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

**1. Introduction and Overview:**

Bazar.com is an online bookstore focused on delivering a smooth, efficient purchasing experience. Key updates to the original system design include adding an LRU cache on the front-end and replica servers, implementing round-robin replication, and creating server and database replicas to enhance system reliability and performance.

**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\*\*: The front-end server uses LRU caching to speed up topic-based book searches, enabling quicker response times and a smoother user experience.

- \*\*Replication\*\*: Catalog servers are replicated to balance server load, improve search availability, and reduce system overhead.

* Detailed Book Information:

- \*\*Cache\*\*: The front-end caches detailed book information, reducing latency and allowing faster access to frequently viewed book details.

- \*\*Replication\*\*: Catalog server replicas provide consistent and reliable access to book details, helping to alleviate server congestion and ensure availability.

* Book Purchases:

- \*\*Cache\*\*: The front-end server checks the cache for recent purchase data, improving response times and lowering server demand.

- \*\*Replication\*\*: Replicated order servers handle purchase requests, ensuring reliability and distributing the load to prevent overburdening 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 essential components.

- Establish replicas for the front-end, catalog, and order servers.

- \*\*Front-end Server with Cache and Replica Handling\*\*:

- Integrate functionality into the front-end server, implementing LRU caching.

- Enable round-robin load balancing for the front-end server replicas.

- Configure front-end replicas to communicate with catalog and order server replicas.

- Check the cache first when retrieving data at the front-end.

- If the data is found in the cache, serve it directly; if not, request it from the server (origin or replica).

- When using a front-end replica, utilize catalog, order, and database replicas; otherwise, access 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.

**To start the application, use the `app.py` file as the front-end interface to enable user interaction. The user will be prompted to choose from options like search, information retrieval, or purchase. Once the user makes a selection, the application will smoothly transition to the back-end to execute the chosen operation.**