Bazar.com Library

(Fall 2021/2022)

DOS Project

Dr. Samer Arandi

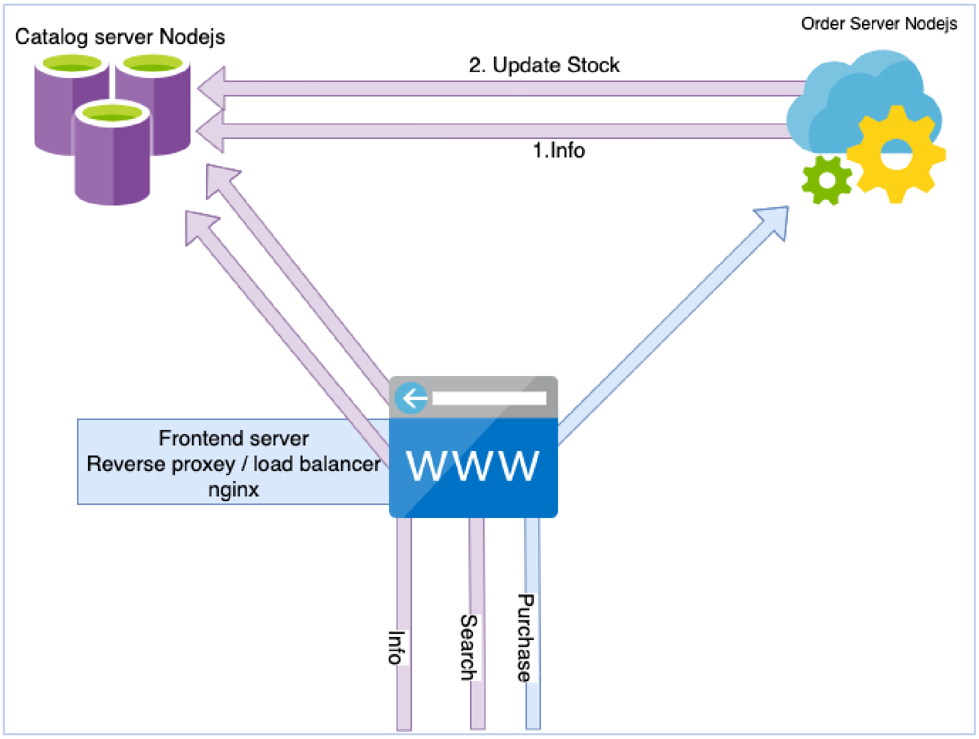
ABDELRAHMAN BABA
a.babaa@stu.najah.edu

NABEL KAYED
nabeelkayed25@gmail.com

Main requirements:

1. *the app should provide three main services*
   - A. *Info: which shows all stored information about a specific book.*
   - B. *Search: filter book based on some criteria, currently topic only*
   - C. *Purchase: be able to buy books*
2. *For purchase we need to make sure the book is in stock first.*
3. *The app should be distributed among multiple servers*

Basic architecture view:



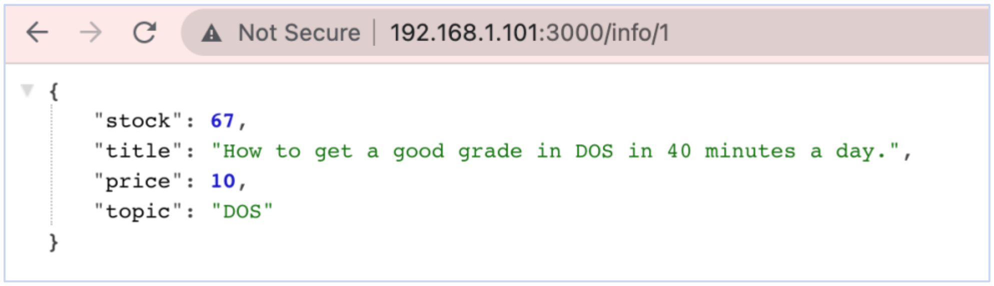
Architecture description:

A. *Catalog server:*

- This server stores all the information about every book in the library.
- Technologies used in this server: Node JS , SQLite.
- It provides a REST interface that allows the following operations:

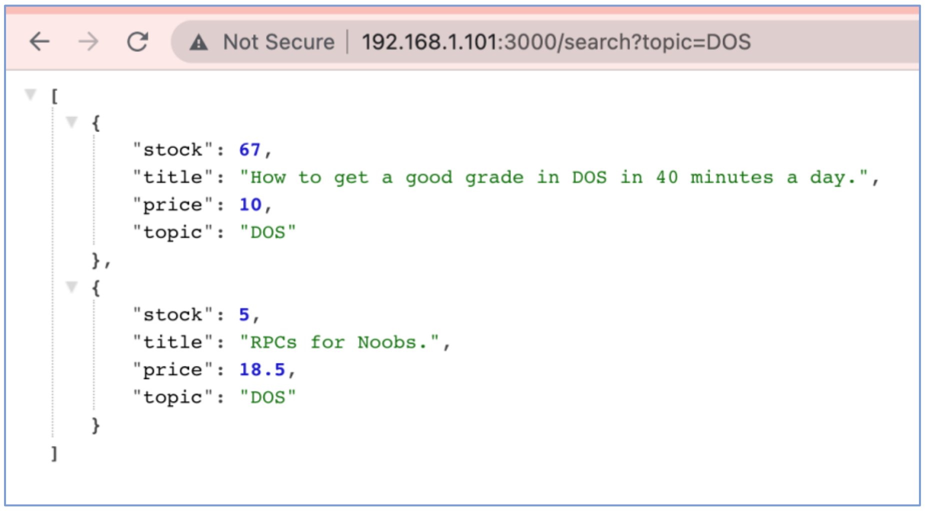
Info: *GET : URL/info/bookId*

All information about a book.


Not Secure | 192.168.1.101:3000/info/1

```
{
    "stock": 67,
    "title": "How to get a good grade in DOS in 40 minutes a day.",
    "price": 10,
    "topic": "DOS"
}
```

Search: *GET: URL/search?topic=VALUE*

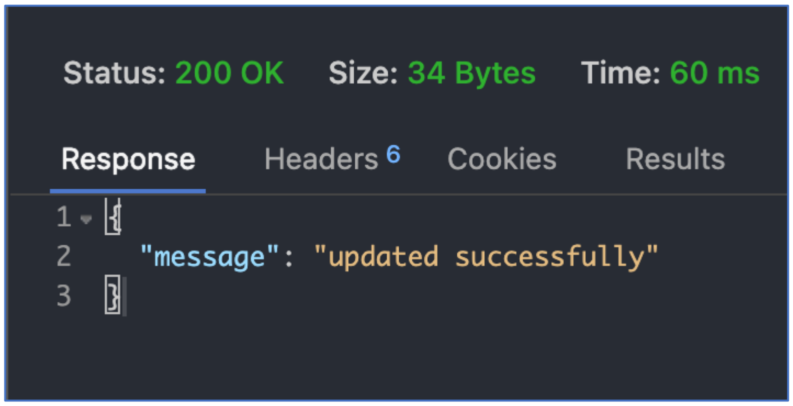


192.168.1.101:3000/search?topic=DOS

```
[
  {
    "stock": 67,
    "title": "How to get a good grade in DOS in 40 minutes a day.",
    "price": 10,
    "topic": "DOS"
  },
  {
    "stock": 5,
    "title": "RPCs for Noobs.",
    "price": 18.5,
    "topic": "DOS"
  }
]
```

Outcome: List of books or empty array.

Update: *PUT URL/book/:id?Stock=[Positive or negative offset]&price=[newPrice]*



Status: 200 OK Size: 34 Bytes Time: 60 ms

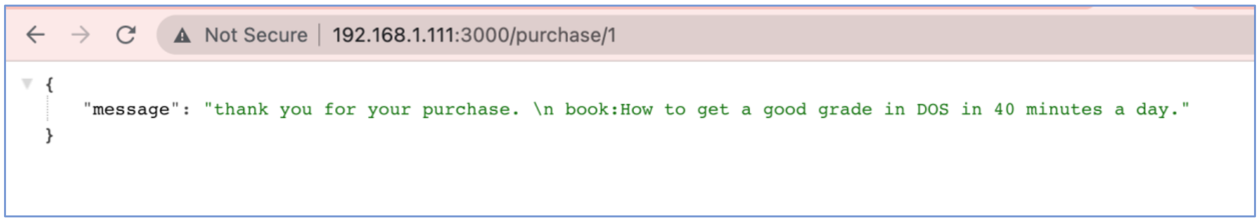
```
{
  "message": "updated successfully"
}
```

Outcome: Success or book not found or stock < 0

B. Order Server:

Has single endpoint (purchase) which so the following:

Get URL/purchase/:id



192.168.1.111:3000/purchase/1

```
{
  "message": "thank you for your purchase. \n book:How to get a good grade in DOS in 40 minutes a day."
}
```

For purchase to work properly it has to contact Catalog server internally:

- Query the book to see if its available and in stock
- If available update stock

Outcome: Success or book not found or stock < 0

3. Frontend server:

The front end server works as a reverse proxy, it forwards incoming request to corresponding servers to processes them and return responses.

Design Tradeoffs:

Using microservices architecture provided the following benefits:

- *Vertical distribution.*
- *Makes horizontal scalability much easier as it can be done at component level.*
- *Increases modularity which is very beneficial* [1] *.*
- *Load balancing*

But it comes with the following disadvantages:

- *High communications between server overhead.*
- *The need to setup and maintain three different servers.*

Future improvements:

The following improvements can be made to improve performance and scalability:

- *Move database instance to a separate server.*
- *Lock update endpoints so that it works locally*
- *Add multiple horizontal instances of catalog server to handle the heavy loads.*

Scenarios where the app may not work correctly:

The app may cause irreversible damage to the stored data if to purchase requests happen to occur at the same time for the same book, when the order server fetches the current stock, this is caused because the purchase operation happens in two independent stages, a solution to this problem may be to make the operation atomic.

Appendix:

- *Github repo for catalog server:* https://github.com/Abdelrahmanba/dos-catalog
- *Github repo for order server:* https://github.com/Abdelrahmanba/dos-order

[1] The Advantages of Modular Software and Programming