Emporium Platform – Lab 2 Report

Team: [Mohammed Anwer Salman / Qais Hweidei]

Project Overview

This lab extends our multi-tier bookstore platform *Bazar.com* by implementing **replication, fault tolerance, and optional caching**. We built this using a **microservices architecture** composed of three key services:

- **Catalog Service:** manages book inventory and supports primary-backup replication.
- **Order Service:** handles purchases, logs transactions, and supports replicated deployments.
- **Gateway Service:** acts as an API gateway with load balancing and optional LRU caching.

All services are containerized using Docker and run together via Docker Compose.

Service Breakdown

1. Gateway Service

The gateway routes client requests to catalog or order services. In Lab 2, it introduces **caching** and **load balancing**.

- **Tech Stack:** Node.js, Express, Axios
- **Key Features:**
    - Round-robin routing to multiple order/catalog replicas
    - In-memory LRU caching for read endpoints
    - Cache invalidation after write requests
    - Health monitoring and stats

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

---

2. Catalog Service

The catalog stores book data and supports replication via **primary-backup synchronization**.

- **Tech Stack:** Python (Flask), Flask-CORS
- **Key Features:**
    - Book search, info retrieval, and updates
    - Primary handles updates; backup syncs automatically
    - Supports cache invalidation via Gateway API
- **Endpoints:**
    - `GET /health`
    - `GET /search/<topic>`
    - `GET /info/<item_number>`
    - `PUT /update/<item_number>`

- `PUT /replica_sync/<item_number>` *(internal use)*

---

3. Order Service

The Order Service is enhanced in Lab 2 to support **replication** and **resilience**.

- **Tech Stack:** Node.js, Express, Axios, fs (CSV for logging)
- **Key Features:**
  - Purchase endpoint (`POST /purchase/:itemId`)
  - Validates stock with Catalog before purchase
  - Logs each transaction in `orders.csv`
  - Deploys two instances: `order-service-1` and `order-service-2`
  - Replica awareness using environment variable `REPLICA_URL`

Endpoints

- **`GET /health`**
- **`POST /purchase/:itemId`**
- **`POST /replicate`**

Deployment & Testing

All services are managed using Docker Compose. This allows easy orchestration and networking.

- **Gateway** → exposed on `localhost:3000`
- **Catalog Primary / Backup** → `localhost:5001`, `localhost:5002`
- **Order Replicas** → `localhost:4000`, `localhost:4001`

Testing Tools:

- Postman
- Docker logs / Docker Desktop for internal verification

Key Design Decisions

- **Microservices & Replication:** Services run in isolated containers with primary-backup designs (Catalog) and mutual replica awareness (Order).
- **Caching (Gateway):** Optional but implemented via LRU with TTL, stats, and invalidation.
- **Docker Compose:** Simplifies development and networking between services.
- **CSV Logging:** Transactions logged persistently without needing a full database.

How to Run

Each service can be run independently or together:

Gateway / Order

```
npm install
npm start
```

Catalog

```
python run.py
```

All Together

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