Emporium Platform – Project Report

Team: [Mohammed anwer salman/Qais Hweidei]

Project Overview

The **Emporium Platform** is our implementation of multi-tier online bookstore called **Bazar.com**, we built it using a microservices architecture. This platform is composed of three services:

- **Catalog Service** : manage book inventory
- **Order Service** : handle purchases and order logging
- **Gateway Service** : act as an API gateway routing client requests to backend services

Each service is containerized using Docker and can run independently across distributed environments.

Service Breakdown

1. Gateway Service

It routes user requests to the appropriate backend service (catalog or order), supports CORS, and includes a basic health check system.

- **Tech Stack**: Node.js, Express, Axios
- **Key Features**:
  - Centralized API routing
  - REST endpoints for searching, info retrieval, and purchases
  - Error handling & logging

- **Endpoints**:
    - `GET /health`
    - `GET /search/:topic`
    - `GET /info/:itemNumber`
    - `POST /purchase/:itemNumber`

2. Catalog Service

The **Catalog Service** manages book data including titles, stock, price, and topics. It supports CRUD-like functionality, allowing books to be searched, queried, and updated.

- **Tech Stack**: Python (Flask), Flask-CORS
- **Key Features**:
    - Book search by topic
    - Book details retrieval
    - Update price/quantity
    - Pre-loaded book data in JSON
- **Endpoints**:
    - `GET /health`
    - `GET /search/<topic>`
    - `GET /info/<item_number>`
    - `PUT /update/<item_number>`

3. Order Service

The **Order Service** handle purchase transaction and maintains an order log. It communicates with the Catalog Service to ensure stock availability before processing purchases.

- **Tech Stack**: Node.js, Express, Axios, fs (CSV for logging)
- **Key Features**:
  - Book purchases
  - Stock validation via Catalog
  - Transaction logging to `orders.csv`
- **Endpoints**:
  - `GET /health`
  - `POST /purchase/:itemId`

Deployment & Testing

All services are built using Docker Compose, with containers for:

- Gateway Service → exposed on port `3000`
- Catalog Service → exposed on port `5000`
- Order Service → exposed on port `4000`

To run the full platform:

```
docker-compose up --build
```

You can test API requests using:

- Postman
- Browser for GET requests

Key Design Decisions

- **Microservices & Docker**: Each service runs independently and can scale or fail without affecting others.
- **Simple Tech Stack**: Used tools like Flask, Express, and CSV/JSON instead of full-scale databases and frameworks.
- **REST over RPC**: Followed REST principles to expose all functionality via HTTP endpoints.

How to Run

Each service can be run independently or as part of the Docker Compose setup. For local development:

```
# Example for Catalog Service (Python)
python run.py

# Example for Gateway/Order Services (Node.js)
npm install
npm run dev
```

Or run all together:

```
docker-compose up --build
```