation

3.1 Frontend Service

Responsibilities:

- Route user requests to backend services
- Aggregate responses
- API Gateway functionality

Key Endpoints:

Endpoint	Method	Description
`/search/<topic>`	GET	Search books by topic
`/info/<item_id>`	GET	Get book details
`/purchase/<item_id>`	POST	Purchase a book

3.2 Catalog Service

Database Schema:

```
CREATE TABLE books (
    id INTEGER PRIMARY KEY,
    title TEXT NOT NULL,
    topic TEXT NOT NULL,
    quantity INTEGER NOT NULL,
    price REAL NOT NULL
)
```

Sample Data:

```
[
    (1, "How to get a good grade in DOS...", "distributed systems", 100, 50.0),
    (2, "RPCs for Noobs", "distributed systems", 100, 30.0),
    ...
]
```

3.3 Order Service

Features:

- Purchase validation
- Inventory updates
- Order logging

4. Technical Challenges & Solutions

Challenge	Solution
Docker port conflicts	Standardized ports (5000, 5001, 5002)
SQLite file permissions	Volume mounts with proper ownership
Flask-Werkzeug version mismatch	Pinned versions in requirements.txt
Empty responses	Added health checks and proper error handling

5. API Documentation

Catalog Service:

```
GET /search/distributed%20systems
Response:
{
    "status": "success",
    "count": 2,
    "results": [
        {"id": 1, "title": "How to get..."},
        {"id": 2, "title": "RPCs for Noobs"}
    ]
}
```

Order Service:

```
POST /purchase/1
Response:
{
    "status": "success",
    "message": "Purchased: How to get..."
}
```

Catalog Service:

```
GET /search/distributed%20systems
Response:
```

```
{
    "status": "success",
    "count": 2,
    "results": [
        {"id": 1, "title": "How to get..."},
        {"id": 2, "title": "RPCs for Noobs"}
    ]
}
```

Order Service:

```
POST /purchase/1
Response:
{
    "status": "success",
    "message": "Purchased: How to get..."
}
```

6. Deployment

Docker Commands:

```
# Build and run
docker-compose up --build

# Test endpoints
curl http://localhost:5000/search/distributed%20systems
```

7. Testing

Test Cases:

1. Search by topic - verify correct books returned
2. Purchase flow - validate stock decrement
3. Error handling - test with invalid item IDs

8. Conclusion

Successfully implemented:

✓ Microservices architecture

✓ REST API communication

✓ Persistent data storage

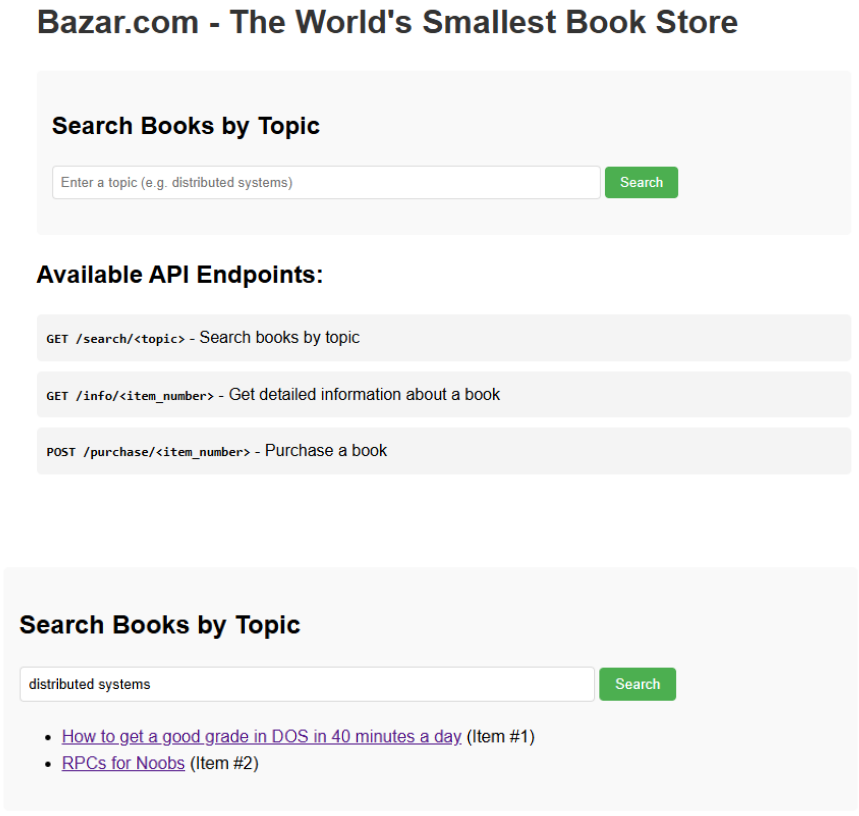
✓ Containerized deployment

Appendix

Project Structure:

```
bazar-microservices/
├── docker-compose.yml
├── frontend/
├── catalog/
└── order/
```

Project ui



Sample Output:

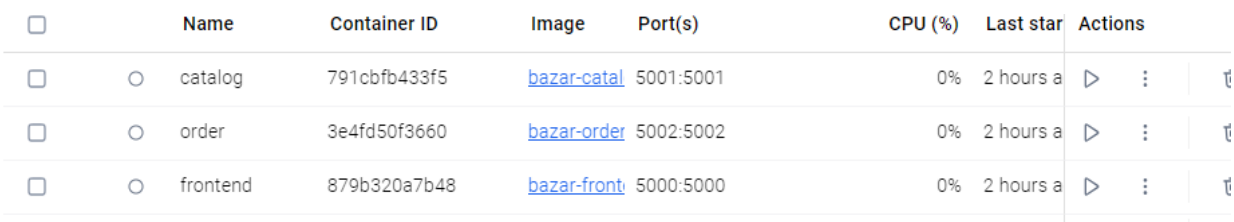
Pretty-print ☐


```
{
  "cost": 30,
  "item_number": 1,
  "stock": 9,
  "title": "How to get a good grade in DOS in 40 minutes a day",
  "topic": "distributed systems"
}
```



```
{
  "cost": 25,
  "item_number": 2,
  "stock": 15,
  "title": "RPCs for Noobs",
  "topic": "distributed systems"
}
```


Docker :



		Name	Container ID	Image	Port(s)	CPU (%)	Last star	Actions
☐	○	catalog	791cbfb433f5	bazar-catal	5001:5001	0%	2 hours a	▷ ⋮ 🗑
☐	○	order	3e4fd50f3660	bazar-order	5002:5002	0%	2 hours a	▷ ⋮ 🗑
☐	○	frontend	879b320a7b48	bazar-front	5000:5000	0%	2 hours a	▷ ⋮ 🗑