**Design Document: Bazar.com - Online Book Store lab 2**

**1. Introduction and Overview:**

Bazar.com is an online bookstore committed to providing customers with a seamless purchasing experience. This document highlights the changes made to the original design, incorporating a Last Recently Used (LRU) cache on the front and front replica, implementing round-robin replication, and creating replicas for each server and database.

**2. System Architecture:**

Bazar.com maintains a two-tier web architecture, featuring a front-end server and replicated microservices: the catalog server and the order server. Communication between components occurs through HTTP REST APIs.

**3. Functional Requirements:**

* Search for Books:
  + Cache: Front-end server utilizes LRU caching for faster topic-based searches.
  + Replication: Catalog servers are replicated, minimizing server load and enhancing search availability while reducing overhead.
* Detailed Book Information:
  + Cache: Front-end caches detailed book data, reducing latency for quicker access.
  + Replication: Replicated catalog servers ensure consistent and available book details, alleviating server congestion.
* Book Purchases:
  + Cache: Front-end checks cache for recent purchases, enhancing response time and reducing server load.
  + Replication: Replicated order servers manage purchases, ensuring reliability and reducing the burden on a single server.

**5. Component Design:**

* Front-end Server:
  + Handles user requests and communicates with the catalog and order servers.
  + Implements LRU caching for improved response time.
  + Replica Handling:
    - Multiple replicas of the front-end server can be deployed for load balancing and fault tolerance.
    - Implements a round-robin or similar algorithm for distributing incoming requests among replicas.
* Catalog Server:
  + Manages the book catalog, supporting query operations and providing book information.
  + Replica Handling:
    - Catalog server replicas ensure availability and distribute query requests.
    - Replicas synchronize data to maintain consistency across the system.
* Order Server:
  + Handles purchase requests, verifies book availability, and processes purchases.
  + Replica Handling:
    - Multiple order server replicas manage purchase requests for reliability and fault tolerance.
    - Replicas coordinate to maintain a consistent view of stock availability.

This updated component design introduces the concept of replicas for each server, providing redundancy, load balancing, and fault tolerance across the front-end, catalog, and order servers. The replicas work

**6. Implementation Plan:**

* Setup and Replica Initialization:
  + Initialize the project with basic components.
  + Set up replicas for the front-end, catalog, and order servers.
* Front-end Server with Cache and Replica Handling:
  + Integrate front-end server functionality, implementing LRU caching.
  + Enable round-robin load balancing for front-end server replicas.
  + Configure front-end replicas to interact with catalog and order server replicas.
  + When attempting to retrieve data at the front-end, check the cache first.
  + If the data exists in the cache, serve it directly; otherwise, proceed to the server (origin or replica) to fetch the data.
  + If using front-end replica, use catalog, order, and database replicas; otherwise, use the origin servers.

**7.Extentions:**

* Advanced Caching Strategies:

Expand caching capabilities by implementing multi-level caching.

Cache for catalog server and cache for order server too.

* + Benefits:
    - Significantly reduces query response times by serving frequently accessed data directly from the front-end caches.
    - Minimizes the load on the catalog and order servers, enhancing overall system performance.
    - Enables a more granular and efficient caching strategy tailored to the specific needs of catalog and order-related queries.

Front end

cacheee

purchase 

Search

Info

cacheee

Update 

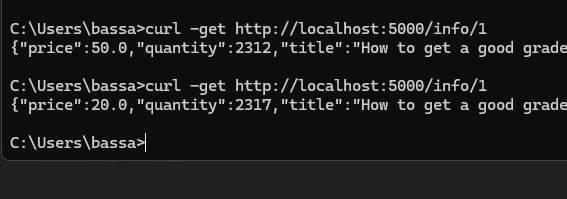
Catalog server

Order server

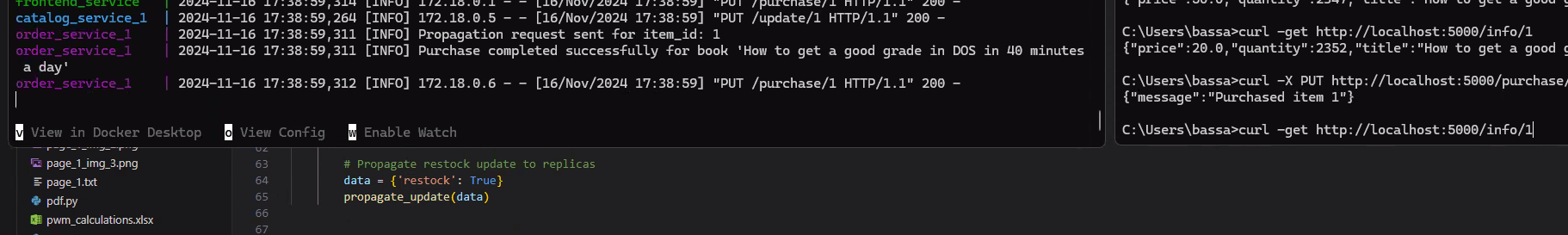
cacheee

**To initiate the application, utilize the `app.py` file as the front-end interface, facilitating user interaction. Prompt the user to select from options such as search, info, or purchase. Subsequently, the application seamlessly transitions to the back-end for the execution of the chosen operation.**

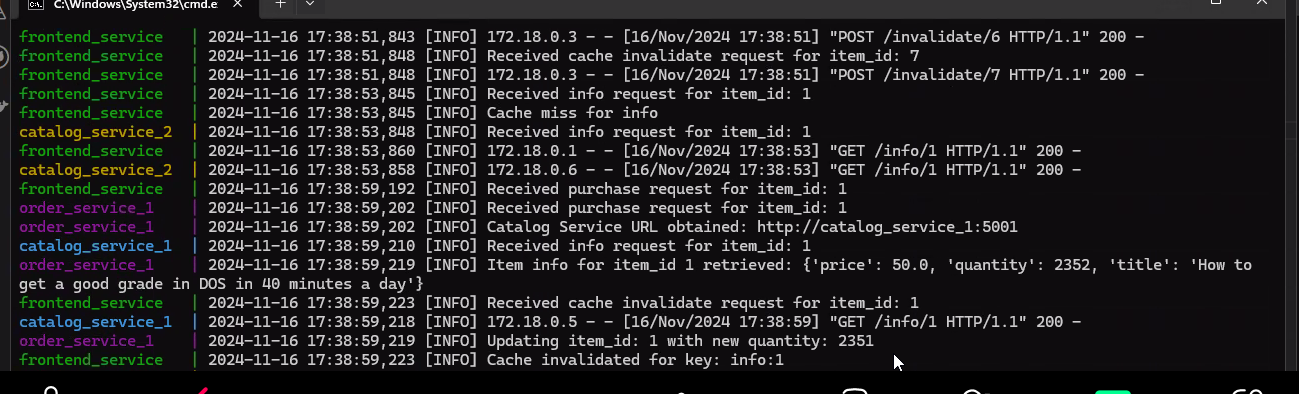
Number of book :



Number of book after seconds:



Cache invalidate:



Without cache:

