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>

1. lookup(item\_number).

http://Frontend-server/lookup/<item\_number>

1. 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>

1. Query\_by\_item\_number(item\_number):

http://Catalog-server/query\_by\_ item\_number/<item\_number>

1. 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 **PUT** 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 **POST** 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

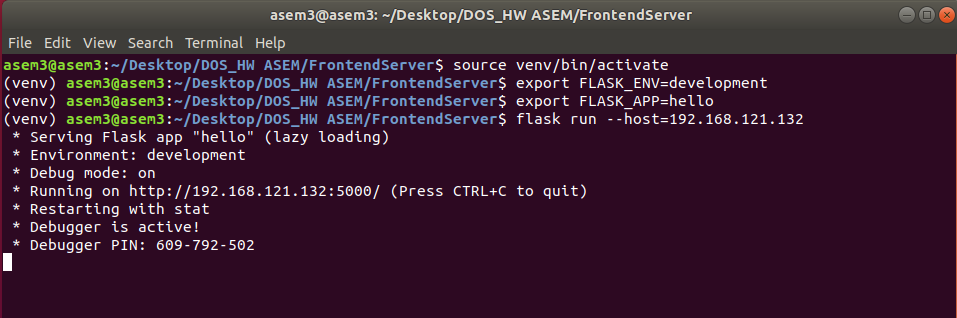
(venv) $ export FLASK\_ENV=development # mode as debugger (developing)

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

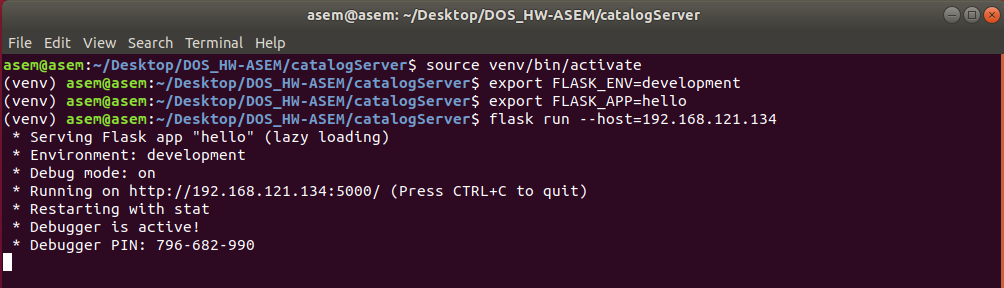
**Output generated:**

Running the servers:

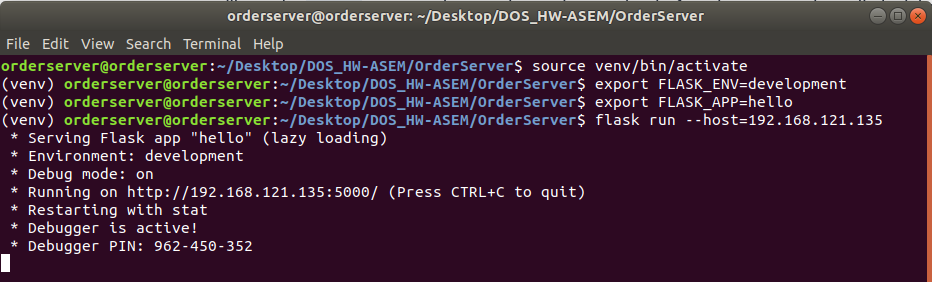
Frontend-Server



Catalog-Server



Order-Server



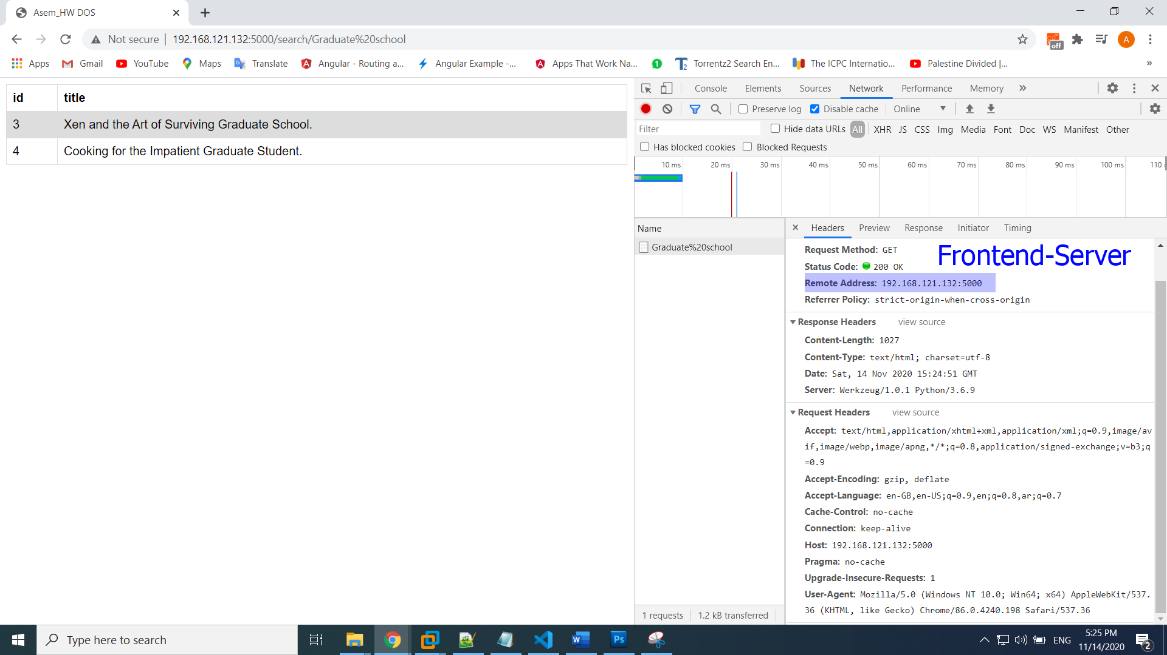
From Host OS (windows 10) we request the **GET** requests to Frontend-Server

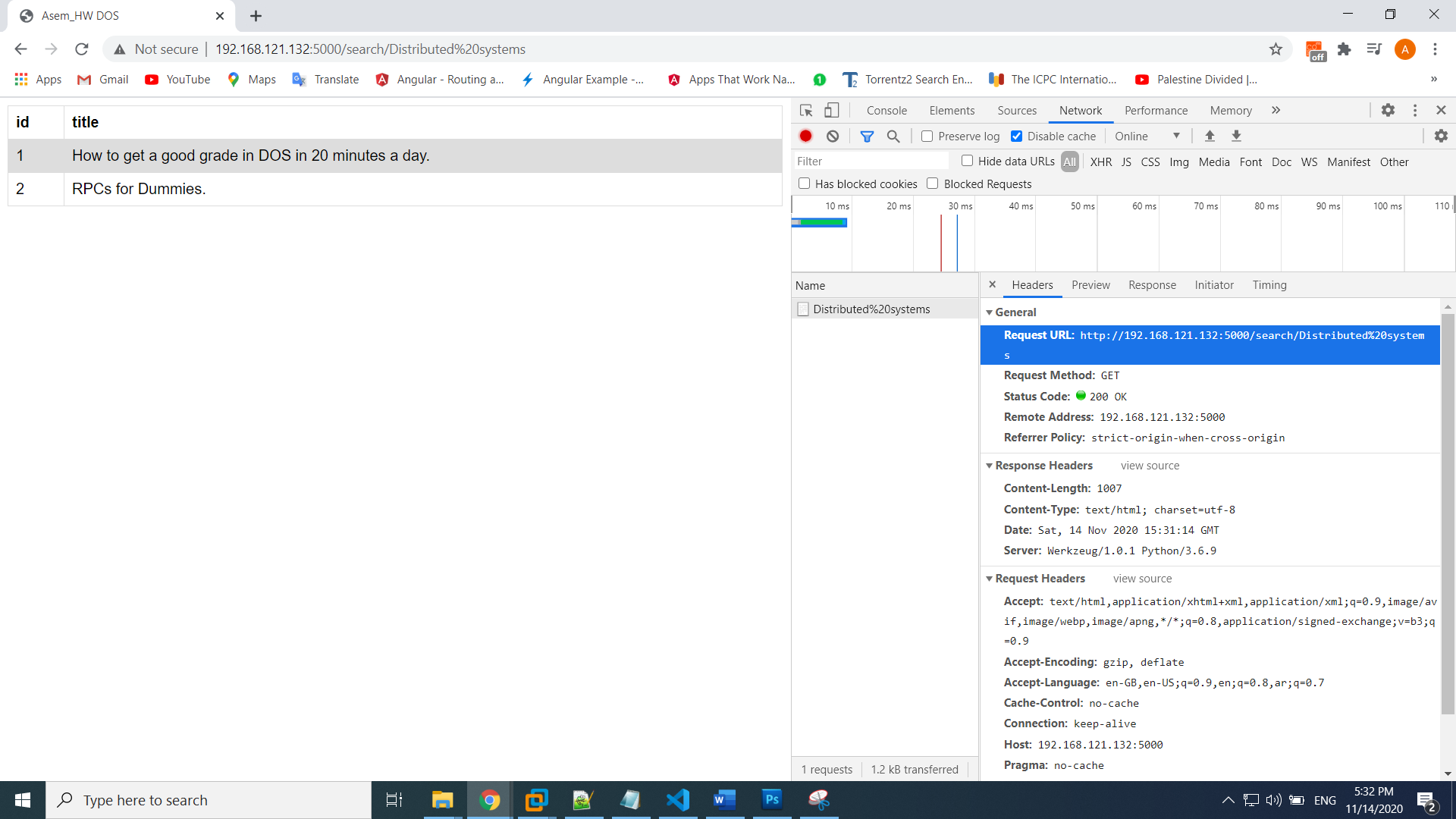
Note that we choose the GET method for all frontend server requests, because we request them from browser application client!

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

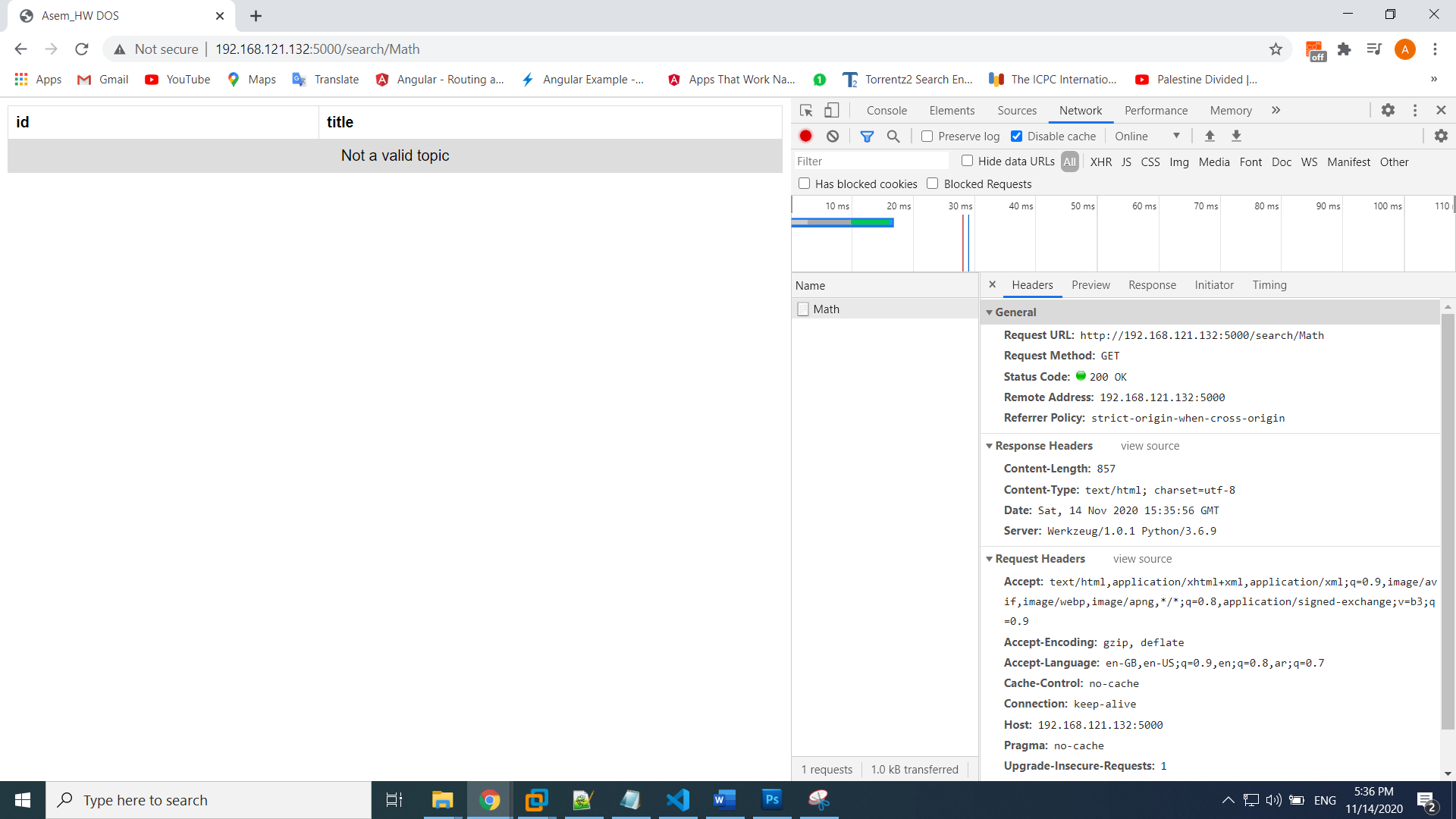
**Search(topic):**

If search success:



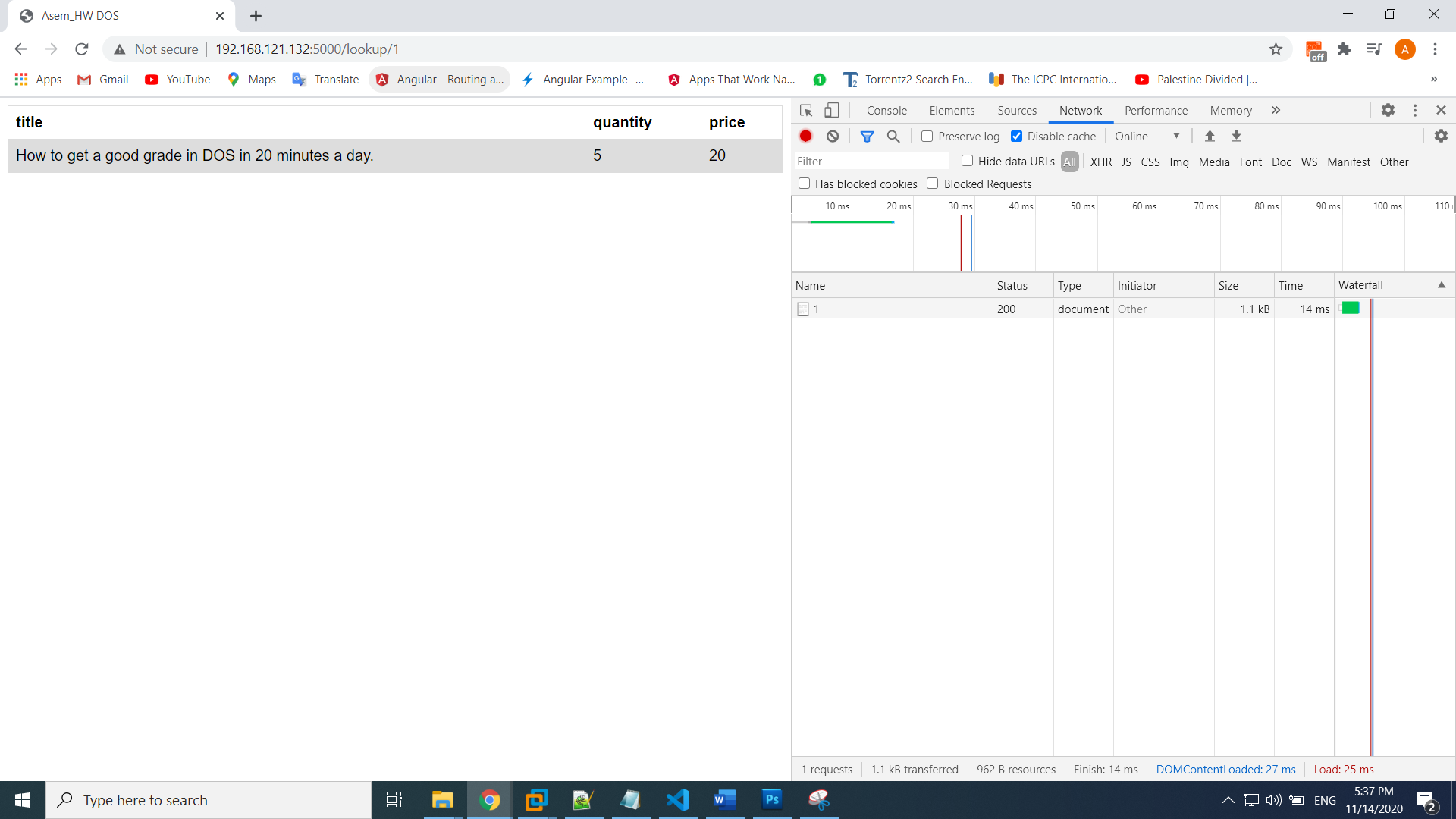


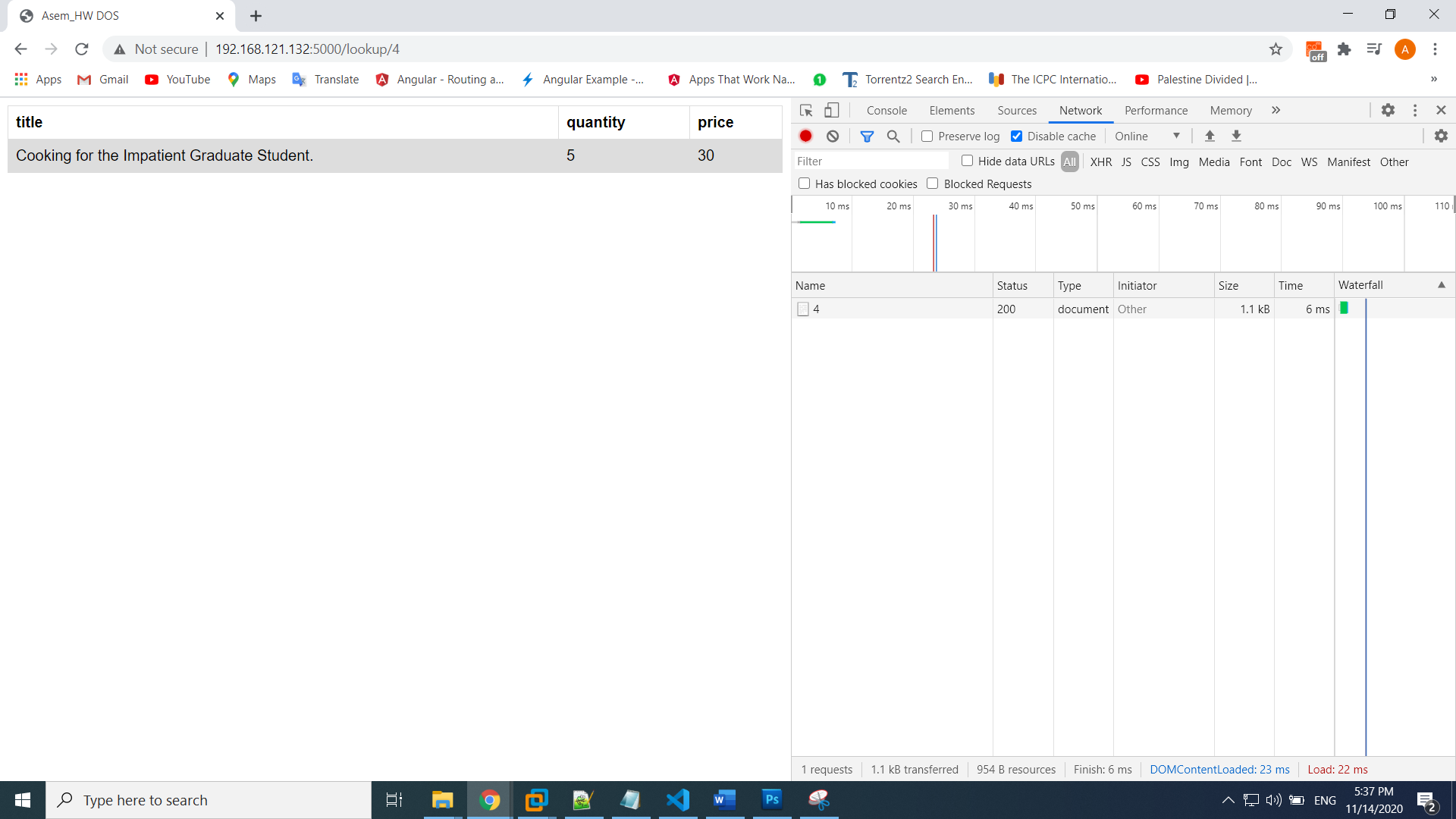
If search(topic) error:



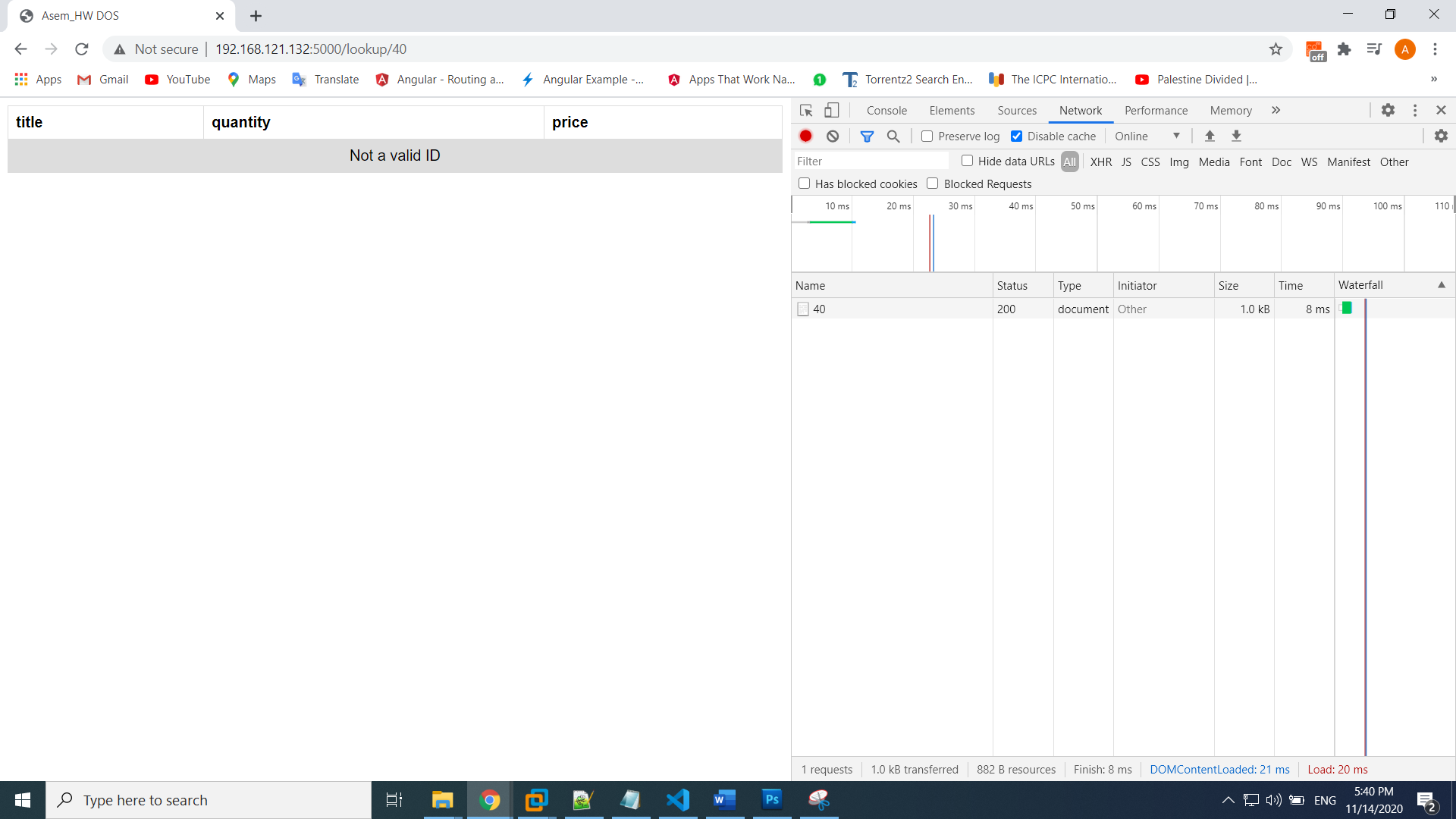
Lookup(item\_number):

Success lookup:



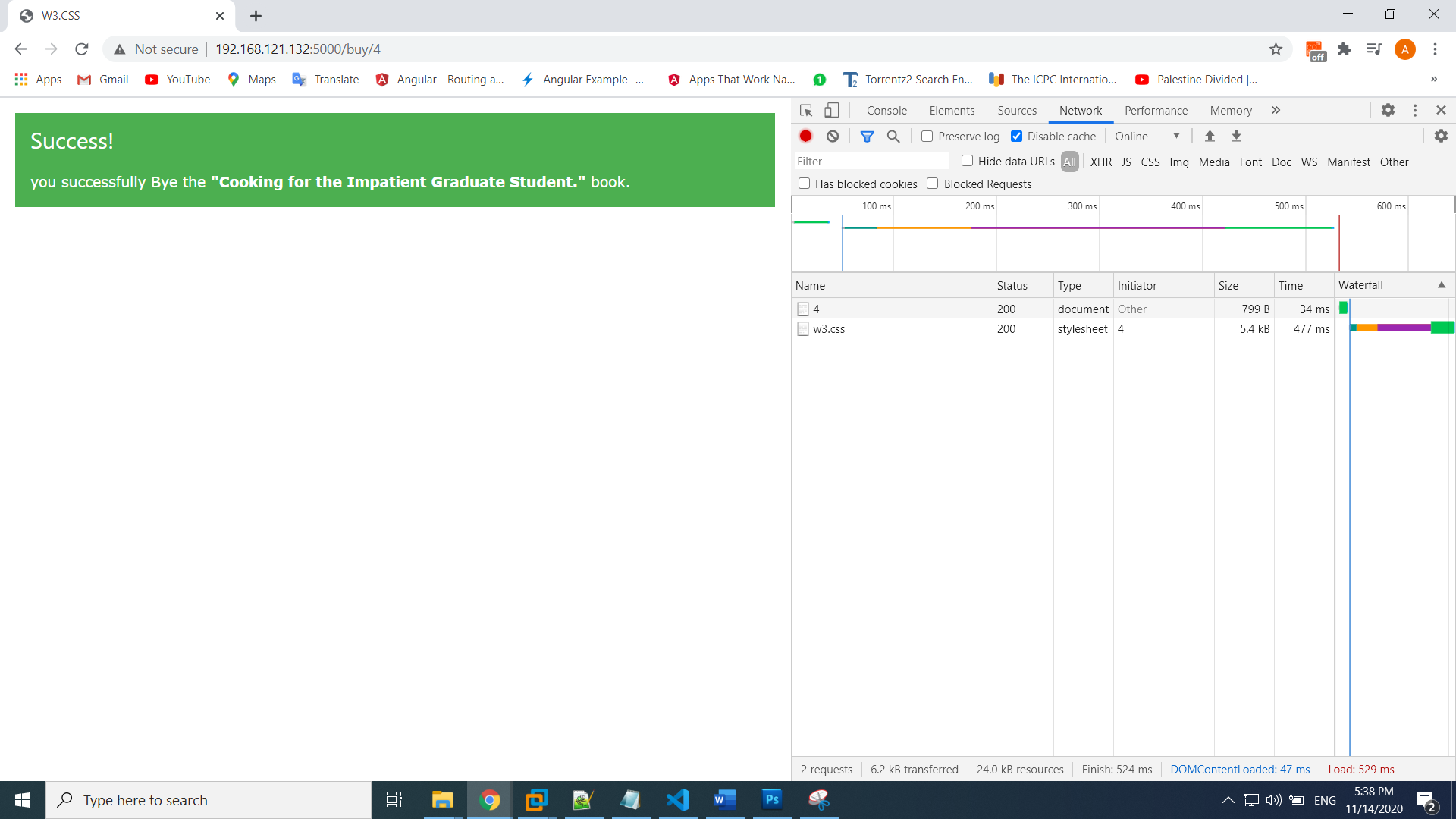


Fail lookup:

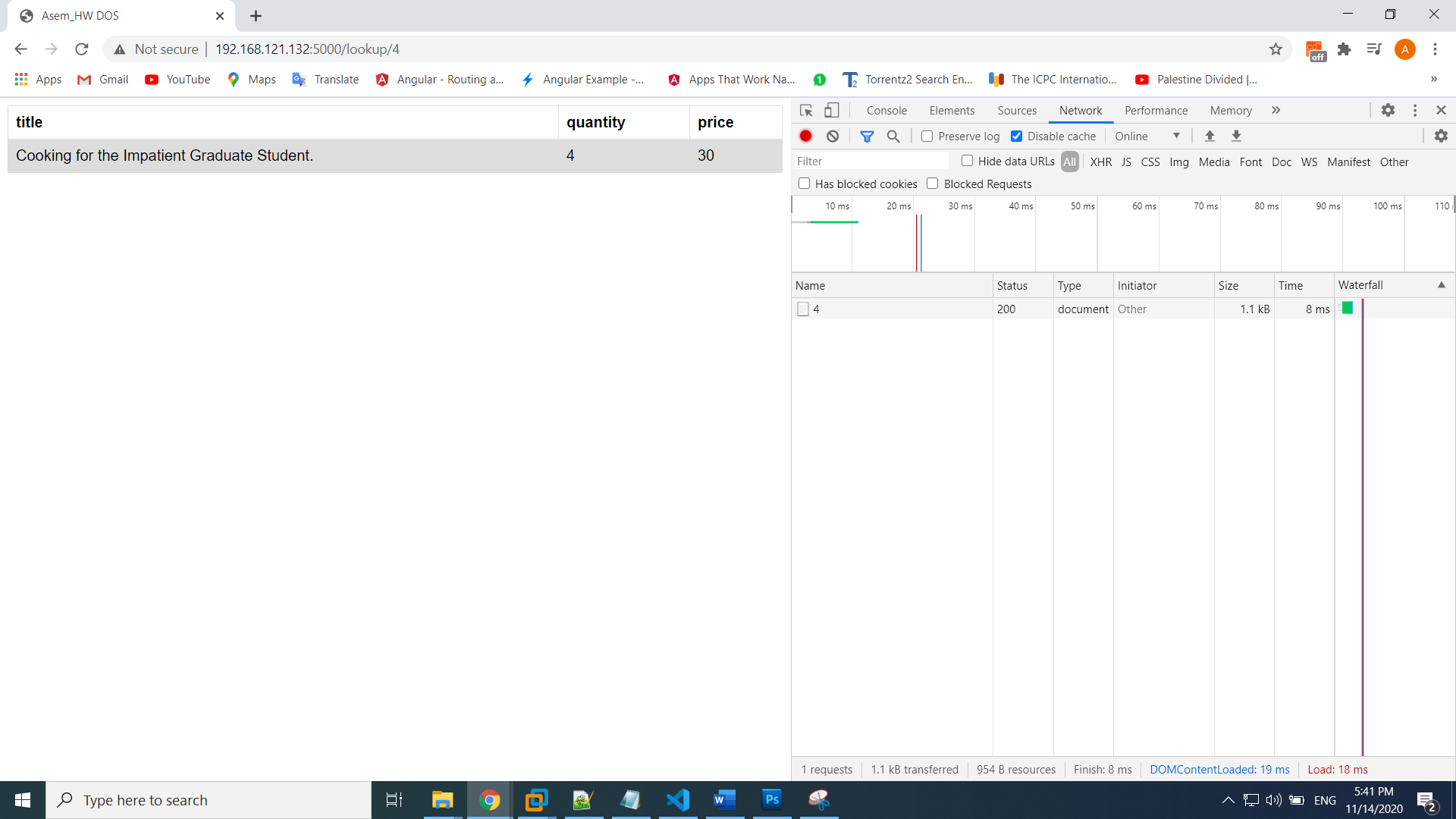


Buy(item\_number):

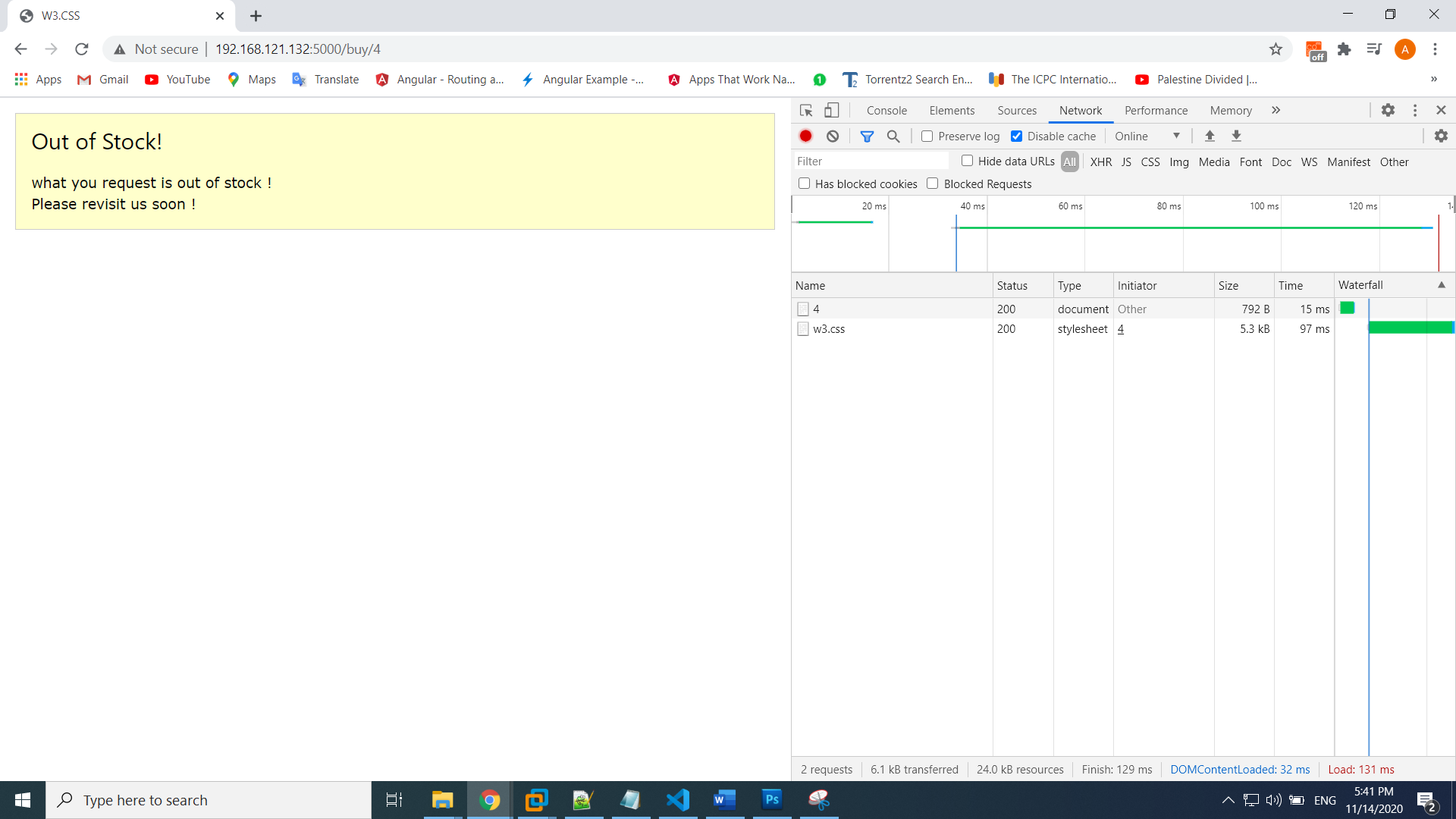
Success Buy



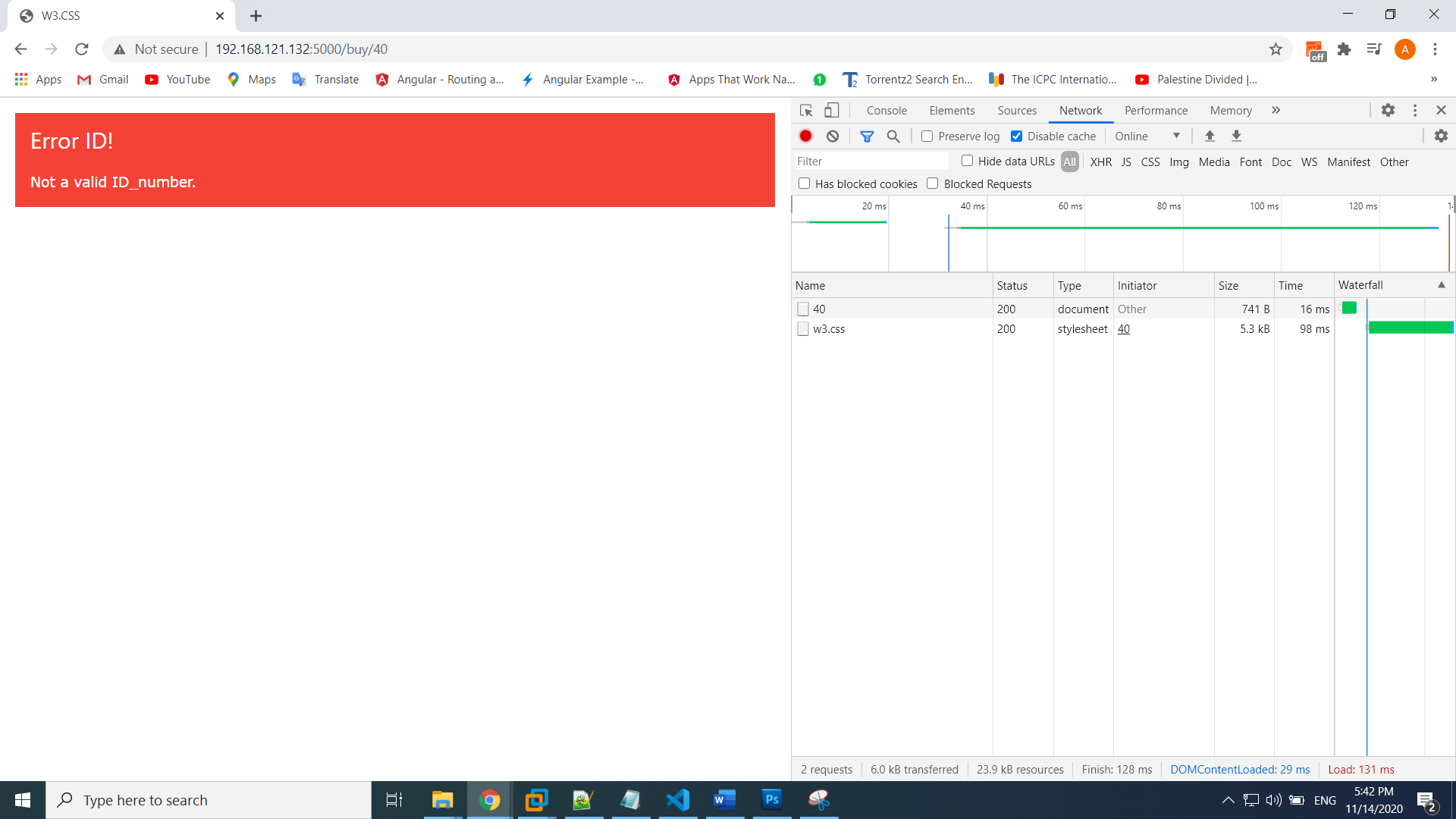
After Buy, must decrement 1 this book quantity:



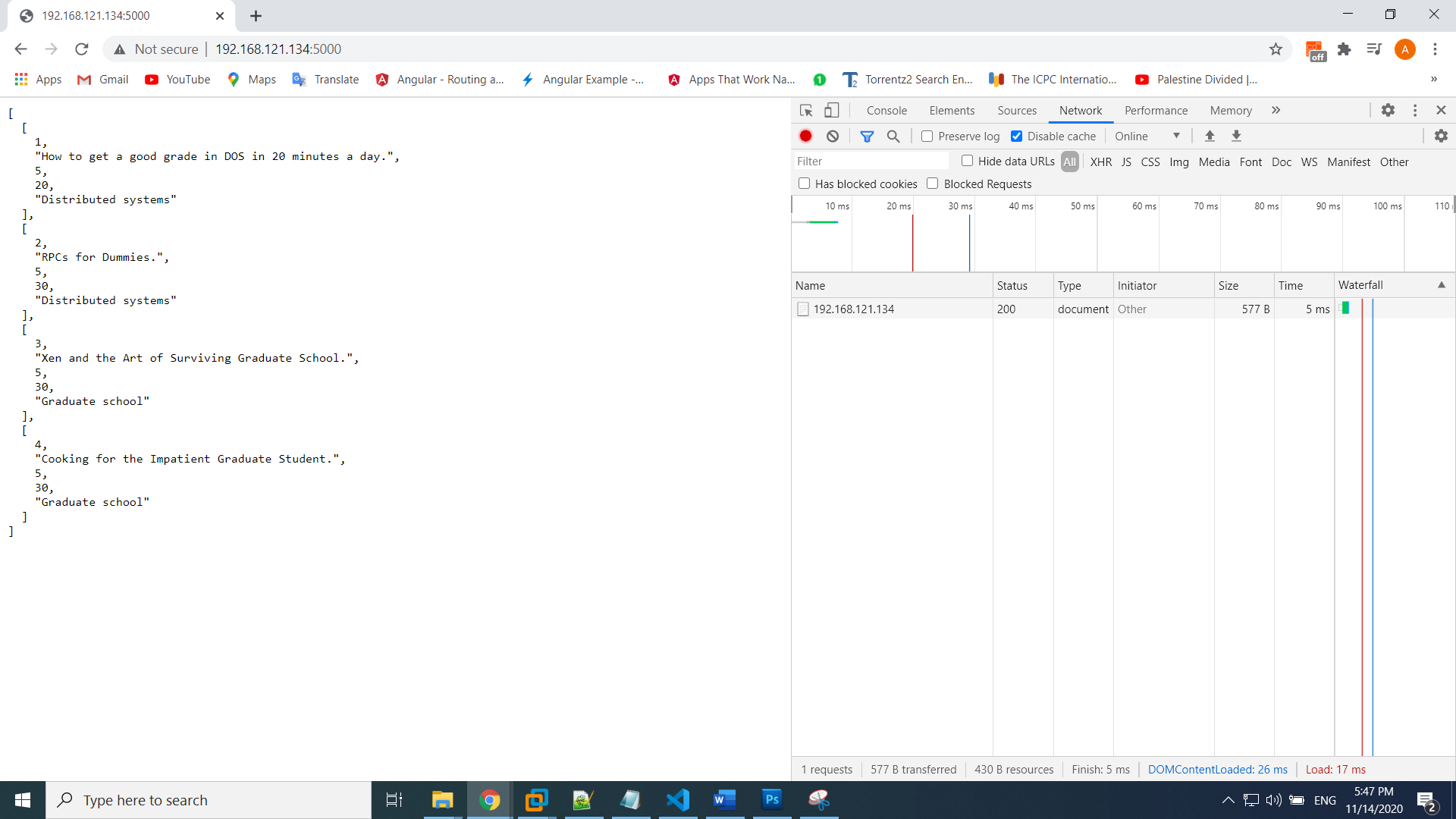
If Bought all of books from the stock:

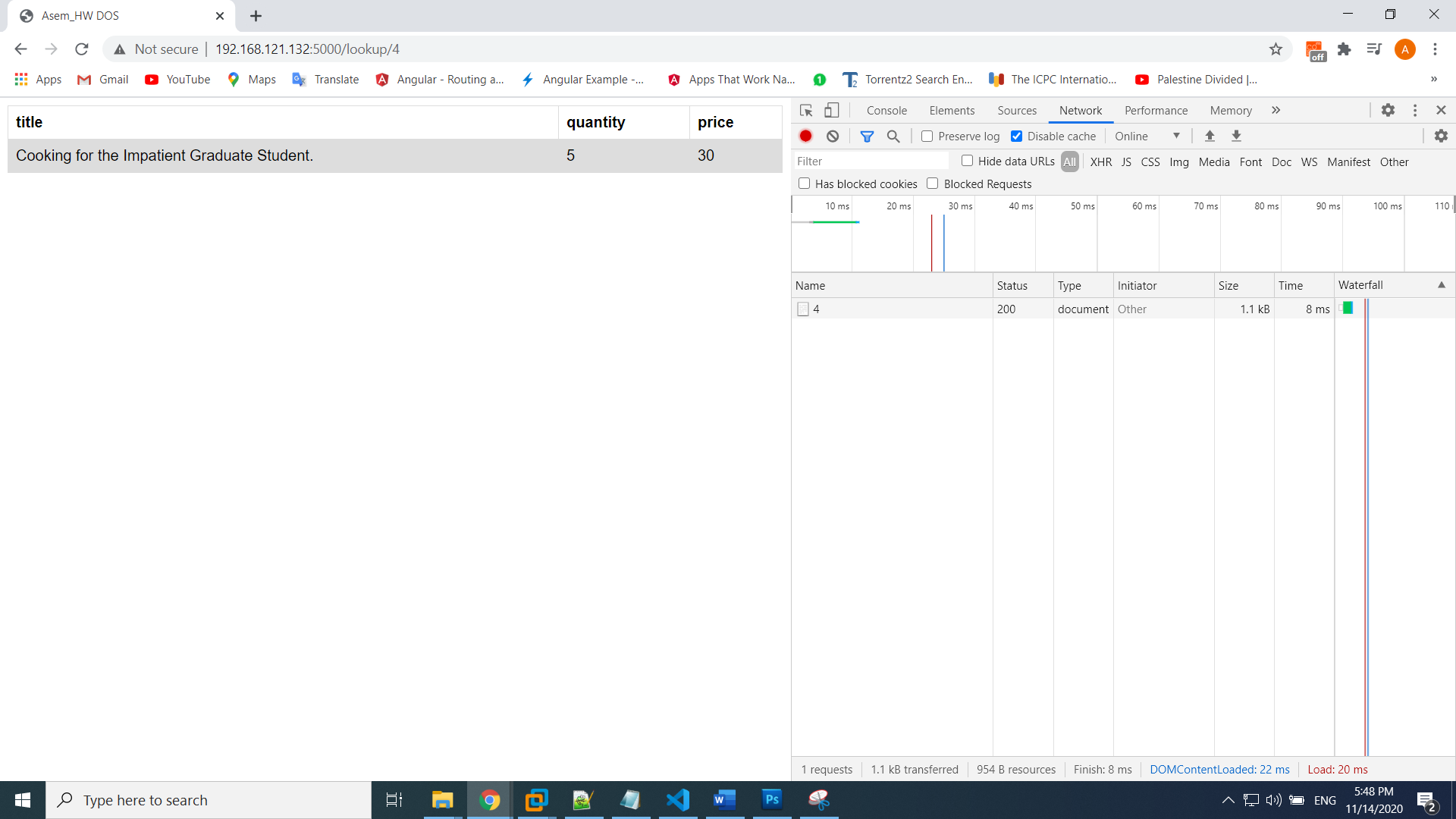


If ID of book is Not a valid:



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





**Inline comments:**

اكتب comments على الكود

**Short demo video (ZOOM APP recorder)**

**Note that if delete venv directory is better !**