Computer Engineering Department
Distributed Operating Systems (10636456)
Course Project (Bookstore)

Bazar.com

Bookstore project built in microservices manner

Instructor
Dr. Samer Arandi

Team Members

Name	Student ID
Amin Nasar	11715212
Hasan Abd Alhaq	11715239

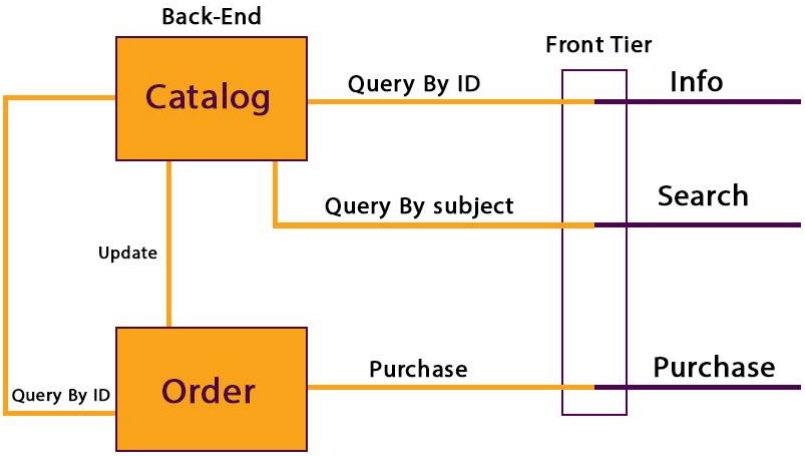
Program Design

This Store (Bazar.com), has been implemented in a microservices manner, in which each part or functionality of the program is considered a service, these are Catalog service (responsible for maintaining the list of available books with their prices and quantities left on the stock), and the Order service (user can order to purchase a book and also it has the record of the placed orders).
Catalog and order services have been split up into separate and independent servers, which together form the Back-End tier of this application.
Front-End application tier is also considered to be a microservice that is the "**Front-Tier**" that is responsible for accepting the direct user requests and handling them by forwarding them to the needed Back-End service (Catalog Or Order).

Each of these services has been implemented as a web RESTful service, using NodeJS runtime environment with the help of express framework, with some helper NPM packages, & for data storage, simple JSON files were used for each of the collections (orders, books).

In terms of code structure, monorepo (single repository) approach is used, where all the code responsible for implementing these services is saved in one repository, and running each service alone is done by a script that NPM (Node package manager) perform to create a process of single Javascript file as a process (Service).

How Does It Work



As illustrated previously, Bazar.com works by providing a front interface that accepts 3 types of operations, **Info** which is trying to get information about a certain book by its ID, **Search** which is the attempt to grab all books that belong to a certain category/subject, and finally, **Purchase** by which a certain book should be checked to be available and if so, bought, and its quantity will decrease by one.
Each of these operations can be handled by a single Back-End service (Catalog or Order) and the Front Tier's job is to perform initial processing and forward each of these functionalities for the responsible back-end service.

Design Tradeoffs Considered & Made

1. In monorepo (single repository) approach, all the implementation code is maintained in one place (repo) which by itself has advantages of:
    a. Having single configuration files and environment variables files (even if multiple files are used, they're written and stored once).
    b. Easier dependency management where every service's dependencies are stored in one place (node-modules) so that they're not replicated.

    However, the choice of monorepo comes with a cost of Not being able to pull a certain piece of code, so each server will have the whole project code instead of the code of that particular server. Also, other disadvantages can occur if this project extended and became large scale (like increasing build time & hard git commands and code traversal).

2. Using virtualization software (like VMWare, VirtualBox) would use the idea of Guest OS's which by itself would be heavy to run and pull off in terms of the machine's physical resources.
    Docker was used to build images and run them on separate containers that run natively on the machine and only carries the application with its dependencies which is a much lighter solution to use.
    Docker's problem is that persistent data storage is complicated & the concept of **Docker Data Volumes** should be used to solve this problem, because restarting a container would clear any changes on the data files.

Possible Improvements And Extensions

1. Having a duplicate code regarding dealing with the storage JSON files is a problem for sure, for simplicity, functions of reading from and writing to any file is duplicated between Catalog and Order services. Good improvement for that would be a file server that handles dealing with any file that stores data (store.JSON, orders.JSON).

2. Using Docker Data Volumes will handle the problem of data being volatile between container's restarts. Using docker named volume would enable the host system to link its file system to the virtual file system of the docker's container.

3. Possible extension is to use the JOI npm package that will enhance the data validation of any book update request, which has body parameters (quantity, price) that should be ideally validated to be actual numbers with valid values.

4. Local IP address of the server **192.168.1.117** is being determined through .env file as an environment variable, but possible improvement would be to use OS node core module in order to get the local IP address and work on it dynamically.

Why Wouldn't The Application Run In Some Cases ?

This application may not work correctly in some end cases, in which the program will either work incorrectly or even not work at all, these cases are:

1. Not having **linux** operating system as a primary OS or even as a **WSL** (Windows subsystem for linux). In this case, Hyper-v should be installed and run on the machine in order to make Docker work, if none of these options is fulfilled, this docker engine won't even launch, so images won't be able to build, and containers won't be able to run.

2. Because the main IP address for the hole server that serves these containers is provided through an environment variable, not providing the correct **IP address** (local address on current LAN) would take the IP value as undefined, or even as a wrong IP value that will make the application run on a device that we cannot even connect to.

3. Not Building the base image, would make a problem because the base image is not hosted on the cloud, so it must be operating locally for other images to build successfully.

4. Running the container individually (Not Using docker-compose) would make each service listen to a predefined PORT which may interfere with any other application running on the machine, So .env file should be overwritten to make sure that environment variables are set correctly.

Note: The last case is just a case in which the application "**MAY**" not work correctly, but in some cases where no other processes are running on the predefined PORTs, it will work perfectly fine.

How To Run This Application

1. Clone the github repository. (Make sure Git is installed on the system).
   Through SSH

   ```
   git clone git@github.com:amin-nassar/dos-bookstore-project.git
   ```

   OR HTTP

   ```
   git clone https://github.com/amin-nassar/dos-bookstore-project.git
   ```

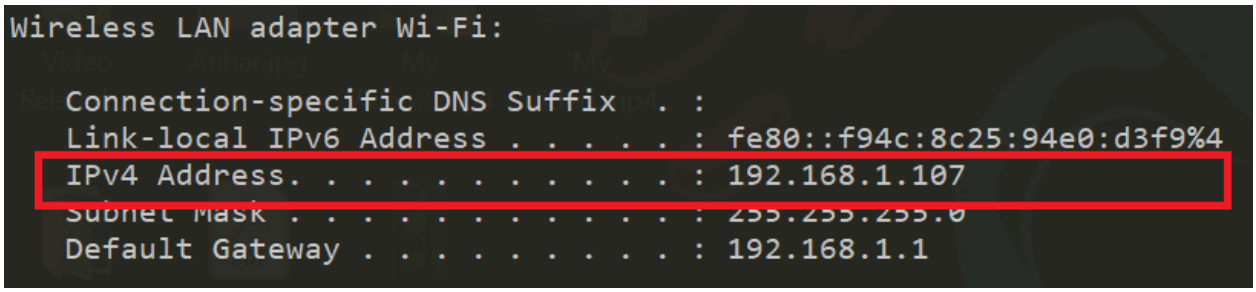
2. Run npm install to install all dependencies (optional).

   ```
   npm install
   ```

   Running this command would be important in case the code will be directly run on the machine, but for the docker images everything is handled through dockerfiles.

3. Get the local IP address.

```
ipconfig
```



4. Set environment variables in the .env file including the MAIN_IP variable with the previously obtained address (recommended).

```
CATALOG_PORT=4000
ORDER_PORT=3000
FRONT_PORT=2000
MAIN_IP=192.168.1.107
```

5. Build the base image (Naming Is Important).

```
docker build -t bookstore:latest .
```

Naming the image with a different name, will make it mandatory to change the image name in each file of (Dockerfile.catalog, Dockerfile.order, Dockerfile.server) because as previously explained, this image is not hosted on the cloud so it need to get built locally with the same name used in these files.

6. Build all images & run them on their containers (Individual running in not recommended)

```
docker-compose up
```

Final Result