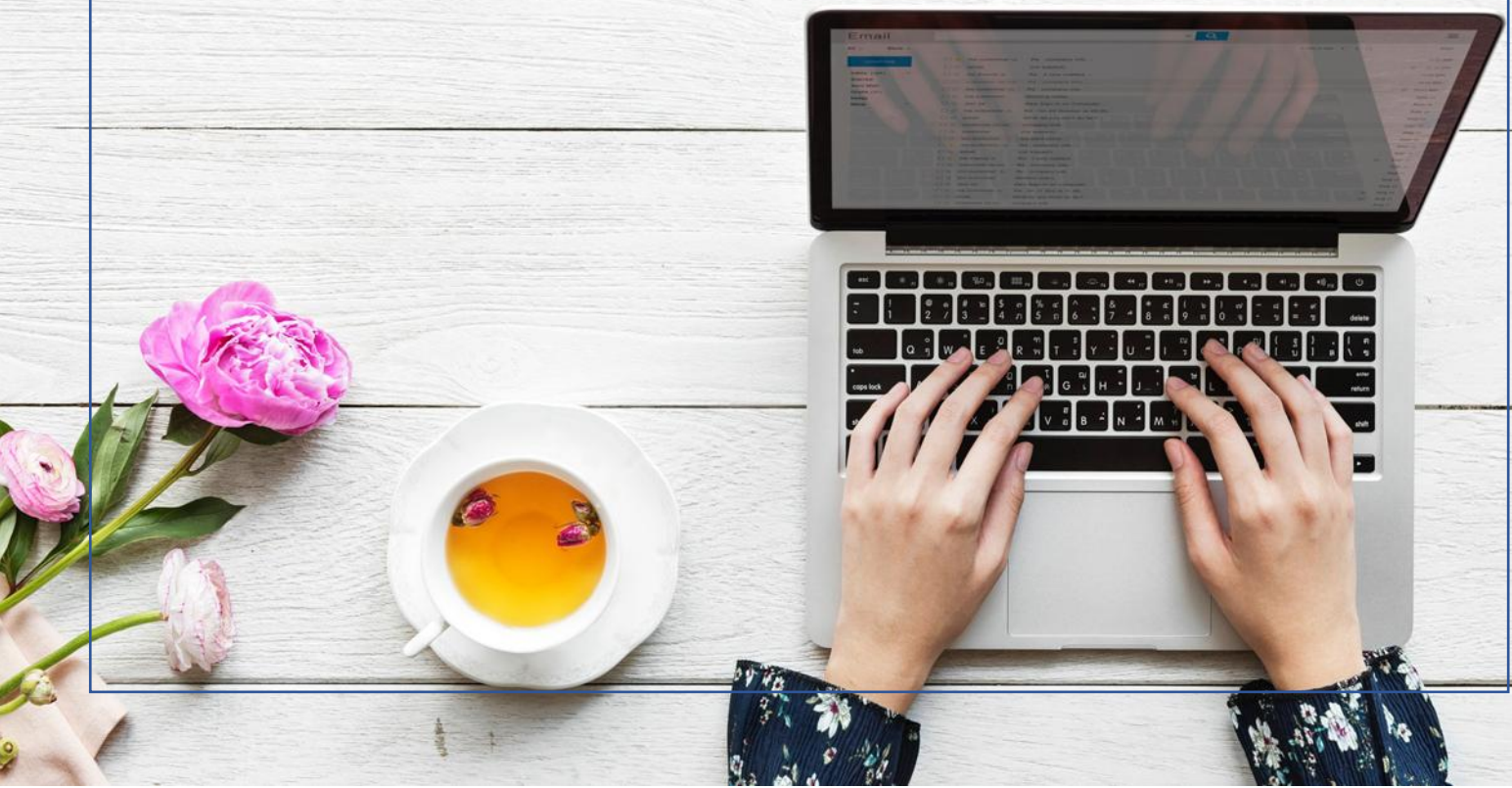
Bazar.com Project

Lina Qurom
11924435



Overall Program Design:

Bazar.com is designed as a two-tier web application employing microservices architecture. The front-end is implemented using Flask, a lightweight web framework for Python. The back-end consists of two microservices: the catalog server and the order server. The system uses a CSV file for data persistence, maintaining the catalog and order log. Docker containers are utilized to enable a distributed deployment of the three components.

Interactions Between Components

Front-end Microservice:

Accepts user requests for search, info, and purchase operations. Communicates with the catalog server for search and info operations. Communicates with the order server for purchase operations.

Catalog Microservice:

Maintains the catalog data using a CSV file. Supports query and update operations via REST endpoints. Handles requests from the front-end for search and info operations.

Order Microservice:

Manages the order log using a CSV file. Supports purchase operations via a REST endpoint. Verifies item availability with the catalog server before processing a purchase.

Design Tradeoffs

CSV for Data Persistence:

Tradeoff: Using CSV files for data persistence simplifies implementation but may have performance limitations compared to a more robust database solution.

Justification: Chosen for simplicity in the context of this lightweight application.

Flask as the Web Framework:

Tradeoff: Flask is lightweight, but it may lack some features offered by more extensive frameworks.

Justification: Chosen for its simplicity, ease of use, and suitability for microservices. Future considerations can involve transitioning to a more scalable database solution if needed.

Query Specificity:

Tradeoff: Offering limited catalog queries simplifies the system but may restrict user interactions; future adjustments might be needed for more complex queries.

Justification: Limiting catalog queries aligns with the current simplicity of Bazar.com. Future updates can enhance query capabilities based on evolving user needs.

Globalization and User Base Expansion

- **Language and Region Assumptions:**

Tradeoff: Presently assuming a single language and region may limit the user base; however, it expedites initial development.

Justification: Initial simplicity was chosen for rapid development. Future iterations should focus on internationalization to broaden the user base.

- **Currency and Regional Considerations:**

Tradeoff: Presently not supporting multiple currencies limits global reach; the initial simplicity was chosen.

Justification: Future updates can address this limitation by implementing support for multiple currencies and regional pricing to enhance the platform's global appeal.

Concurrency Handling

Flask provides built-in support for handling multiple concurrent requests. The web framework handles concurrency transparently through its underlying mechanisms, allowing the system to scale without manual intervention.

Data Persistence with CSV

Catalog Data: The catalog microservice reads and updates catalog information stored in a CSV file. Each modification operation on the catalog results in updating the CSV file.

Order Log: The order microservice maintains the order log using a CSV file. Purchase operations update the order log, with checks against the catalog data for item availability.

How It Works

Search Operation:

- User sends a search request to the front-end microservice.

- The front-end communicates with the catalog microservice using REST.
- The catalog microservice processes the query and returns matching entries.

Info Operation:

- User sends an info request to the front-end microservice with an item number.
- The front-end communicates with the catalog microservice using REST.
- The catalog microservice retrieves and returns details for the specified item.

Purchase Operation:

- User sends a purchase request to the front-end microservice with an item number.
- The front-end communicates with the order microservice using REST.
- The order microservice verifies item availability with the catalog and processes the purchase.

Communication between Microservices

Front-end to Catalog: REST calls for search and info operations.

Front-end to Order: REST call for purchase operations.

Known Issues

Data Integrity:

Concurrent updates to the catalog or order data may result in data integrity problems. Also, simultaneous updates from multiple clients may lead to race conditions.

Running the Program

Dependencies:

- Python
- Flask
- Docker with ubuntu

I have attached a file for generated output, explains how to run and use the system and the results for each case.