Distributed Operating System

HW.2: Microwebservices

Student Name: Asem Jamil Ishtayah

Student ID: 11316990

Supervisor: Dr. Samer Arandi

Introduction:

On this project, or we might call it a lab, we will deal with some frameworks, database, virtual machines to operate gusts operating systems …etc. we will learn to deal with micro-webservices and make requests to these services and handle the responses to build a small distributed system project.

Requirements:

1. VMware: to operate gusts operating systems that will operate servers separately.
2. Gusts OS: ubuntu-18.04.3.
3. Micro-web framework: We use FLASK for Python language.
4. Database: We use SQLite3.
5. Text editor: Visual Studio Code.
6. Github account: https://github.com/AsemIsht/DOS_HW-ASEM

Design:

The project consists of three severs each server is installed on Flask framework for Python language that consist of micro-webservices, each server has separate purpose and works on separate gust operating system on VMware for virtualization, and these servers are:

1. Frontend Sever.
2. Order Server.
3. Catalo Server.

The client will request one of three micro-webservices to Frontend server:

1. search(topic).
   `http://Frontend-server/search/<topic>`
2. lookup(item_number).
   `http://Frontend-server/lookup/<item_number>`
3. buy(item_number).
   `http://Frontend-server/buy/<item_number>`

Note that the **Database** is controlled by catalog server only. And if any of other servers need any data from Database will request from Catalog. As well as requests for update and modification any data.

Catalog Server will response to frontend and order servers, therefor it has these micro-webservices:

1. Query_by_topic(topic):
   `http://Catalog-server/query_by_topic/<topic>`
2. Query_by_item_number(item_number):
   `http://Catalog-server/query_by_ item_number/<item_number>`
3. Update(item_number):
   `http://Catalog-server/update/<item_number>`

Order Server will response only to frontend server, therefor it has this micro-webservices:

buy(item_number):

`http://Order-server/buy/<item_number>`

Sequence of operating:

`Search for a topic/item category(search, lookup):` client requests a **GET** request to frontend server which handle this request and make a **GET** request from Catalog server which make a query to get all items that categorized on this topic to response to Frontend server. When Frontend server get the data will return it to client application with nicely formatted.

Host requests from frontend server ➔ frontend server request the data from Catalog server ➔ Catalog server check this query and return the results to frontend server which in turn will return the information in a nicely formatted to Client application.

Buy an item (Buy): Client requests a **GET** request to Frontend server, Frontend handle the request and requests a **POST** request to order server which make some processes. Order server request a **GET** request to catalog server to get the quantity of this book which wanted from client. After the order server get the quantity and if it was positive number will request a **PUT** request to catalog server to update the quantity (decrement 1 then send request with new_value as parameter). When Catalog response positively to Order server, Order server will response to frontend server and inform it if the buy was done successfully. Then frontend server response to client application.

How to run the program:

I will describe how to run any server on any machine with ubuntu-18.04.3 OS.

Each server has source code file that has extend of (.py) indicate to Python text. This file includes all of necessary code for the micro-webservices related to the specific server.

* Open Terminal, and insert:

```
$ python3 -V            ➔ output: Python 3.6.6 # Python3 Version
```

install a virtual environment: insert:

```
$ sudo apt install python3-venv
```

Change the directory to work directory (server directory where python code exists) by insert `cd` command.

run the following command to create your new virtual environment:

```
$ python3 -m venv venv → #not needed because the venv directory is existed.
```

The command above creates a directory called `venv`, which contains a copy of the Python binary

To start using this virtual environment, you need to activate it by running the `activiate` script:

```
$ source venv/bin/activate
```

Installing Flask:

```
(venv) $ sudo apt install python-pip ➔ # at first install pip
```

```
(venv) $ pip install Flask
```

```
(venv) $ pip sudo apt install python3-flask
```

Installing requests library:

```
(venv) $ pip install requests
```

Run the hello.py code:

```
(venv) $ export FLASK_APP=hello.py
```

```
(venv) $ export FLASK_ENV=development     # mode as debugger (developing)
```

```
(venv) $ flask run –host=0.0.0.0    # run the server on the network to be
                                    accessed from any machine on the network
```

To be more honest, what I wrote before is sufficient to run the project, but I want to note that I found the best way to run the project is to delete the `venv` directory (delete manually), and reinstall it again on your machine. (yellow line on the page above).

Output generated:

Running the servers:

Frontend-Server

```
asem3@asem3: ~/Desktop/DOS_HW ASEM/FrontendServer
File  Edit  View  Search  Terminal  Help
asem3@asem3:~/Desktop/DOS_HW ASEM/FrontendServer$ source venv/bin/activate
(venv) asem3@asem3:~/Desktop/DOS_HW ASEM/FrontendServer$ export FLASK_ENV=development
(venv) asem3@asem3:~/Desktop/DOS_HW ASEM/FrontendServer$ export FLASK_APP=hello
(venv) asem3@asem3:~/Desktop/DOS_HW ASEM/FrontendServer$ flask run --host=192.168.121.132
 * Serving Flask app "hello" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://192.168.121.132:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 609-792-502
```

Catalog-Server

```
asem@asem: ~/Desktop/DOS_HW-ASEM/catalogServer
File  Edit  View  Search  Terminal  Help
asem@asem:~/Desktop/DOS_HW-ASEM/catalogServer$ source venv/bin/activate
(venv) asem@asem:~/Desktop/DOS_HW-ASEM/catalogServer$ export FLASK_ENV=development
(venv) asem@asem:~/Desktop/DOS_HW-ASEM/catalogServer$ export FLASK_APP=hello
(venv) asem@asem:~/Desktop/DOS_HW-ASEM/catalogServer$ flask run --host=192.168.121.134
 * Serving Flask app "hello" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://192.168.121.134:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 796-682-990
```

Order-Server

```
orderserver@orderserver: ~/Desktop/DOS_HW-ASEM/OrderServer
File  Edit  View  Search  Terminal  Help
orderserver@orderserver:~/Desktop/DOS_HW-ASEM/OrderServer$ source venv/bin/activate
(venv) orderserver@orderserver:~/Desktop/DOS_HW-ASEM/OrderServer$ export FLASK_ENV=development
(venv) orderserver@orderserver:~/Desktop/DOS_HW-ASEM/OrderServer$ export FLASK_APP=hello
(venv) orderserver@orderserver:~/Desktop/DOS_HW-ASEM/OrderServer$ flask run --host=192.168.121.135
 * Serving Flask app "hello" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://192.168.121.135:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 962-450-352
```

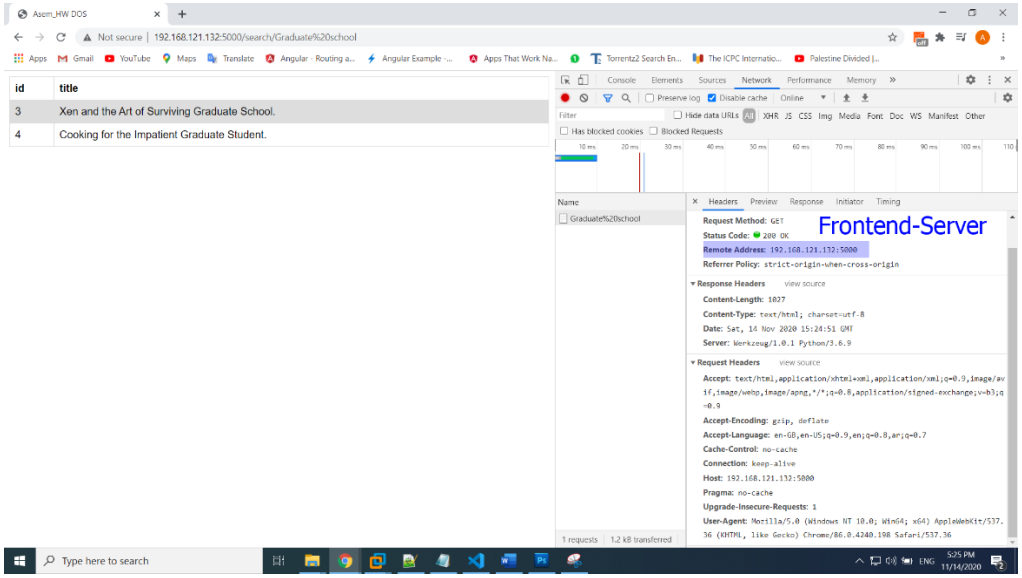
From Host OS (windows 10) we request the **GET** requests to Frontend-Server

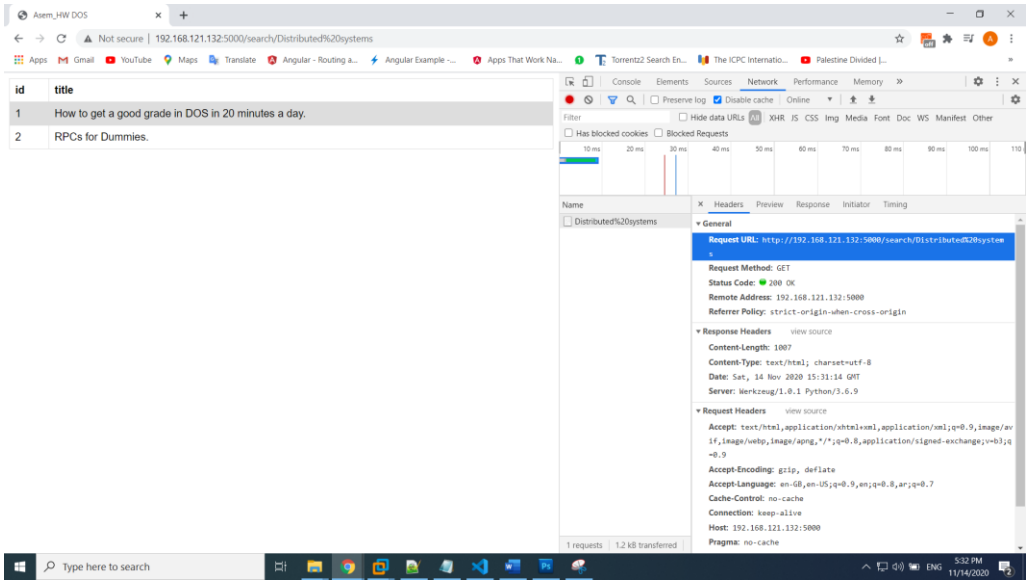
Note that we choose the GET method for all frontend server requests, because we request them from browser application client!

But to be more obvious and specific, frontend server sends **POST** request to order server (make new buy operation), and order server send **GET** request and **PUT** request to catalog server (**GET** for get data and **PUT** for update request).

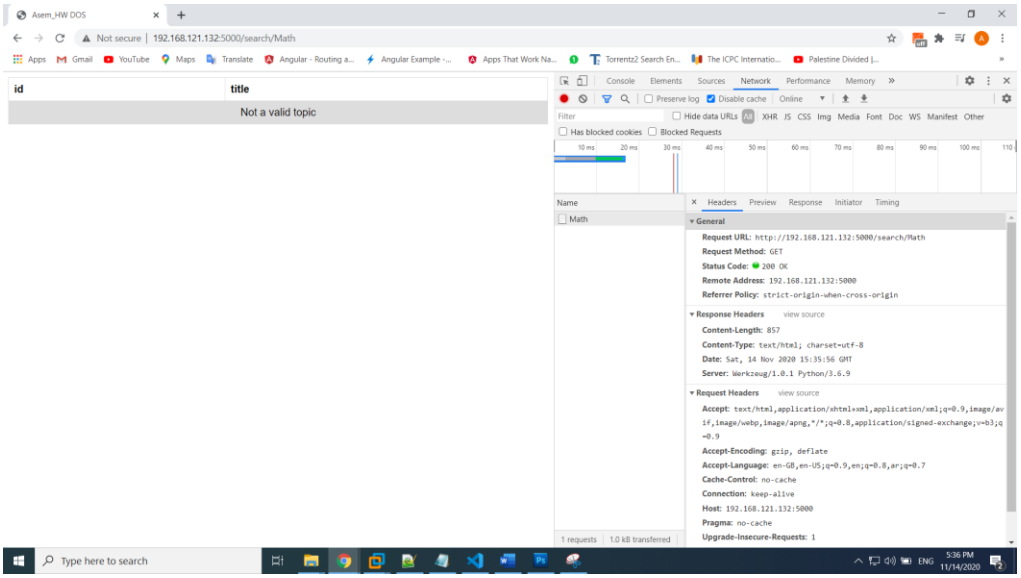
Search(topic):

If search success:



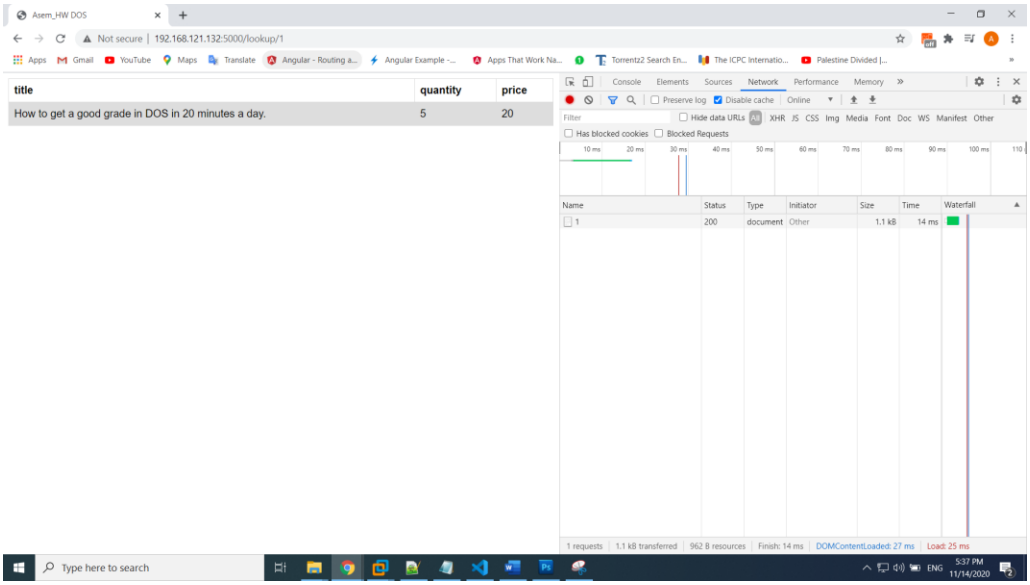


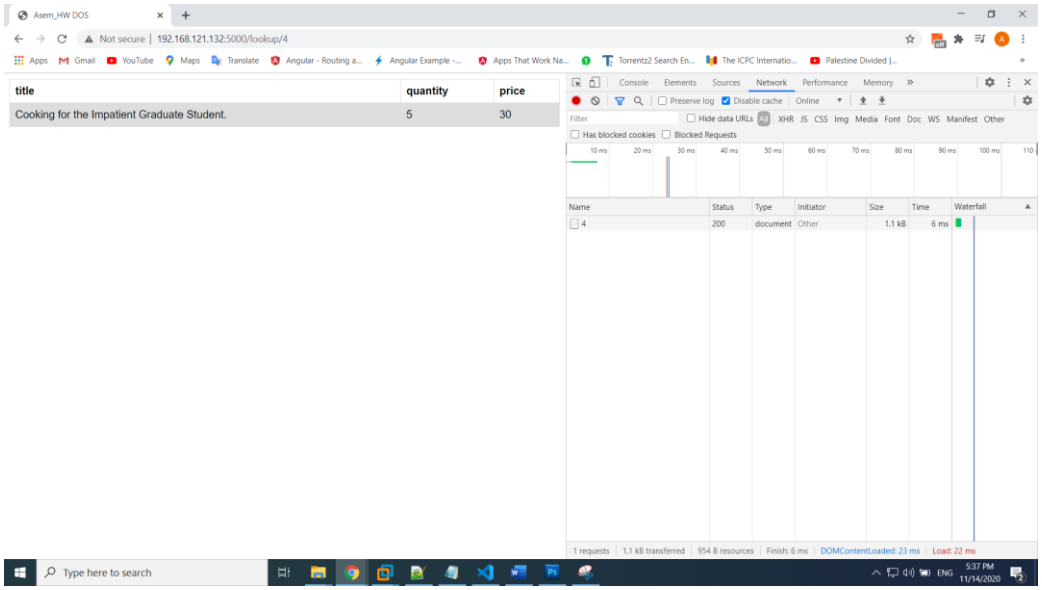
If search(topic) error:



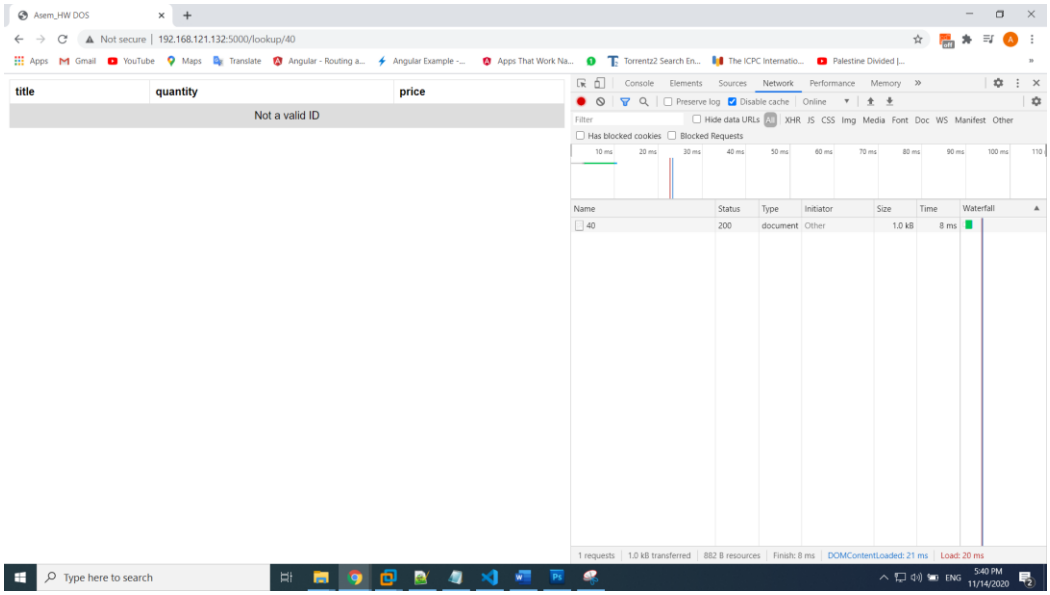
Lookup(item_number):

Success lookup:



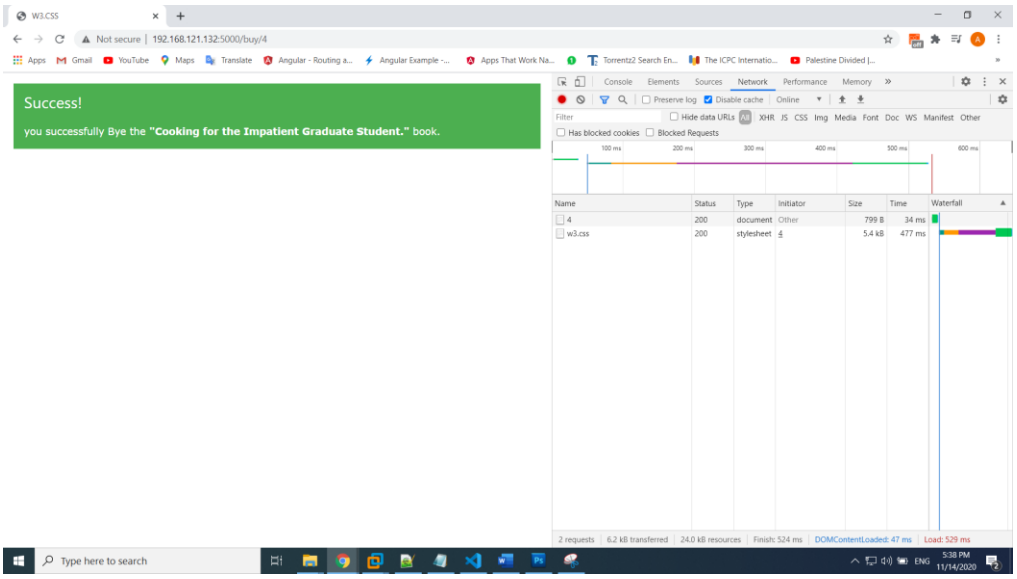

Asem_HW DOS
Not secure | 192.168.121.132:5000/lookup/4
title
quantity
price
Cooking for the Impatient Graduate Student.
5
30
Type here to search

Fail lookup:

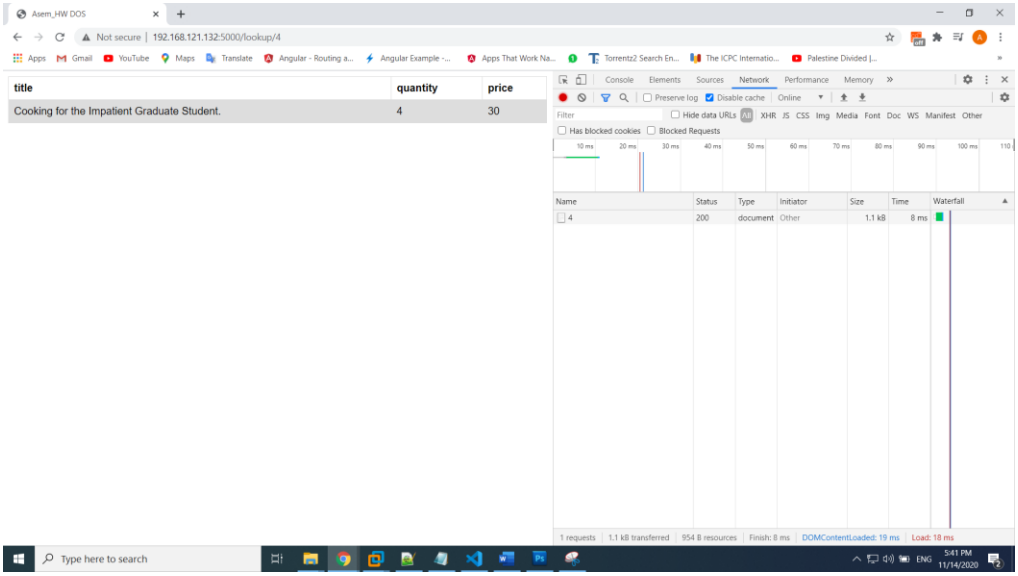

Asem_HW DOS
Not secure | 192.168.121.132:5000/lookup/40
title
quantity
price
Not a valid ID
Type here to search

Buy(item_number):

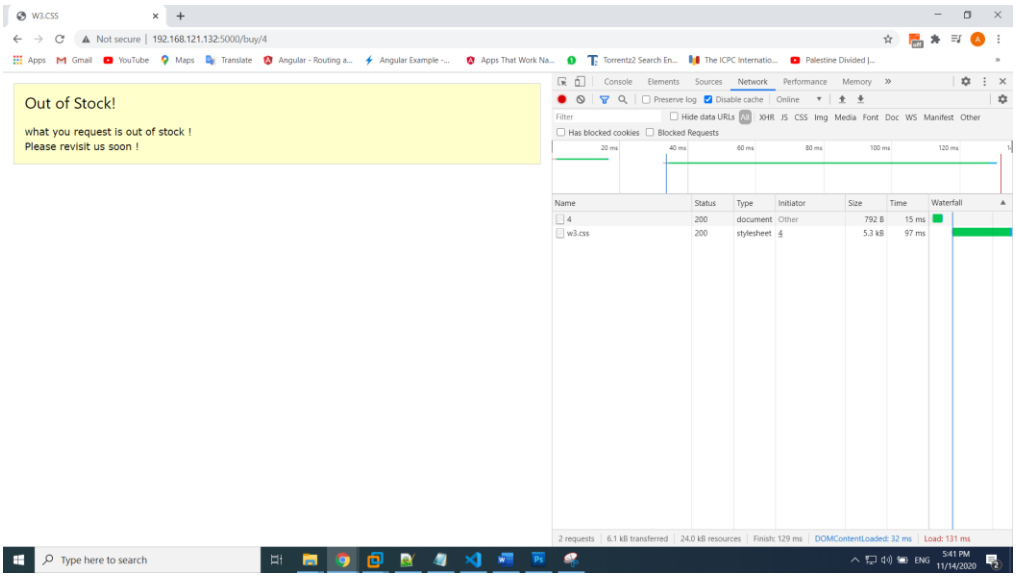
Success Buy



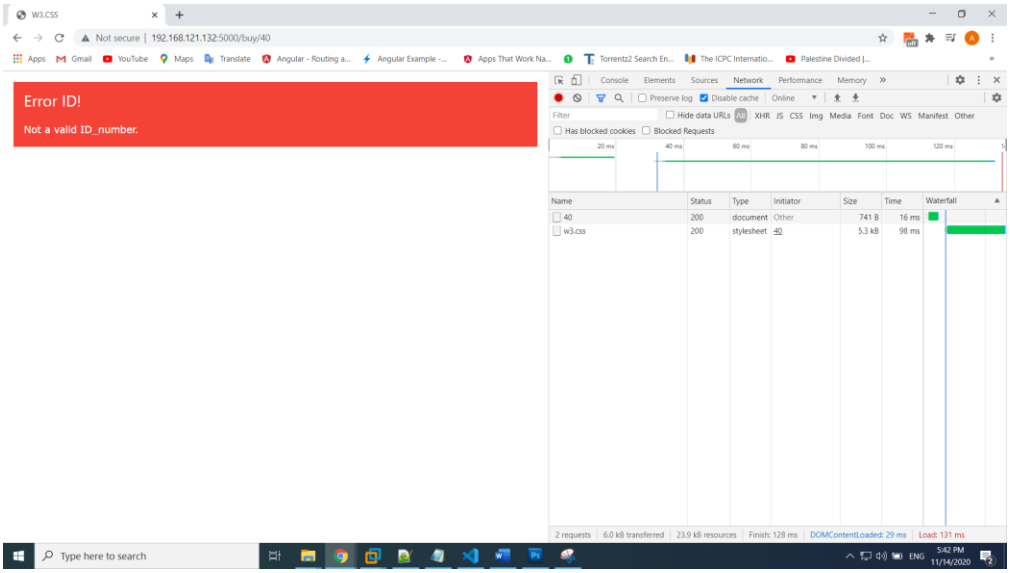
After Buy, must decrement 1 this book quantity:



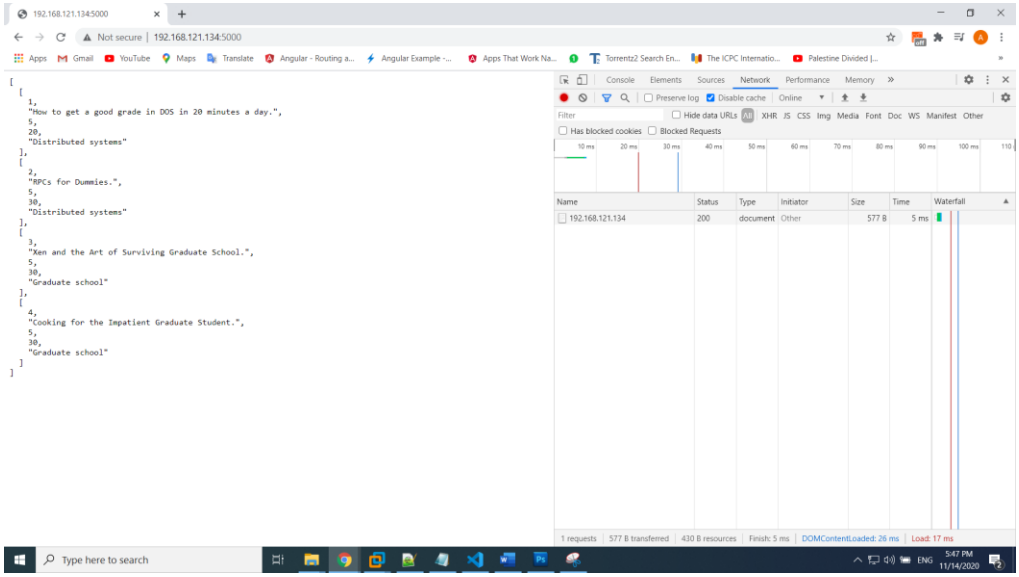
If Bought all of books from the stock:

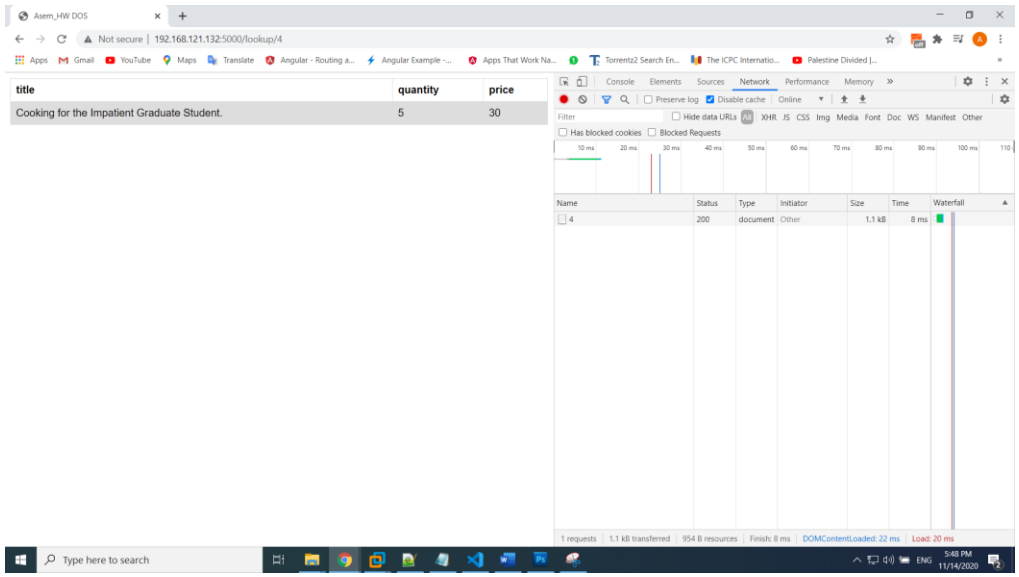


If ID of book is Not a valid:



If a new stock arrives: we request: http://catalog-server/ ,Then update all items to new quantity (5)





All comments that needed was written on the code, I upload Demo video on the repository. And I I tried to make everything clear.

Thank you 😍