LAB 2 – Dos.

https://github.com/AreenBelal/DOS_project

Areen Belal Ateeq – 11923874

First of all. Write –compose up command on cmd.
I have five server: two server for catalog, two for order and one for FE(frontend).
Catalog and Catalogweb2 have same operations (3 function : info, search , update) but run on different port, Default one at 8003 and replica at 8004. Also Order server same functionality, default one run at port 8001 and replica at 8005.

```
3 usages
class CatalogServicePicker(ServerPicker):
    servers = ["http://catalogweb:8003", "http://catalogweb2:8004"]


1 usage
class OrderServicePicker(ServerPicker):
    servers = ["http://orderweb:8001", "http://orderweb2:8005"]
```

FE same as part1 decided where the request will send to catalog or order server.
The difference in this part that i have two servers for Catalog and server. The choice depend on function (pick_server) and use Round Robin technique. which used in various computing contexts, including load balancing.

The round-robin load balancing technique divides up incoming requests or tasks among several servers in an equal amount. By doing this, it is made sure that no server is overloaded with queries while the others are left idle.
The first request send to server 1, next request to server 2 and so on to do load balancing.

```python
2 usages
class ServerPicker:
    servers = []

    4 usages
    def pick_server(self):
        server = self.servers[0]
        self.servers.reverse()
        return server
```

I have databases for each catalog server and I should make consistency between them.

For example purchase operation means update data. To ensure consistency the data must edited in two database in each server.

Always read (has information) from default database. But if data edited then it update in both to get fresh and same data from both.

```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR.parent / 'db.sqlite3',
    },
    'replica': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR.parent.parent / '../catalog2' / 'db.sqlite3',
    }

}
```

About Cache. i do mapping for each API with it response.

if API doesn’t exist then get it from server.

When request happened, the system has two options, if data stored in cache then the response will be from it which reduce the time to get response. Else the system get needed data from original or replica server.

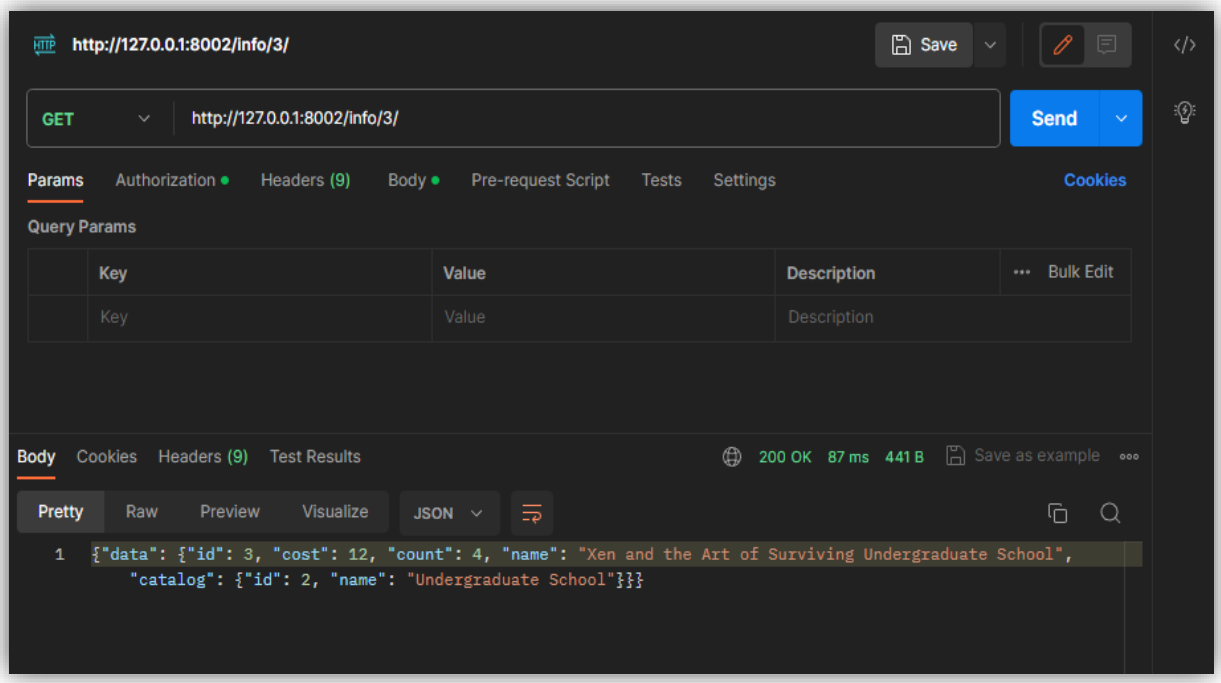
When I have purchase operation the cache will be clear.

Here the operation related to cache.

```python
class CacheService:
    def __init__(self):
        self.cache = {}

    1 usage (1 dynamic)
    def clear_cache(self):
        self.cache = {}
        return True

    6 usages (6 dynamic)
    def get_cache(self, api):
        return self.cache.get(api, {}).get('value')

    3 usages (3 dynamic)
    def set_cache(self, api, value):
        self.cache[api] = {'value': value, 'editied': False}
        return True

    def delete_cache(self, api):
        del self.cache[api]
```

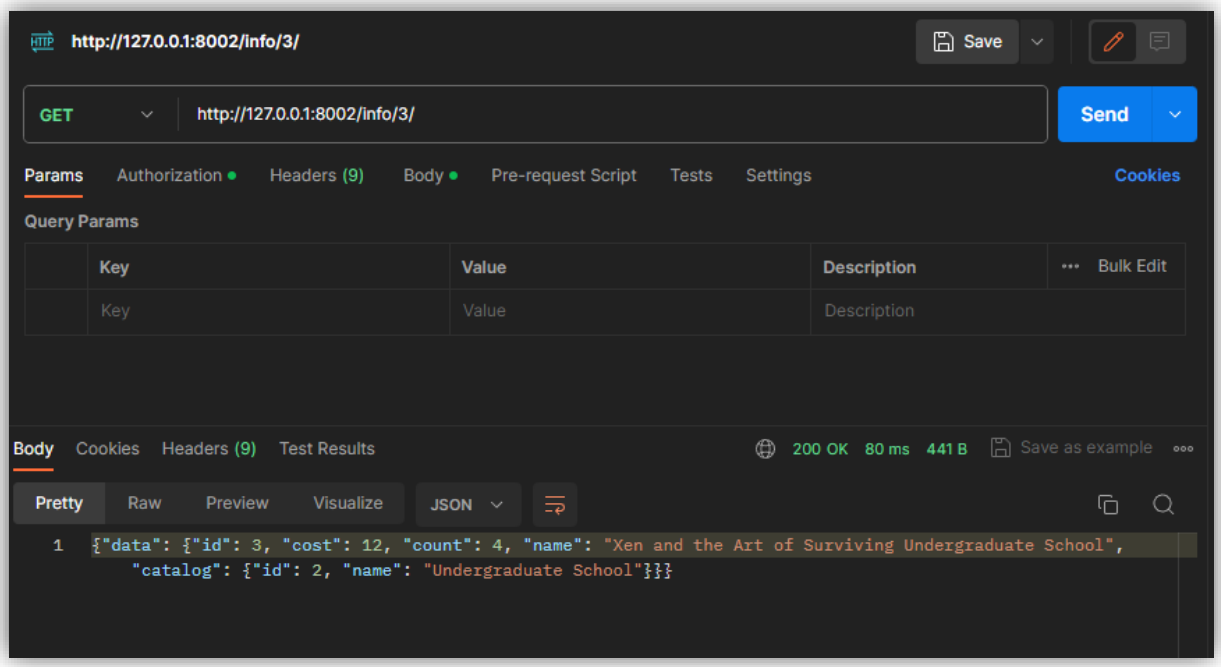
Testing:

First request send to server 1 – 87ms

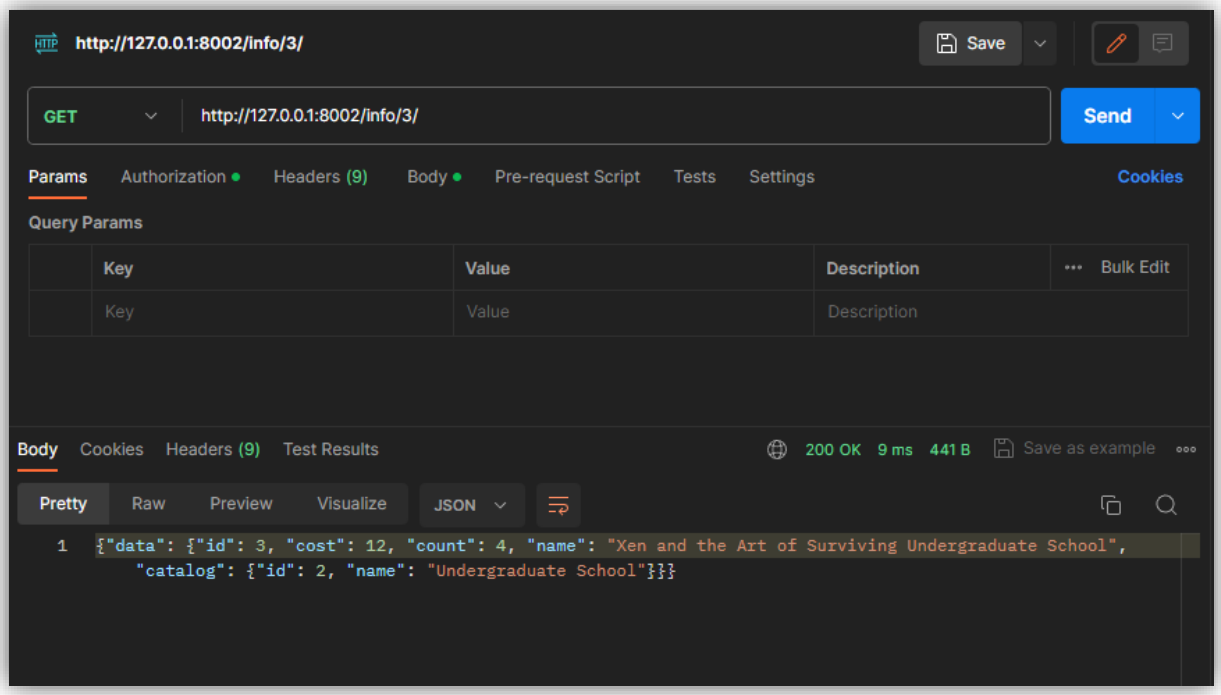


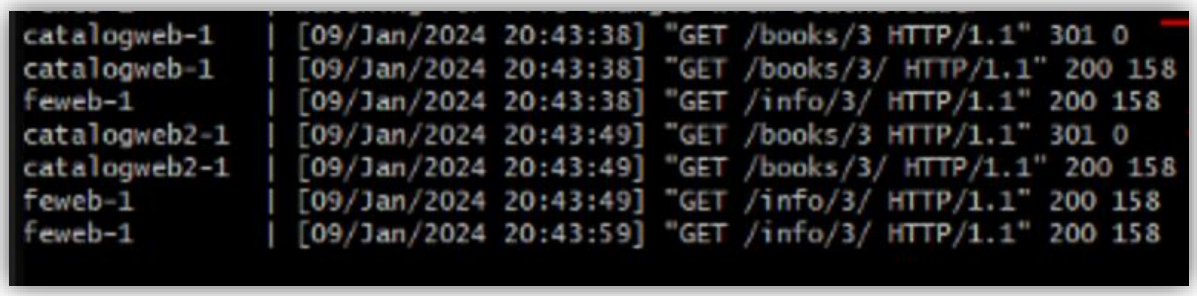
Second request to replica (server 2) – 80ms

Same request again → should send to server 2 to get load balance

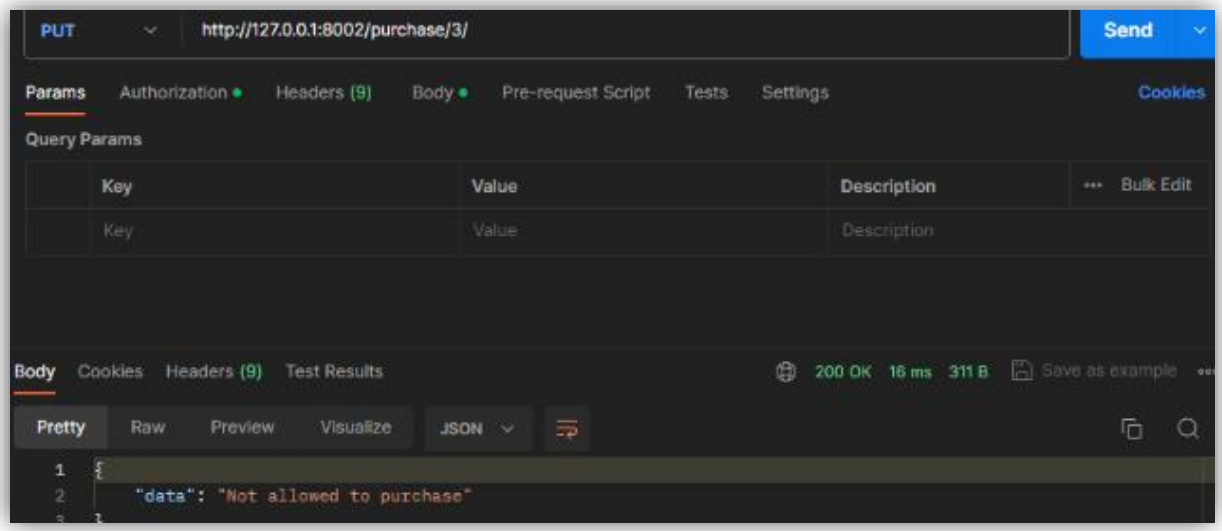


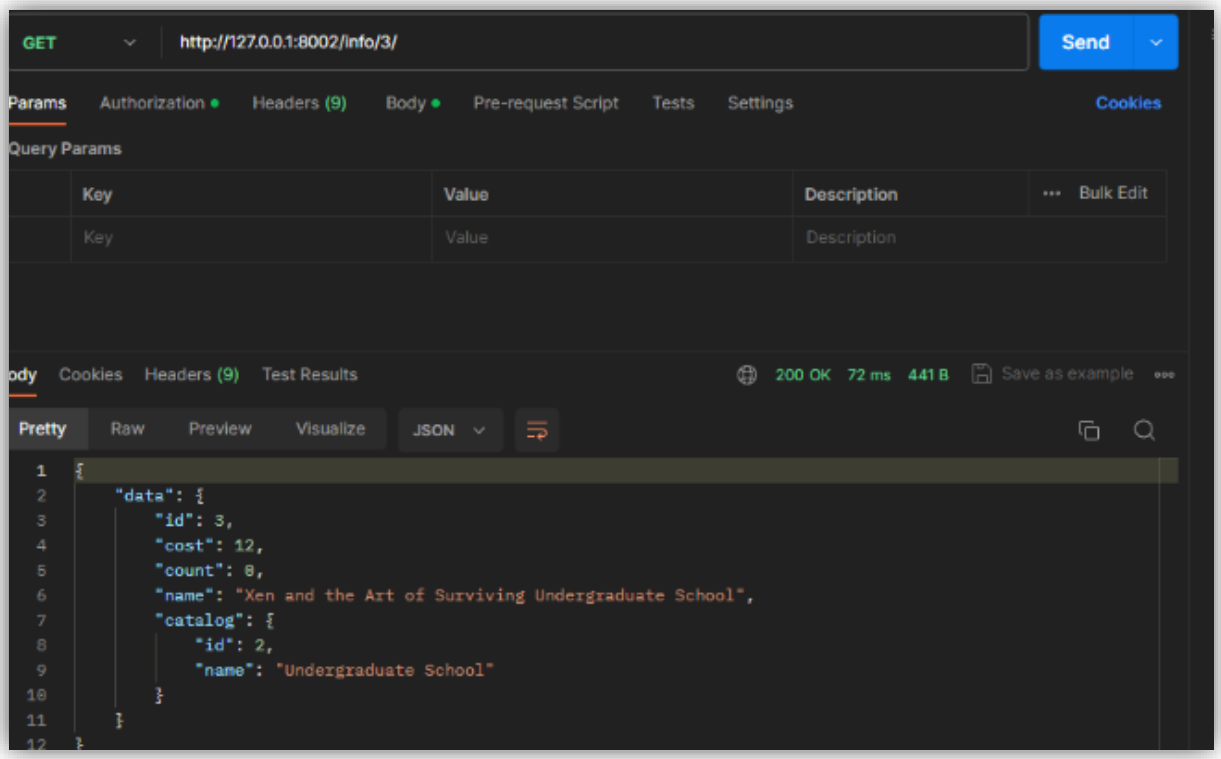
And Last one to cache (does not send to any server) – 9ms !
I noticed that the time needed to get response decreased so the system operates correctly.

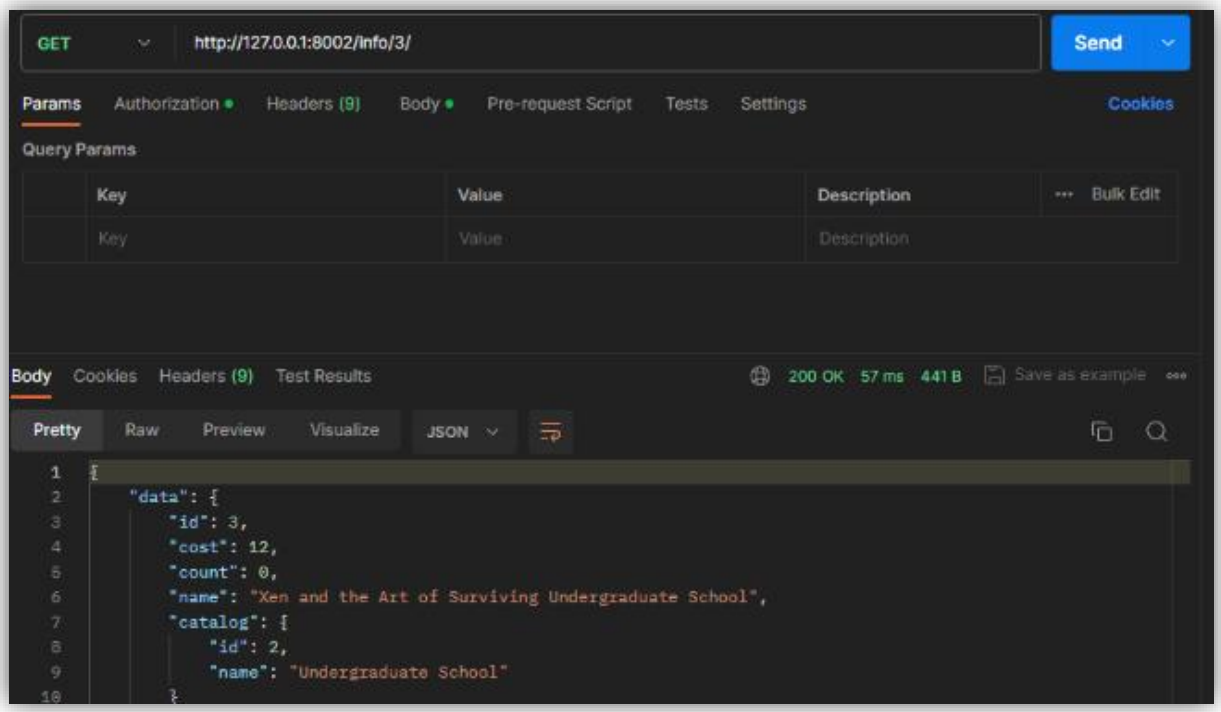


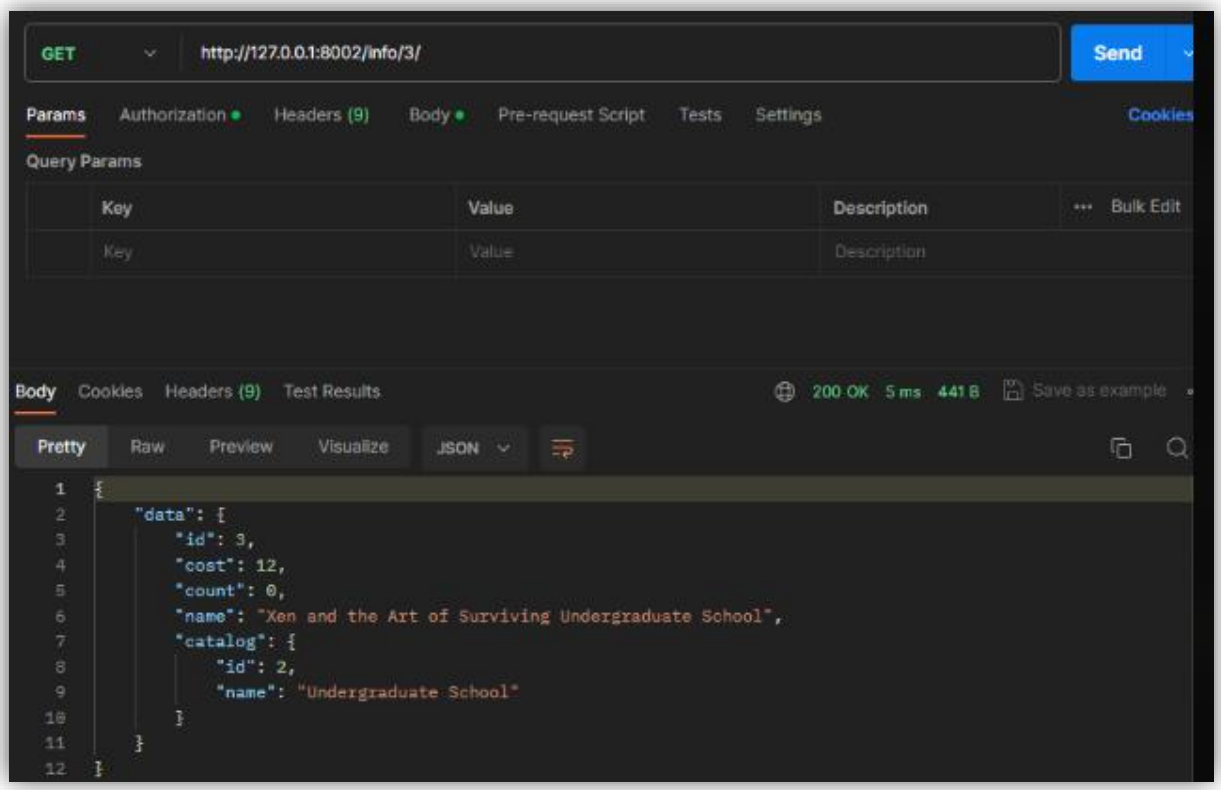


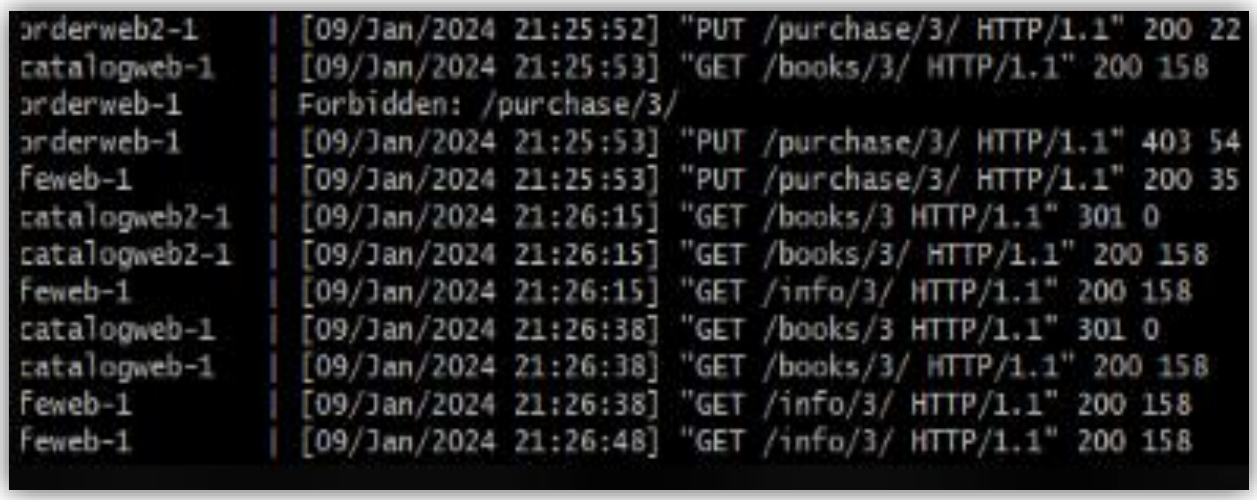
I do many purchase operation until count become 0, and servers should be empty. (consistency achieved)






GET
http://127.0.0.1:8002/info/3/
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (9)
Test Results
200 OK
57 ms
441 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"data": {
"id": 3,
"cost": 12,
"count": 0,
"name": "Xen and the Art of Surviving Undergraduate School",
"catalog": {
"id": 2,
"name": "Undergraduate School"
}


GET
http://127.0.0.1:8002/info/3/
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (9)
Test Results
200 OK
5 ms
441 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"data": {
"id": 3,
"cost": 12,
"count": 0,
"name": "Xen and the Art of Surviving Undergraduate School",
"catalog": {
"id": 2,
"name": "Undergraduate School"
}
}
}


orderweb2-1 | [09/Jan/2024 21:25:52] "PUT /purchase/3/ HTTP/1.1" 200 22
catalogweb-1 | [09/Jan/2024 21:25:53] "GET /books/3/ HTTP/1.1" 200 158
orderweb-1 | Forbidden: /purchase/3/
orderweb-1 | [09/Jan/2024 21:25:53] "PUT /purchase/3/ HTTP/1.1" 403 54
feweb-1 | [09/Jan/2024 21:25:53] "PUT /purchase/3/ HTTP/1.1" 200 35
catalogweb2-1 | [09/Jan/2024 21:26:15] "GET /books/3 HTTP/1.1" 301 0
catalogweb2-1 | [09/Jan/2024 21:26:15] "GET /books/3/ HTTP/1.1" 200 158
feweb-1 | [09/Jan/2024 21:26:15] "GET /info/3/ HTTP/1.1" 200 158
catalogweb-1 | [09/Jan/2024 21:26:38] "GET /books/3 HTTP/1.1" 301 0
catalogweb-1 | [09/Jan/2024 21:26:38] "GET /books/3/ HTTP/1.1" 200 158
feweb-1 | [09/Jan/2024 21:26:38] "GET /info/3/ HTTP/1.1" 200 158
feweb-1 | [09/Jan/2024 21:26:48] "GET /info/3/ HTTP/1.1" 200 158