

HW DOS: Lab 1

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

Lab Report: Bazar-Microservices Implementation

1. Introduction

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

- Frontend service (Flask)
- Catalog service (Flask + SQLite)
- Order service (Flask + SQLite)

2. System Architecture

```
mermaid
Copy
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	Metho d	Description
/search/ <topic></topic>	GET	Search books by topic
/info/ <item_id></item_id>	GET	Get book details
/purchase/ <item_i d></item_i 	POST	Purchase a book

3.2 Catalog Service

Database Schema:

```
sql
Copy
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:

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:

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

6. Deployment

Docker Commands:

```
bash
Copy

# 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

Future Improvements:

- Add authentication
- Implement load balancing
- Add monitoring

Appendix

Project Structure:

```
Copy
bazar-microservices/

— docker-compose.yml

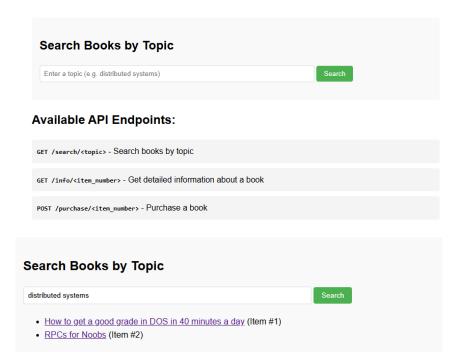
— frontend/
— catalog/

— order/
```

Sample Output:

```
json
Copy
{
    "status": "success",
    "data": {
       "title": "RPCs for Noobs",
       "quantity": 99,
       "price": 30.0
    }
}
```

Bazar.com - The World's Smallest Book Store



Available API Endpoints:

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"
}
```

	Name	Container ID	Image	Port(s)	CPU (%)	Last star	ar Actions		Actions		
0	catalog	791cbfb433f5	bazar-catal	5001:5001	0%	2 hours a	\triangleright	:	Ţ		
0	order	3e4fd50f3660	bazar-orde	5002:5002	0%	2 hours a	\triangleright	:	ť		
0	frontend	879b320a7b48	bazar-front	5000:5000	0%	2 hours a	\triangleright	:	Ţ		