Distributed & Operating Systems – Lab 1 Report

Bazar.com — A Multi-Tier Online Book Store

Student name: Buthaina Jallad

Student registration number: 12027915

Introduction

Introduction. The project implements Flask microservices to develop a multi-tiered online bookstore system. The system consists of three independent Docker containers which operate as separate services:

- Catalog Service.
- Order Service.
- Client Service (Web UI + CLI).

The system uses REST APIs to connect its services while storing data in SQLite databases.

System Architecture

The system design follows clean microservices architecture principles.

The Catalog Service maintains all book information and search functionality and price and quantity data and additional details.

The Order Service maintains control over purchase operations and maintains purchase records in its database.

The Client Service delivers access to both web-based and command-line interfaces to users.

The system uses Nginx as a reverse proxy to direct user requests toward specific microservices.

Catalog Service

The Catalog microservice provides the following REST endpoints:

• GET /search or /search/<topic>

Searches for books by topic and returns matching titles and IDs.

• GET /info/<id>

Returns book details including title, price, quantity, and topic.

• POST /decrement/<id>

Decreases stock when a purchase request is received from the Order service.

• POST /update/<id>

Updates price or quantity in the SQLite database.

The SQLite database contains four fields which are id and title and topic and price and quantity.

Order Service

Handles the purchase operation through the following workflow:

1. Receives a purchase request at:

POST /purchase/<id>

2. Contacts the Catalog service:

POST /decrement/<id>

to ensure the item exists and has available stock.

3. Records the successful purchase in an SQLite database.

4. Returns a JSON confirmation response:

{ "ok": true, "item_id": <id> }

This ensures the Catalog service controls inventory while the Order

service logs all transactions.

Client Service

The Client layer provides two user interfaces:

A) Web Interface (Flask + HTML + JavaScript)

• /api/search – Search books by topic

• /api/info/<id> – Fetch book info

• /api/buy/<id> – Trigger a purchase

B) CLI Interface (client_cli.py)

The system provides three main functions for users to perform operations.

1. Users can search for books through the system.

2. Users can access book information through the system.

3. Users can initiate book purchases through the system.

The CLI uses the same REST endpoints which the Client service provides for access.

Deployment with Docker & Nginx

The Dockerfiles for each microservice include:

The base image uses Python 3 as its foundation.

The application depends on Flask and its required libraries.

The service contains its application code.

The command starts Gunicorn for service operation.

The docker-compose.yml file manages all service operations by performing three main tasks.

The system creates three separate containers for catalog and order and client services.

The system enables database persistence through host directory mounting.

The system sets three environment variables which include:

o CATALOG_URL=http://catalog:5000

o ORDER_URL=http://order:5001

The system operates Nginx as an independent container which reads its routing configuration from nginx.conf.

All services within the system establish connections through the Docker network.

How to Run the System

You have to open docker first then

To build and start all containers:

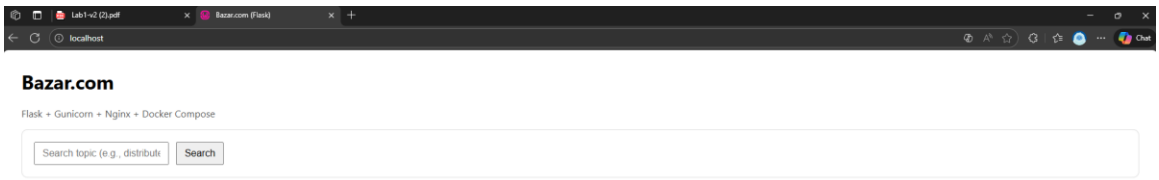
docker-compose up --build

To stop the system:

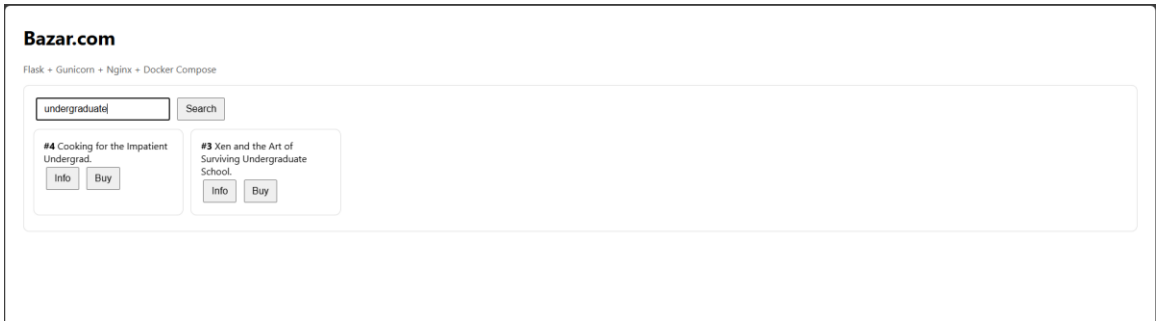
docker compose down

Web Interface available at:

http://localhost/



Sample output:



To run the CLI:

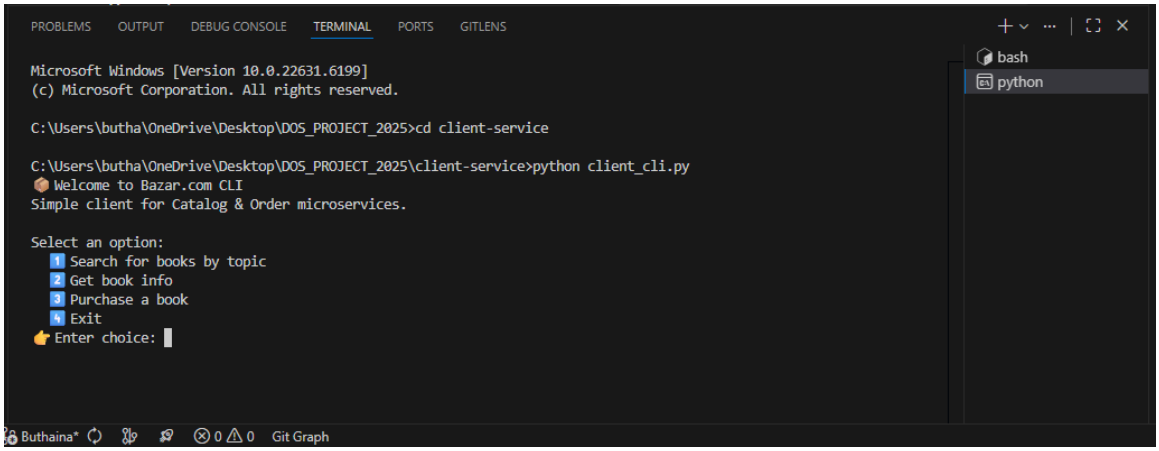
Open new terminal in visual studio (cmd terminal)

You have to enter client-service folder first

cd client-service

then write :

python3 client_cli.py



Sample output:

```
Select an option:
  1 Search for books by topic
  2 Get book info
  3 Purchase a book
  4 Exit
👉 Enter choice: 2

ℹ  Enter book ID: 1

📖 Book Info:
{
  "id": 1,
  "price": 19.99,
  "quantity": 3,
  "title": "How to get a good grade in DOS in 40 minutes a day. ",
  "topic": "distributed systems"
}

Select an option:
  1 Search for books by topic
  2 Get book info
  3 Purchase a book
  4 Exit
👉 Enter choice: 3

💳 Enter book ID to purchase: 1
✅ Bought book #1 successfully!
```

After purchasing serch info :

```
Select an option:
   1️⃣ Search for books by topic
   2️⃣ Get book info
   3️⃣ Purchase a book
   4️⃣ Exit
👉 Enter choice: 2

ℹ️  Enter book ID: 1

📖 Book Info:
{
  "id": 1,
  "price": 19.99,
  "quantity": 2,
  "title": "How to get a good grade in DOS in 40 minutes a day. ",
  "topic": "distributed systems"
}
```

Quantity decremented after purchasing

To run the cmd alone without python CLI:

Open new cmd in visual studio

Write curl commands

Sample outputs:

curl http://localhost/api/catalog/search

```
C:\Users\butha\OneDrive\Desktop\DOS_PROJECT_2025>docker compose down
time="2025-11-29T17:42:31+02:00" level=warning msg="C:\\Users\\butha\\OneDrive\\Desktop\\DOS_PROJECT_2025\\docker-com
pose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 5/5
 ✓ Container nginx_proxy            Remo...                 0.6s
 ✓ Container client                 Removed                 0.9s
 ✓ Container order                  Removed                 0.8s
 ✓ Container catalog                Removed                 0.7s
 ✓ Network dos_project_2025_default Removed                 0.4s

C:\Users\butha\OneDrive\Desktop\DOS_PROJECT_2025>curl http://localhost/api/catalog/search
{"items":{"Cooking for the Impatient Undergrad.":4,"How to get a good grade in DOS in 40 minutes a day.":1,"RPCs for
Noobs.":2,"Xen and the Art of Surviving Undergraduate School.":3}}

C:\Users\butha\OneDrive\Desktop\DOS_PROJECT_2025>
```

docker
cmd
cmd

You can only update quantity using curl commands:

curl -X POST http://localhost/api/catalog/update/2 -H "Content-Type: application/json" -d "{\"quantity\":3}"

Design Considerations

- Flask chosen for lightweight, modular microservices.
- SQLite selected for simplicity and OS independence.
- REST architecture ensures clear separation between components.
- Docker + Nginx provide portability and deployment consistency.

Design Tradeoffs

In designing the system, several architectural decisions were made:

1. Flask vs. Django: Flask is lightweight, minimal, and ideal for microservices. Django is heavy and unnecessary for a small REST-based lab.

2. SQLite vs. MySQL: SQLite is simple, file-based, no configuration needed.

3. REST vs. RPC: REST is stateless and easier to test (curl, browser).Widely used and simpler to route with Nginx. More scalable in distributed settings.

4. Microservices vs. Monolithic: Allows running each component independently.

Possible Improvements

1. The system requires the following future development work:
   The Catalog/Order service requires horizontal scaling through Nginx for better performance.

2. The Redis caching system needs optimization to achieve faster cache search operations.
3. The system needs an administrative dashboard to enable catalog modification functions.

Conclusion

The Bazar.com project demonstrates a complete microservice-based architecture using Flask, REST APIs, SQLite, Docker, and Nginx.