Bazar Bookstore 2 – Program Output

D. Samer Arandi

Rand Johari	12027653
Abeer Kharouf	12028125

1. System Overview

In this lab, we rearchitected the Bazar.com bookstore to support high concurrency, improve response latency, and ensure consistency across replicated microservices. We introduced key distributed system techniques such as:

- In-memory caching at the frontend
- Load balancing across replicated catalog and order servers
- Distributed mutual exclusion using logical clocks
- Cache invalidation protocols
- Synchronized database updates across replicas

---

2. How It Works

Architecture

The system is composed of:

- **Frontend Server** (localhost:3002 external, 3100 internal): Handles client requests, manages caching, and load balances between replicas.
- **Catalog Service** (2 replicas): Stores book metadata and stock. Supports updates and info/search queries.
- **Order Service** (2 replicas): Processes purchase requests using mutual exclusion.

- **Cache**: Integrated within the frontend for fast access to info and search results only.

All services communicate over REST APIs.

Load Balancing

Frontend performs **round-robin load balancing** per request using CATALOG_SERVICES and ORDER_SERVICES environment variables. It maintains local indices and routes each request to a different replica in cyclic order.

This ensures equal distribution of requests without relying on a single leader.

Caching

The frontend includes an **in-memory cache** with:

- **TTL (Time to Live)**: 1 minute
- **Size limit**: 4 items
- **Replacement policy**: Least Recently Used (LRU)

Cache is only used for GET /info/:id and GET /search/:topic. Admins can clear cache via:

POST /cache/clear

Header: x-admin-secret: secret123

Cache Consistency

Writes (buy/update) trigger cache invalidation:

- Before a catalog replica writes to disk, it sends a POST /invalidate to the frontend.
- Frontend removes the item from cache if present.

This guarantees strong consistency across reads and writes.

Mutual Exclusion for Writes

Order and catalog servers use **Ricart & Agrawala's algorithm**:

- Logical clocks determine request timestamps.
- All replicas exchange request-access and reply-access.
- Updates proceed only after all OKs received.

This allows fully distributed writes with no leader, maintaining load balancing.

---

3. Design Tradeoffs & Justifications

Decision	Justification
In-memory integrated cache	**Simpler and faster than external REST-based cache.**
Round-robin load balancing	**Lightweight and effective in absence of session affinity.**
CSV-based DB	**Simple for lab context; easy to inspect and synchronize.**
LRU + TTL	**Mimics real-world caching and allows testability.**

---

4. How to Run the System

Prerequisites

- Node.js
- Docker & Docker Compose

Steps:

From root directory

docker-compose up –build

Access frontend at: http://localhost:3002
Manual Cache Clear: curl -X POST http://localhost:3002/cache/clear -H "x-admin-secret: secret123"
Example Queries:
GET http://localhost:3002/info/2
POST http://localhost:3002/purchase/2

PUT http://localhost:3002/admin/update/2
Headers: x-admin-secret: secret123

---

5. Possible Improvements

Feature	Description
Persistent DB (e.g., SQLite/Postgres)	**To avoid reloading from CSV on restart.**
More advanced caching	**caching search results with invalidation rules.**
LRU + TTL	**Mimics real-world caching and allows testability.**