**How does it work?**

In the beginning, I tried to make a simple GUI that is easy to use, and it contains every functionality required.

First, in the home page the user can see all books in the store, this is done when the home page loads, a GET request is sent to the catalog server to fetch all books in the store, the user then can click on any of the books to view more info about that book, however this increases the loading time of the home page.

Second, in the info page, info of the item requested by the user is retrieved, this is done through sending a GET request to the catalog server with the ID of the book that the user want to show its info, the user can see all info about the book and can purchase that book, if the user decides to purchase that book he can click on purchase to send a POST request to the order server to do 2 things: sends a GET request to check if the book is out of stock or not, then create a record for the purchased book. After creating the record, the order server sends a PUT request to the catalog server informing it that a purchase happened, and you need to update the quantity of that book by decrementing it by 1.

Third, in the search part, the user can search for a book by either its ID or its topic, the user can then view info about books from results, this is done by sending a GET request to the catalog server with search info.

However, if the store had so many books, then fetching and displaying all books in the store wouldn’t be a good thing to do, this can be solved by returning constant number of books or returning most popular books in the store, also, during the purchase operation, if the catalog server fails or drops before the order server informs it of a purchase then the stock will not be updated even though a purchase was recorded. This can be solved by some event handling.

**How to run the program:**

To run the application, the 3 parts must be deployed to 3 different machines and each one is given a static IP address, in my case, I had 2 virtual machines and the host machine to deploy parts of the application on.

Host machine had windows 10 (education edition) operating system and both virtual machines had ubuntu (version 18.04.1), for virtual machines management VMBox was used.

To make the connection between the machines easier and give each one static IP address, machines were connected to the same LAN, to connect a virtual machine to the LAN, go to virtual machine settings in VMBox, network tab on the left, and change attached to option to bridged adapter, this option allows the virtual machine to acquire its own IP address from the DHCP server and make it access the internet.

After giving each machine a static IP address, host machine won’t be able to communicate properly with the virtual machines, to work around this issue:

Go to windows firewall->advanced settings, a window with advanced settings appears, on the left select inbound rules and then on the right click on new rule, a wizard appears, select custom as rule type, then allow all programs, then allow any protocol type, then enter the host machine’s IP address as local IP address and enter virtual machines IP addresses as remote IP addresses, then give the rule a name and finish.

Regarding the implementation of the application, lumen framework was used, lumen requires composer to instantiate a web app. Composer is a dependency manager for PHP, thus, it requires PHP to be able to install it, more info on how to install it is in their website[[1]](#footnote-1).

After installing composer, you must download dependencies by typing **composer install** in terminal**.**

After downloading dependencies, the machine is ready to serve the application, it can be done by typing **php -S 192.168.1.21:8000 -t public**, this IP address and port number were in my case, you can put any port number you like that is not used by any other application.

**Documentation:**

1)Frontend service, BooksController.

1. Index

This function retrieves all books in the store from catalog server and returns a view with information from response.

1. Show

Searches catalog server for a specific item, returns a view with information from response.

1. Info

Retrieves all information about a specific item from catalog server, returns a view with information from response.

1. Purchase

Decrements quantity of a specific item by 1 in case of purchase, returns a view with the status of purchase.

2)Catalog service, CatalogController.

1. Index

Handles the request of fetching all books, returns all books in the store

1. Show

Handles search requests, returns search results of given topic (or ID) and the search method.

1. Info

Handles info requests, returns all information about a specific book.

1. Purchase

Handles updating of stock after a purchase, returns nothing.

1. Write

This function is used to initialize the CSV file of the store, redirects to read.

1. Read

This function is used for testing purposes only, returns data in the CSV file.

3)Order service, PurchaseController.

1. Purchase

Handles purchase requests, creates a record for the purchase and informs catalog service of that purchase, returns status of the purchase (whether it was successful or not).

1. Read

used for testing purposes only, returns everything in the purchases CSV file.

1. https://getcomposer.org/ [↑](#footnote-ref-1)