

DISTRIBUTED OPERATING SYSTEMS

An-Najah National University



Made by:
Anas Mansour
Ruba Obaid

The Design

In this homework, the Laravel framework was used to build all the servers (order servers, catalog servers and the front-end server). Laravel was used since it's easy to use. It has a huge variance of plugins that can be used during development. One plugin used in this homework is Guzzle, which is a plugin to help send HTTP requests between servers easily. Also, the database used was just a simple text file (.txt), which was stored in the catalog server.

Caching

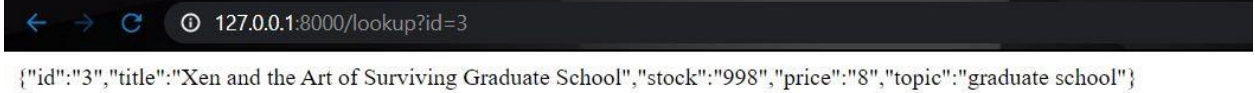
Caching was done in the front-end server using a text file stored in the front-end server. The file contains the request type (buy, lookup, search), its parameters and the response that was sent back for that request for each request sent by the front-end server. The Cache file would be updated every time a modification occurs on the database file in the catalog server, by deleting the cached entries in the cache file that are related in any way to the modified entry. The cache update happens immediately after the modifying request's response arrives.

Load Balancing and Consistency

For load balancing, the round robin algorithm was used. A session is created immediately when the front-end server is up, and that session works as a flag, which helps in alternating between servers for every request to spread the load among them. The problem with this is that it adds even more overhead on the back-end servers (catalog specifically) which contain the database file, where they need to send extra requests between each other in order to maintain the consistency. That was done by simply sending a "buy" request from one catalog server to another every time one of them modifies its database file.

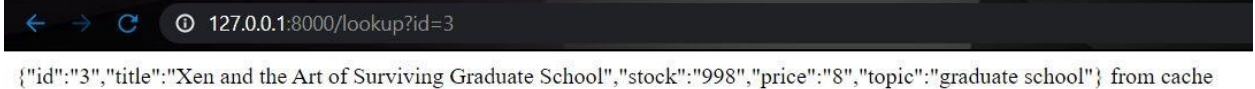
Example:

At first, a simple “lookup” request was sent to the front-end server, the request would be redirected to one of the catalog servers and will get the following response:



```
127.0.0.1:8000/lookup?id=3  
{ "id": "3", "title": "Xen and the Art of Surviving Graduate School", "stock": "998", "price": "8", "topic": "graduate school" }
```

Then, if the same request was sent directly after, the response would be retrieved from the cache without having to send a request to the catalog server (“from cache” was added for debugging purposes):



```
127.0.0.1:8000/lookup?id=3  
{ "id": "3", "title": "Xen and the Art of Surviving Graduate School", "stock": "998", "price": "8", "topic": "graduate school" } from cache
```

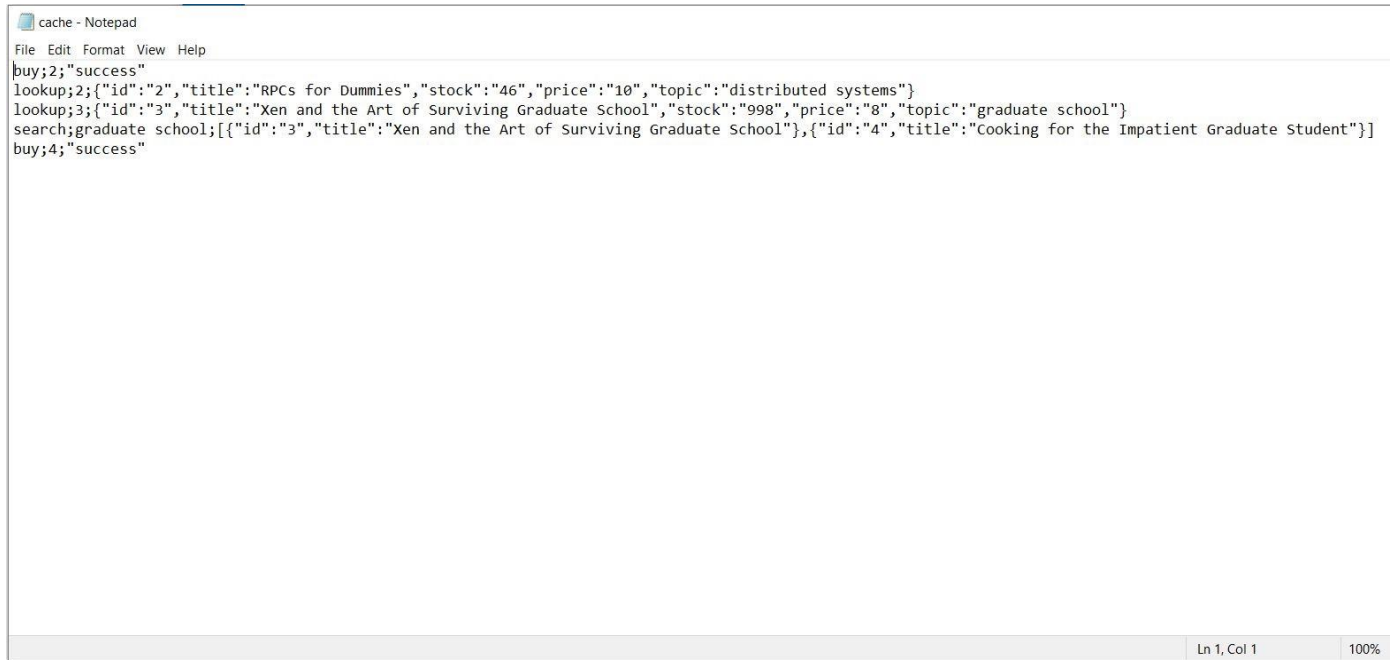
This is what the cache file looks like:

```
cache - Notepad
File Edit Format View Help
buy;1;"success"
lookup;1;{"id":"1","title":"How to get a good grade in DOS in 20 minutes a day","stock":"1","price":"20","topic":"distributed systems"}
buy;2;"success"
lookup;2;{"id":"2","title":"RPCs for Dummies","stock":"46","price":"10","topic":"distributed systems"}
lookup;3;{"id":"3","title":"Xen and the Art of Surviving Graduate School","stock":"998","price":"8","topic":"graduate school"}
```

The cache file is designed to max out on storing 5 requests (can be easily modified), after that, it will start deleting the earliest stored request:

```
cache - Notepad
File Edit Format View Help
buy;2;"success"
lookup;2;{"id":"2","title":"RPCs for Dummies","stock":"46","price":"10","topic":"distributed systems"}
lookup;3;{"id":"3","title":"Xen and the Art of Surviving Graduate School","stock":"998","price":"8","topic":"graduate school"}
lookup;4;{"id":"4","title":"Cooking for the Impatient Graduate Student","stock":"2000","price":"5","topic":"graduate school"}
search;graduate school;[{"id":"3","title":"Xen and the Art of Surviving Graduate School"}, {"id":"4","title":"Cooking for the Impatient Graduate Student"}]
```

When performing a “buy” request, the related entries to that request will be deleted from the cache upon the response arrival:



```
cache - Notepad
File Edit Format View Help
buy;2;"success"
lookup;2;{"id":"2","title":"RPCs for Dummies","stock":"46","price":"10","topic":"distributed systems"}
lookup;3;{"id":"3","title":"Xen and the Art of Surviving Graduate School","stock":"998","price":"8","topic":"graduate school"}
search;graduate school;[{"id":"3","title":"Xen and the Art of Surviving Graduate School"}, {"id":"4","title":"Cooking for the Impatient Graduate Student"}]
buy;4;"success"
```

Ln 1, Col 1 100%

Performance:

The response time of the request was calculated for several attempts as in the table, it was noticed that caching saves almost half of the request time, this happened for this simple project, the difference will be notably huge when having more complex system with extra hops between servers.

Direct call	From cache
418ms	225ms
465ms	210ms
414ms	191ms
520ms	206ms

How to run the servers:

First, store each server file in a separate machine (or virtual machine), you will need at least 5 machines. Then, you have to install PHP and Laravel on each machine.

To install Laravel, you need to first install the Composer which you can get on <https://getcomposer.org/>. After that, open command prompt on Windows (or the terminal on Linux) and type in the following command:

```
composer global require laravel/installer
```

After Laravel has been installed, we need to install the Guzzle plugin which we mentioned earlier. Navigate to the server folder on each machine and type in the command `composer require guzzlehttp/guzzle`

Now all we need is to run the servers. For running the servers, navigate to the server folder on each machine and type the following command

```
php artisan serve
```

However, for the backend servers, you need to assign custom IPs and Ports for them to work, and to do that, simply add two parameters to the previous command:

```
php artisan serve --host *serverIPAddress* --port *serverPort*
```

Replace the `*serverIPAddress*` and `*serverPort*` for each server as follows:

	Server IP Address	Server Port
Catalog #1	192.168.1.103	8000
Catalog #2	192.168.1.104	8000
Order #1	192.168.1.105	8000
Order #2	192.168.1.108	8000

And now, you have a fully functioning system that will work perfectly.