An-Najah National University
Faculty of Engineering and IT

جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا المعلومات

DOS-Project

Lab 1: Bazar.com

Supervised by Dr. Samer AL-Arandi

Presented by:

Ayham Thiab 12218365
Ayham Baarah 12218367

Introduction:

The goal of this lab is to design and implement a **small, yet realistic distributed online bookstore** called Bazar.com. The store sells four predefined books and supports three main operations from the client's point of view:

- **search(topic) – list all books under a given topic**
- **info(item_number) – show details about a specific book**
- **purchase(item_number) – buy a book if it is in stock**

The system must follow a **two-tier architecture** (front-end and back-end) and use **microservices** at each tier. The back-end is split into two services: a **catalog server** and an **order server**. All interactions between services are implemented as **RESTful HTTP endpoints**, and the data must be stored persistently on disk.

In addition to functional correctness, the lab emphasizes:

- **Service decomposition and microservice design**
- **RESTful APIs and JSON data exchange**
- **Handling concurrent requests**
- **Basic persistence using files or a lightweight database**
- **Deployment using Docker containers**

Overall Program Design

The system is composed of three independent microservices:

1. **Front-End Service**
   - Accepts client requests (search, info, purchase)
   - Communicates with the Catalog and Order services
   - Returns responses to the client as JSON
2. **Catalog Service**
   - Maintains the book catalog (id, title, topic, price, quantity)
   - Supports queries and updates on the catalog
3. **Order Service**
   - Processes purchase requests
   - Checks stock via the Catalog service
   - Decrements the quantity if a purchase succeeds
   - Logs each order for persistence/audit

Each service runs in its own Docker container and exposes an HTTP REST API. Communication between services is done using the container IP/hostnames and ports.

Detailed Description

The main functionalities provided by the Bazar.com system include the following endpoints:

- **Front-end Service Endpoints**
  - /search/ (**GET**): Forwards the request to the Catalog Service to find books matching the given topic.
  - /info/ (**GET**): Retrieves detailed information about a specific book by forwarding the request to the Catalog Service.
  - /purchase/ (**PUT**): Initiates a purchase by forwarding the request to the Order Service.
  - /orders (**GET**): Retrieves all orders by forwarding the request to the Order Service.
- **Catalog Service Endpoints**
  - /search/ (**GET**): Returns all books belonging to the specified topic.
  - /info/ (**GET**): Returns detailed information about a specific book, such as title, quantity, and price.
  - /update/ (**PUT**): Updates the quantity or price of a book. 1
- **Order Service Endpoints**
  - /purchase/ (**PUT**): Initiates a purchase for a specified book ID by verifying availability, updating the Catalog Service, and recording the order.
  - /orders (**GET**): Retrieves all orders that have been placed.

Design Trade-offs Considered and Made

- **Database Selection**: SQLite was chosen for its simplicity and suitability for small-scale applications like Bazar.com. Although it does not offer the scalability of more advanced databases (e.g., PostgreSQL or MySQL), it is sufficient for this project given the limited scope and the number of items managed.

- **Data Consistency**: Data consistency was prioritized when interacting between services to avoid race conditions. Thread safety mechanisms were implemented using Flask's built-in capabilities to handle concurrent requests.

- **Microservice Communication**: RESTful APIs were used for inter-service communication, providing a simple and stateless mechanism for data exchange.

Possible Improvements and Extensions

- **Database Scalability**: Transitioning from SQLite to a more robust database, such as PostgreSQL, would support larger datasets and concurrent access. Adding replication and sharding strategies could improve data availability and load balancing across multiple nodes.

- **Microservices Scalability**: Enable horizontal scaling by replicating microservices across multiple instances. Using a load balancer can distribute requests evenly, ensuring high availability and responsiveness under heavy loads.

- **Security and Rate Limiting**: Implement user authentication to secure endpoints and apply rate limiting to prevent abuse. This protects the system from excessive requests and enhances security.

- **Monitoring and Logging**: Centralized logging and real-time health monitoring of services would enable proactive issue detection and streamlined debugging, ensuring system reliability.

Instructions on How to Run the System

❖ **Prerequisites:** Install Docker and Docker Compose on your machine.

❖ **Setup Steps:**

- Ensure that the Docker Compose file (docker-compose.yml) is present in the project directory.

- Run the following command to start all services (Front-end, Catalog, and Order):

  docker-compose up

- Docker Compose will create and start the necessary containers for each service.

- Use a tool like Postman or curl to test the service endpoints. Example request to purchase an item:

  curl -X PUT http://localhost:5002/purchase/4

Conclusion:

The Bazar.com project demonstrates a simple implementation of a multi-tier online bookstore using microservices.

By dividing responsibilities among the Front-end, Catalog, and Order services, the system maintains modularity and separation of concerns.

The project meets the requirements of a small-scale bookstore, while future improvements can enhance scalability, security, and robustness.

Overall, the design and implementation of Bazar.com strike a balance between simplicity and functionality, making it suitable for learning and demonstrating basic distributed system principles.