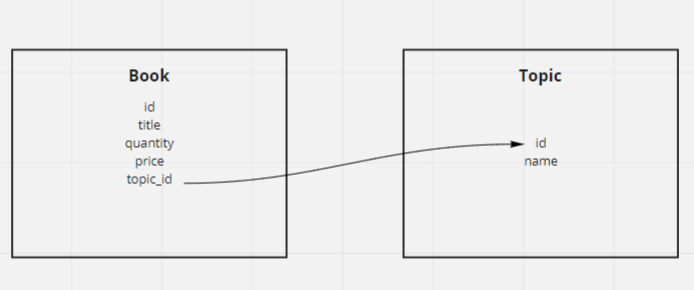
**DOS Bazar.com**

# System Design

As required in the homework specifications, the system’s design follows a microservice architecture in which the backend consists of two microservices:

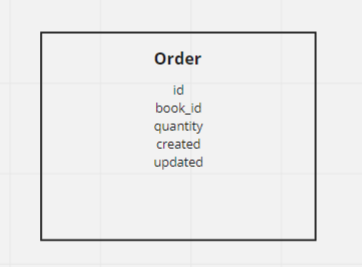
1. Catalog Microservice: This microservice is responsible for holding and manipulating data related to books catalog. The database consists of two tables as shown below:



This microservice exposes three endpoints:

* [GET] /search/{search\_phrase}: returns books with a topic that matches the search phrase
* [GET] /info/{book\_id}: returns the details of a book with the specified id
* [PUT] /update/{book\_id}: updates the book with the sent id using the values sent in the body

1. Order Microservice: This microservice is responsible for the orders details. It has a database with the following schema:



This microservice exposes one endpoint:

* / purchase/{book\_id}: this endpoint checks first if the book exists by quering the catalog microservice, if it exists then it decrements the quantity value by using the update endpoint in the catalog microservice and it stores the order entry

On the other hand, the frontend runs on its own separate server to and consists of three main parts that use the backend MSs:

1. search.sh: asks the user for the search phrase then queries the catalog microservice.
2. info.sh: asks the user for the book id then queries the catalog microservice.
3. purchase.sh asks the user for the book id then posts the request to the order microservice.

# Implementation & Deployment

### Backend Implementation:

Flask framework was used to implement the backend microservices because of both its ease of use and being a lightweight framework, which is suitable for our use case.

Alongside Flask, SQLAlchemy is used for object relational mapping (ORM) with the SQLite database which offers the capabilities to easily update the DB.

### Frontend Implementation:

As we were not required to implement a GUI, I implemented the frontend as a group of bash scripts each of them calls a different API. These bash scripts use cURL to make the API calls and use python to encode inputs for URLs and for formatting the output.

### Deployment:

To deliver a unified deployment experience, I used Vagrant which is a virtualization management software which allows the creation of configuration files (Vagrantfile) which can be used to reproduce the same virtual machine on different devices with the same settings which allows different developers to collaborate faster and with less deployment issues.

# How to Run It

As stated before, vagrant was used for the creation of the VMs. A detailed description on how to run each part of the project is provided in the README.md file in each part repository.

Before running any part of the project, we need to make sure that vagrant and VirtualBox are installed then follow these steps:

1. Go to the repo directory and open the terminal.
2. Run the virtual machine using the command **vagrant up**
3. Connect to the virtual machine using the command **vagrant ssh**
4. Go to the folder that contains the project files using the command **cd /vagrant**

Here the steps are a little different between backend MSs and the frontend (Note: make sure to run backend MSs before using frontend).

For backend do the following:

1. Create a virtual environment using **python3 -m venv ~/venv**
2. Activate the virtual environment using **source ~/venv/bin/activate**
3. Upgrade pip using **pip install –upgrade pip**
4. Install the project requirements using **pip install -r requirements.txt**
5. Set the environment variables using **export FLASK\_APP=server**
6. Run the server using **python -u -m flask run --host=0.0.0.0**

For frontend do the following:

1. For search run the command **sh search.sh**
2. For info run the command **sh info.sh**
3. For purchase run the command **sh purchase.sh**