An-Najah National University

Distributed Operation Systems
Dr. Samer Arandi
Online Book Store
Fall-Semester 2022

Student Name	Student ID
Essa Waleed Abbadi	11819883
Ahmad Hamad Qorom	11821492

Online Book Store

✓ Program Design

This design is constructing from three micro-services the front-end, catalog and order micro-services. The front-end will accept user requests and perform initial processing while the catalog server maintains the catalog which consists of these four books:

1-How to get a good grade in DOS in 40 minutes a day.
2-RPCs for Noobs.
3-Xen and the Art of Surviving Undergraduate School.
4-Cooking for the Impatient Undergrad.

For each entry, it maintains the number of items in stock, cost of the book and the topic of the book. All books belong to one of two topics: distributed systems and undergraduate school.

The order server maintains a list of all orders received for the books (it contains the id and timestamp).

The front-end server supports three operations:

• **search(topic)** - which allows the user to specify a topic and returns all entries belonging to that category.

• **info(item_number)** - which allows an item number to be specified and returns details such as number of items in stock and cost.

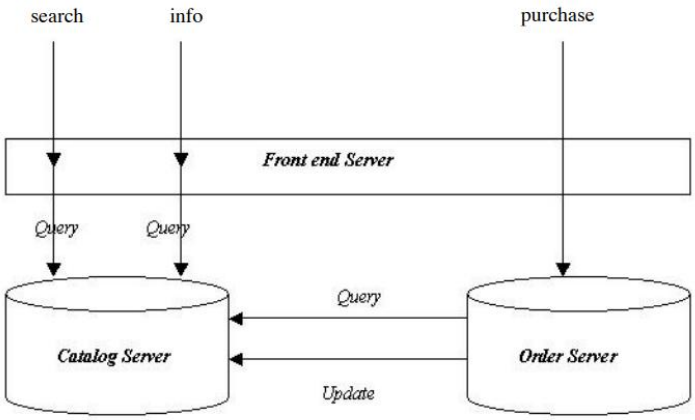
• **purchase(item_number)** - which specifies an item number for purchase.

The first two operations trigger queries on the catalog server. The purchase operations trigger a request to the order server.

The catalog server supports two operations: query and update. Two types of queries are supported: query-by-subject and query-by-item.

The update operation allows the cost of an item to be updated or the number of items in stock to be increased or decreased.

The order server supports a single operation: purchase(item_number). Upon receiving a purchase request, the order server must first verify that the item is in stock by querying the catalog server and then decrement the number of items in stock by one.



✓ How it works

it works in three different virtual machines or PCs as each service is separated from other services and has its own database, they interact with each other by sending http requests and received response in JSON formats.

The client uses the browser/postman to send http requests to the front end microservice and it will do the request by interacted with other microservices and replay with response to the client.

Design trade-offs

PHP is not good for code reusability and maintenance, also it has no Data Streaming and functional programming, moreover it has no JSON Inherent Payload support, so we don't used it.

Also speed of execution and the nature of the interpreter environment are often cited as the biggest potential drawbacks in **python**.

Why we used **java**?
Annotation syntax, which is easy to read. This feature makes Java Micro-services much easier to develop when powered by Micro-services frameworks. It offers more value in readability

Java includes many opinions to support developing & deploying Java Micro-services. It offers a user Interface, model components as well as connectivity to back-end resources, everything within the boundaries of a single, isolated and independently deployed apps.

In addition, many of Java EE standards are well suited for microservices applications like:

1- JAX-RS for APIs
2- JPA for data handling

CDI for dependency injection & lifecycle management in addition, service discovery solutions like Consul, Netflix Eureka or Amalgam8 are effortless in connecting with Java Microservices.

We used Spark Because, it is one of the best Java Microservices frameworks, supports creating web apps in Java 8,17 and Kotlin with less effort.

Possible Improvements and extensions

We can improve the design by make the microservices search for each other IP's dynamically without the need to put static IP's. Also we can do the same for port numbers.

We can add another catalog and order microservices that do the same job and interact with the same databases to sparate the http requests between them dynamicaly using the frontend microservices as a controller.

When it will not work

It will not work in these cases:

1- There is no connection between the microservices.
2- There are too many requests to the database at the same time.
3- The IP for the other microservices is wrong.
4- The path to the database is wrong.

Steps to run the program

First step: put the IP and port number for the catalog and orders microservice in the strings below in the "FrontMain.java".

```
public static String catalogIP_Port = "192.168.1.107:8077"; // ip and port for  catalog microservice
public static String orderIP_Port = "192.168.1.248:8088"; // ip and port for  order microservice
```

Also, but the catalog IP and the port number in "OrderMain.java"

```
public static String catalogIP_Port = "192.168.1.107:8077"; // ip and port for  catalog microservice
```

Then, you might create runnable jar files and run them in different machines, or use java IDE to run the microservices in different machines.

Finally, use browser or postman to send the http requests to the frontend microservice using the IP and port number for it in the URL.