

Bazar e-commerce API

DOS Project 1

Suhaib Hamdallah	Mohammed Abu-Qasedo
11716086	11715872

In this project, we will build an API for Bazar e-commerce, the API consists of Catalog server, Order server and client application. We will build the API in microservice architecture so we will separate all component.

Catalog Server

The catalog server provides 3 operations for the system that we will see below.

This server contains a database that saves the book information represented in:

1. Book Id
2. Book Title
3. Book topic. Can be (Distributed Systems or Undergraduate School)
4. Book Price
5. Number of books in stock.

The operations the catalog server provides are:

1. Find book by id.
2. Find books by topic.
3. Update book quantity in stock.

In order to implement this microservice we used ASP.NET CORE WEB API framework and Sqlite database.

After implementation finished, we deploy the code in apache server that installed in Ubuntu.

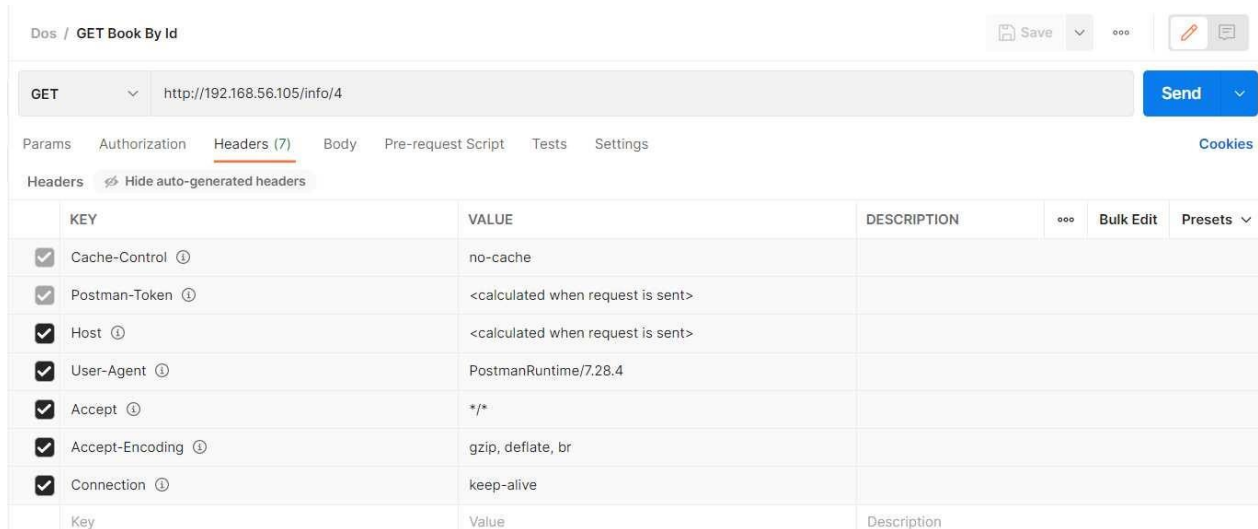
ASP.NET core applications use kestrel server to run in localhost so we use apache server as a reverse proxy server to forward requests to kestrel server.

Test catalog server

To test catalog server we start the virtual machines that host our server and get the local ip for this virtual machines.

1. Test find book by id API

We used Postman for testing



Our API used HTTP Protocol, GET Verb



192.168.56.105 ➔ IP of virtual machine that run apache server in port 80 and the request arrives to apache in port 80 are forwarded to kestrel server that runs in port 5000 in virtual machine.



/info/{id} is the route path for the book with id


No headers is needed in this API




No body in needed



The response:



Dos / GET Book By Id  

GET  http://192.168.56.105/info/4 

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

Key	Value	Description
<input checked="" type="checkbox"/> Accept 	*/*	
<input checked="" type="checkbox"/> Accept-Encoding 	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection 	keep-alive	

Body Cookies Headers (6) Test Results  Status: 200 OK Time: 976 ms Size: 321 B 



Pretty Raw Preview Visualize JSON  



```
1 {
2   "id": 4,
3   "title": "Cooking for the Impatient Undergrad",
4   "topic": "undergraduate school",
5   "numberOfItemsInStock": 8,
6   "price": 10.99
7 }
```


Status code 200 if id is valid and the items in stock in greater than 0.




The response body in JSON Object with intended book properties .



If the id in invalid or the items in stock is 0 then the response will be 404 not found.



Dos / GET Book By Id  

GET  http://192.168.56.105/info/8 

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

Key	Value	Description
<input checked="" type="checkbox"/> Accept 	*/*	
<input checked="" type="checkbox"/> Accept-Encoding 	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection 	keep-alive	

Body Cookies Headers (6) Test Results  Status: 404 Not Found Time: 77 ms Size: 373 B 

Pretty Raw Preview Visualize JSON  

```
1 {
2   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",
3   "title": "Not Found",
4   "status": 404,
5   "traceId": "00-1eceb663136e9e41be056bf833d8e5f4-391f31358bc14241-00"
6 }
```

2. Test find by topic API:

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

- Method:** GET
- URL:** http://192.168.56.105/search/distributed%20systems
- Status:** 200 OK
- Time:** 79 ms
- Size:** 438 B
- Response Format:** JSON
- Response Body:**

```
[{"id": 1, "title": "How to get a good grade in DOS in 40 minutes a day", "topic": "distributed systems", "numberOfItemsInStock": 7, "price": 40.5}, {"id": 2, "title": "RPCs for Noobs", "topic": "distributed systems", "numberOfItemsInStock": 6, "price": 30.5}]
```

Our API used HTTP Protocol, GET Verb

192.168.56.105 ➔ IP of virtual machine that run apache server in port 80 and the request arrives to apache in port 80 are forwarded to kestrel server that runs in port 5000 in virtual machine.

/search/{topic} is the route path for the books with topic

No headers is needed in this API

No body is needed

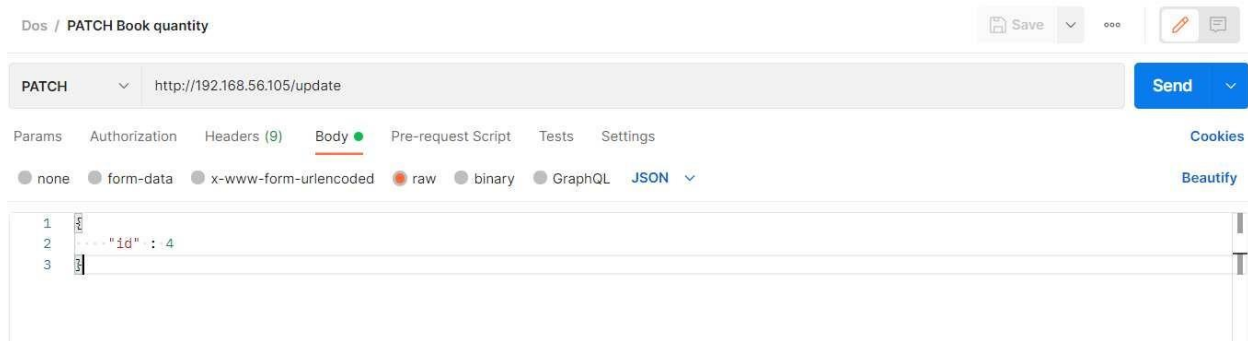
The response is 200 status code with array of books objects with topic specifies in the url.

If there is no books the response will be 200 status code with empty array.

3. Test update book quantity API:

As we see in first testing (Test get book by id API) the book with id 4 have 8 books

So now we update the book by decrements items number by 1.

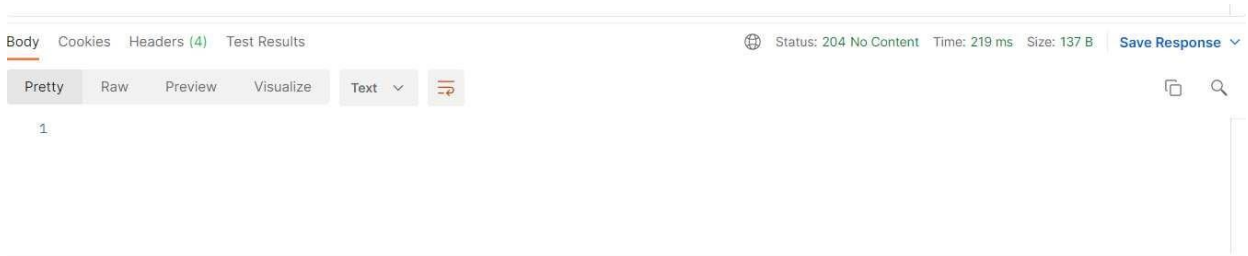


Our API used HTTP Protocol, PATCH Verb → PATCH because we want to modifies only NumberOfItemsInStock property only.

192.168.56.105 → IP of virtual machine that run apache server in port 80 and the request arrives to apache in port 80 are forwarded to kestrel server that runs in port 5000 in virtual machine.

/update is the route path for the update book

Content-type header = application/json because we want to send the id of the book inside request body as json object



The response is (204 success code no content) because we don't return any data. If the book id is invalid or the items = 0 the return code is (400 bad request).

Order server

The order server provides only one operation for the system that we will see below.

This server contains a database that saves the book information represented in:

1. OrderId
2. Book Id
3. Quantity

The operation the catalog server provides is:

1. Purchase an book

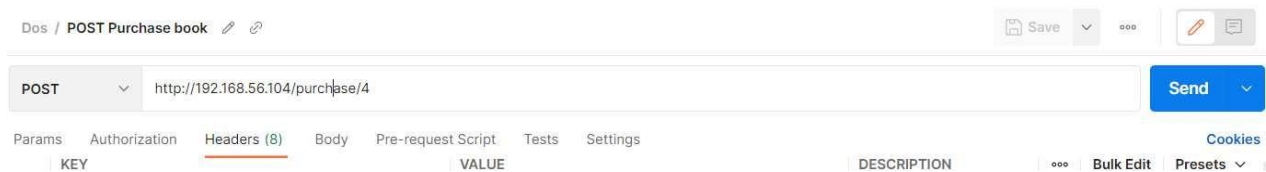
In order to implement this microservice we used ASP.NET CORE WEB API framework and Sqlite database.

After implementation finished, we deploy the code in apache server that installed in Ubuntu.

ASP.NET core applications use kestrel server to run in localhost so we use apache server as a reverse proxy server to forward requests to kestrel server.

Test Order server

1. Purchase API

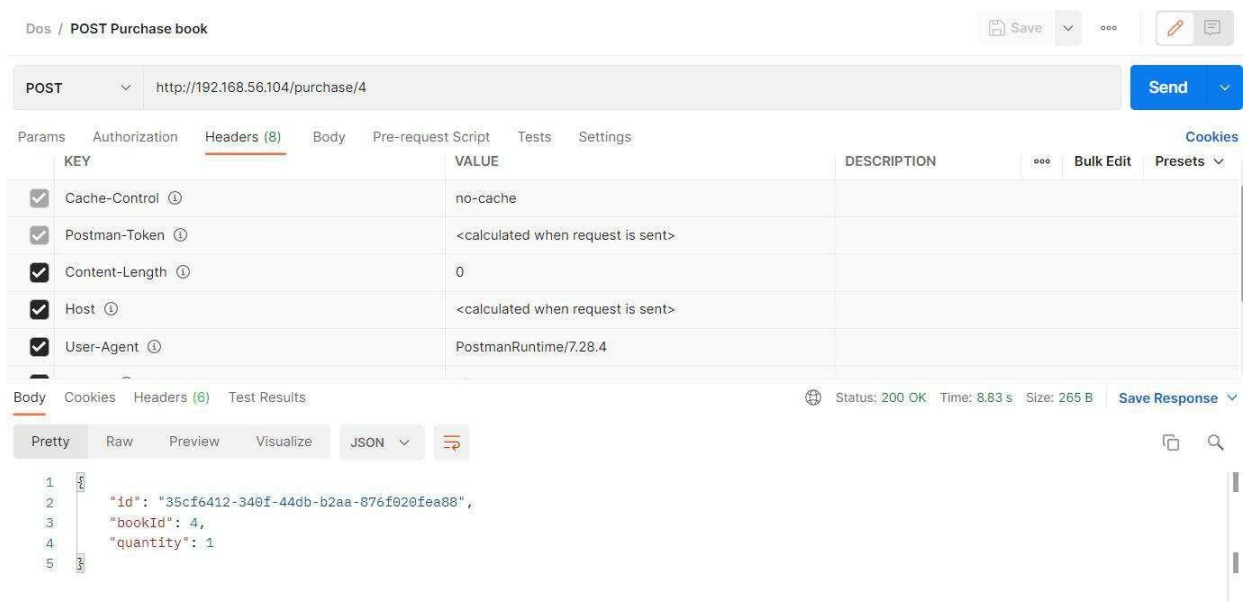


Our API used HTTP Protocol, POST Verb.

192.168.56.104 ➔ IP of virtual machine that run apache server in port 80 and the request arrives to apache in port 80 are forwarded to kestrel server that runs in port 5000 in virtual machine.

/purchase/{id} is the route path for the purchase book

No headers is needed and no request body is needed



Then response when item id is valid and item stocks number > 0 is 200 with response body the new order record in db otherwise is 404 not found

NOTE THAT THE CLIENT WE USED IS POSTMAN