

Online book store – DOS project

Masa alsayyed

Meera saymeh

To build our online book store we used [Laravel php](#), Xamp and mySQL database ,we build three micro web services one for the front-end where the user can search for a book, look for information about a book and purchase books, and two services in the server side one for the orders and the other for the books catalog.

We faced a problem trying to run the services on different virtual machines and finally decided to work using the localhost and give each service a different port.

Front-end -> localhost:8002

Order -> localhost:8001

Catalog -> localhost:8000

```
Command Prompt - php -S localhost:8002 -t public
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\meera saymeh>cd desktop

C:\Users\meera saymeh\Desktop>cd front-end

C:\Users\meera saymeh\Desktop\front-end>php -S localhost -t public
Invalid address: localhost

C:\Users\meera saymeh\Desktop\front-end>php -S localhost:8002 -t public
[Wed Aug 10 19:48:26 2022] PHP 8.1.8 Development Server (http://localhost:8002) started
```

```
Command Prompt - php -S localhost:8000 -t public
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\meera saymeh>cd desktop

C:\Users\meera saymeh\Desktop>cd books-api

C:\Users\meera saymeh\Desktop\books-api>php -S localhost:8000 -t public
[Wed Aug 10 19:47:05 2022] PHP 8.1.8 Development Server (http://localhost:8000) started
```

```
Command Prompt - php -S localhost:8001 -t public
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\meera saymeh>cd desktop

C:\Users\meera saymeh\Desktop>cd order-api

C:\Users\meera saymeh\Desktop\order-api>php -S localhost:8001 -t public
[Wed Aug 10 19:47:39 2022] PHP 8.1.8 Development Server (http://localhost:8001) started
```

Purchase :

When sending a purchase request from the front-end the user have to send the id for the book in the URL request and the number of books (items) in the Jason , the request will be sent to the orderIP.

Then in the orderIP if the request was valid, we'll get the info about the pouched book from the catalogIP and send it back as a Jason, the user will see the information about his purchase and the total cost.

in the database the order will be added to the order table, and the number of items for the book will get decreased in the book table. If the book does not exist or if it's out of stock the purchase won't get completed.

Example:

GET localhost:8002/api/info POST localhost:8002/api/pt. + ... No Environment

localhost:8002/api/purchase/1 Save Edit

POST localhost:8002/api/purchase/1 Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "items": 1
3 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 1433 ms Size: 355 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "bookID": 1,
3   "numItems": 1,
4   "total": 40,
5   "updated_at": "2022-08-10T16:57:39.000000Z",
6   "created_at": "2022-08-10T16:57:39.000000Z",
7   "id": 28
8 }
```

In case you made a purchase for unexacting book -> "order is not completed" massage will appear

localhost:8002/api/purchase/11

POST localhost:8002/api/purchase/11

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "items": 1
3 }
```

Body Cookies Headers (5) Test Results Status: 500 Internal Server Error Time: 416 ms Size:

Pretty Raw Preview Visualize

```
^ "Order is not completed"
```

Search:

When sending a search request we need to send the book name in the URL, the request will go to the catalogIP, we'll get the info about the book and send it back as a JSON.

Example:

localhost:8002/api/search/Spring in the Pioneer Valley

GET localhost:8002/api/search/Spring in the Pioneer Valley

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 182 ms Size: 404 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 7,
3   "title": "Spring in the Pioneer Valley",
4   "topic": "Spring is Here",
5   "items": 17,
6   "cost": 70,
7   "created_at": "2022-08-07T21:00:00.000000Z",
8   "updated_at": "2022-08-09T09:55:14.000000Z"
9 }
```

Info:

When sending an info request from the front-end we need to send the book id in the URL, it will go to the catalogIP as in the search function and send back the results as JSON.

Example:

GET localhost:8002/api/info/1 POST localhost:8002/api/p...

No Environment

localhost:8002/api/info/1

GET localhost:8002/api/info/1

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (7) Test Results Status: 200 OK Time: 195 ms Size: 431 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "title": "How to get a good grade in DOS in 40 minutes a day",
4   "topic": "Distributed systems",
5   "items": 23,
6   "cost": 40,
7   "created_at": "2022-10-06T21:00:00.000000Z",
8   "updated_at": "2022-08-10T16:57:39.000000Z"
9 }
```

And If the book is not in store -> “this book does not exists” message will appear

The screenshot shows a REST client interface with the following details:

- URL: localhost:8002/api/info/8
- Method: GET
- Query Params table:

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Status: 500 Internal Server Error
- Time: 513 ms
- Size: 16.75
- Response Body (Pretty):

```
^ "This Books Does Not Exists"
```

In the second part of the project we were asked to made replicas for the server side. the catalog, the order and the database. And to guarantee consistence between the replicas.

We added another catalog and order server and we used round-robin to choose which replica the request will go to by sitting a flag that will change value each time a request will get sent.

Also we were asked to have a cache system for the request to enhance the performance, we did our research on Laravel and there's a build in cache system that can be use in different ways, the default way if the file, a file will be created for each server (catalog, order) and this way is a per-request cache.

The other way is the array , an array will be created for out project and it will cache the requests, we found this way has a slightly better performance than the file one. Also there is popular caching backends that larval supports like Memcached and DynamoDB.

As much as we wanted to have an external database cache like Redis as the performance would be much better, we didn't have the time to set it up and decided to go with the array approach.

Repeating a request we can notice the time of response is faster each time.

-First time we will get the data from the database, next time is faster since the data is cached .

localhost:8002/api/info/1

GET localhost:8002/api/info/1

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 1876 ms Size: 431 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "title": "How to get a good grade in DOS in 40 minutes a day",
4   "topic": "Distributed systems",
5   "items": 24,
6   "cost": 40,
7   "created_at": "2022-10-06T21:00:00.000000Z",
8   "updated_at": "2022-08-10T11:34:58.000000Z"
9 }
```

GET localhost:8002/api/info/1 POST localhost:8002/api/info/1

No Environment

localhost:8002/api/info/1

GET localhost:8002/api/info/1

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 223 ms Size: 431 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "title": "How to get a good grade in DOS in 40 minutes a day",
4   "topic": "Distributed systems",
5   "items": 24,
6   "cost": 40,
7   "created_at": "2022-10-06T21:00:00.000000Z",
8   "updated_at": "2022-08-10T11:34:58.000000Z"
9 }
```

The screenshot shows the Postman interface for a GET request to `localhost:8002/api/info/1`. The request is saved and ready to be sent. The 'Headers' tab is selected, showing a table with columns for KEY, VALUE, and DESCRIPTION. The 'Body' tab is also visible, showing a JSON response. The status bar at the bottom indicates a successful response with a status of 200 OK, a time of 195 ms, and a size of 431 B. The response is displayed in the 'Body' tab, showing a JSON object with fields for id, title, topic, items, cost, created_at, and updated_at.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```
{
  "id": 1,
  "title": "How to get a good grade in DOS in 40 minutes a day",
  "topic": "Distributed systems",
  "items": 23,
  "cost": 40,
  "created_at": "2022-10-06T21:00:00.000000Z",
  "updated_at": "2022-08-10T16:57:39.000000Z"
}
```

- In order to run our project, we need to open the xamp and every server on their port, we used postman while working since we can see more details about the request, time and the status