

2checkout

- Engineering Back-end Internship -

Dorobanțu Florin-Claudiu

The main structure of the application consists in:

1. README.pdf – explanations about implementation and testing
2. Makefile – compiles the program
3. util.h – auxiliary (error) definitions
4. w_epoll.h – epoll wrapper functions
5. sock_util.h – connection header
6. sock_util.c – connection functions
7. server.h – server header
8. server.c – server implementation
9. sql – directory containing the database
10. resources – directory containing other resources

I chose to implement the application in C language on the Linux platform. It uses multiplexing mechanisms and asynchronous operations. The essential body is found in the server.c file.

The program contains information about the use of each function through relevant comments in the code. Here I will focus on the important implementation details.

Because the requirements of the problem were quite undetailed, I used my imagination to create my own RESTful API application structure. I solved the tasks (**CRUD, authentication, rate limit**) and I carefully followed the final observations: **working without a framework, using proper HTTP verbs and status codes, error handling**. The implementation is not based on any framework and it is written in C at a low level.

For the following tests, the server is local (localhost) and it is waiting for requests on port 8888 (specified as SERVER_LISTEN_PORT in server.h file). The server is waiting for requests that have a specific HTTP style format. Requests containing the GET method will return certain products from the catalog (read from CRUD). Requests containing the POST method will introduce new products in the catalog (create from CRUD). Requests containing the PUT method will update products from the catalog (update from CRUD). Requests containing the DELETE method will delete products from the catalog (delete from CRUD).

Also, the format of clients requests must be similar to the following form:

```
[METHOD] /sql/product_catalog.db HTTP/1.1\r\n
Host: localhost:8888\r\n
User-Agent: OpenBSD netcat\r\n
Accept: */*\r\n
Authorization: Basic ZG9ybzpwYXNzX2Rvcn8=\r\n
Connection: Keep-Alive\r\n
\r\n
[CONTENT]\r\n
end\r\n\r\n
```

Product catalog data is stored persistently in a database on disk. The database will be created from the program in the sql directory, and it will be called product_catalog.db (more precisely, the path will be sql/product_catalog.db). Also, a table called product_catalog will be created from the program in this database and it will contain the columns Id (the primary key of the table), Name, Price, Category, CreatedDate, UpdatedDate. All clients requests will indirectly query this table, which contains the product catalog. Thus, all requests will be transformed into sqlite3 requests within the program.

The server administrator will add users who will be able to interact with the database. Three members have already been added, being the only ones who can successfully authenticate using the basic authorization field in the http request:

1. user: doro, pass: pass_doro => base64 token: ZG9ybzpwYXNzX2Rvcn8=
2. user: claudiu, pass: pass_claudiu => base64 token: Y2xhdWRpdTpwYXNzX2NsYXVkaXU=
3. user: florin, pass: pass_florin => base64 token: ZmxvcmluOnBhc3NfZmxvcmlu

The request rate limit is updated at every REFRESH_USER_RATE_LIMITING (60) seconds interval (connection_time and number_of_requests are updated) and it is checked every time by the formula $(\text{current_time} - \text{connection_time}) / \text{number_of_requests} \geq \text{RATE_LIMITING} (3)$.

For client requests and server responses, I kept a specific HTTP format. Also, for all cases, I kept the status codes (number and message) according to the HTTP status codes, both for the success cases and for the error cases.

At the same time, I tried to handle in code many of the errors that can occur during the operation of the application.

For a correct request, the response from the server to the client will be similar to the following form:

```
HTTP/1.1 200 OK\r\n
Content-Length: 140\r\n
Connection: close\r\n
Content-Type: text/plain\r\n
\r\n
[[CONTENT ONLY FOR THE GET METHOD – LIST OF PRODUCTS]]\r\n
\r\n
[METHOD] done ([MESSAGE FROM THE DATABASE – SUCCESS OR A SPECIFIC ERROR]).\r\n
\r\n
```

For a wrong request, the response from the server to the client will be similar to the following form:

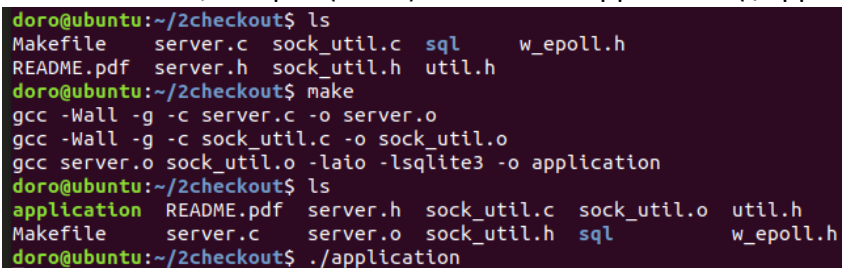
```
HTTP/1.1 [STATUS CODE ERROR]\r\n
\r\n
```

Below, I will present a series of examples for testing the application.

Firstly, make sure you can use libaio.h (used for asynchronous operations) and libsqlite3 (used in working with the database). Otherwise, install them:

```
sudo apt-get install libaio-dev
sudo apt-get install libsqlite3-dev
```

In one terminal, compile (make) and run the application (./application) (this will be the server):



```
doro@ubuntu:~/2checkout$ ls
Makefile  server.c  sock_util.c  sql      w_epoll.h
README.pdf server.h  sock_util.h  util.h
doro@ubuntu:~/2checkout$ make
gcc -Wall -g -c server.c -o server.o
gcc -Wall -g -c sock_util.c -o sock_util.o
gcc server.o sock_util.o -laio -lsqlite3 -o application
doro@ubuntu:~/2checkout$ ls
application README.pdf server.h  sock_util.c  sock_util.o  util.h
Makefile    server.c  server.o  sock_util.h  sql          w_epoll.h
doro@ubuntu:~/2checkout$ ./application
```

As an observation, to clean the solution (remove object and executable files), use: make clean.

In another terminal, you can check that the database has been created in the sql directory. Also, an empty table of products was created in this database.

```
doro@ubuntu:~/2checkout$ cd sql/
doro@ubuntu:~/2checkout/sql$ ls
product_catalog.db
doro@ubuntu:~/2checkout/sql$ sqlite3 product_catalog.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> .tables
product_catalog
sqlite> .schema product_catalog
CREATE TABLE product_catalog(Id NUMBER(5) PRIMARY KEY,Name VARCHAR2(30),Price
NUMBER(5,2),Category VARCHAR2(30),CreatedDate DATE,UpdatedDate DATE);
sqlite> .header on
sqlite> .mode column
sqlite> pragma table_info('product_catalog');
cid      name      type      notnull    dflt_value  pk
-----
0         Id        NUMBER(5) 0          0           1
1         Name      VARCHAR2(30) 0          0           0
2         Price     NUMBER(5,2) 0          0           0
3         Category  VARCHAR2(30) 0          0           0
4         CreatedDat DATE        0          0           0
5         UpdatedDat DATE        0          0           0
sqlite> SELECT * FROM product_catalog;
sqlite>
```

In other terminals, send requests from clients. I used netcat clients. Below are some examples.

1. **POST method** – the content must have the following form: Id,Name,Price,Category.

```
-----
POST /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2Rvcn8=
Connection: Keep-Alive
```

```
6,Shirt,19.99,Clothes
end
```

```
-----
echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-
Alive\r\n\r\n6,Shirt,19.99,Clothes\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout$ echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localho
st:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNz
X2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n6,Shirt,19.99,Clothes\r\nend\r\n\r\n" | nc -q 1
localhost 8888 > response.txt
doro@ubuntu:~/2checkout$ cat response.txt
HTTP/1.1 200 OK
Content-Length: 22
Connection: close
Content-Type: text/plain

POST done (success).

doro@ubuntu:~/2checkout$
```

```
doro@ubuntu:~/2checkout$ ./application
POST /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2RvcM8=
Connection: Keep-Alive

6,Shirt,19.99,Clothes
end

sqlite> SELECT * FROM product_catalog;
Id      Name      Price      Category    CreatedDate      UpdatedDate
-----
6       Shirt      19.99      Clothes     09-06-2020 01:12:47  09-06-2020 01:12:47
sqlite>
```

The server added the creation date and the last update date according to the current date.

2. **PUT method:** the content must have the following form: Id,Name,Price,Category.

```
-----
PUT /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2RvcM8=
Connection: Keep-Alive
```

```
6,Suit,29.99,Clothes
end
```

```
-----
echo -ne "PUT /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9yb3pwYXNzX2RvcM8=\r\nConnection: Keep-
Alive\r\n\r\n6,Suit,29.99,Clothes\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout$ echo -ne "PUT /sql/product_catalog.db HTTP/1.1\r\nHost: localhos
t:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX
2RvcM8=\r\nConnection: Keep-Alive\r\n\r\n6,Suit,29.99,Clothes\r\nend\r\n\r\n" | nc -q 1 l
ocalhost 8888 > response.txt
doro@ubuntu:~/2checkout$ cat response.txt
HTTP/1.1 200 OK
Content-Length: 21
Connection: close
Content-Type: text/plain

PUT done (success).
doro@ubuntu:~/2checkout$
```

```
doro@ubuntu:~/2checkout$ ./application
PUT /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2Rvcn8=
Connection: Keep-Alive
```

```
6,Suit,29.99,Clothes
end
```

```
sqlite> SELECT * FROM product_catalog;
-----
Id      Name      Price      Category    CreatedDate      UpdatedDate
-----
6       Suit      29.99      Clothes     09-06-2020 01:12:47  09-06-2020 01:26:07
sqlite>
```

The product with the specified Id has been updated with the new parameters (name, price). Also, the last update date has been changed according to the current date.

3. **GET method:** the content must have the following form: Id or 'all'.

The first version will return the data about the product with the specified Id.

```
-----
GET /sql/product_catalog.db HTTP/1.1
```

```
Host: localhost:8888
```

```
User-Agent: OpenBSD netcat
```

```
Accept: */*
```

```
Authorization: Basic ZG9yb3pwYXNzX2Rvcn8=
```

```
Connection: Keep-Alive
```

```
6
```

```
end
```

```
-----
echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1
localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout$ echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhos
t:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX
2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > res
ponse.txt
```

```
doro@ubuntu:~/2checkout$ cat response.txt
```

```
HTTP/1.1 200 OK
Content-Length: 142
Connection: close
Content-Type: text/plain
```

```
Id = 6
Name = Suit
Price = 29.99
Category = Clothes
CreatedDate = 09-06-2020 01:33:56
UpdatedDate = 09-06-2020 02:06:19
```

```
GET done (success).
```

```
doro@ubuntu:~/2checkout$
```

```
doro@ubuntu:~/2checkout$ ./application
GET /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2Rvcn8=
Connection: Keep-Alive

6
end
```

The second version will return the data about all products in the catalog.

```
-----
GET /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2Rvcn8=
Connection: Keep-Alive
```

```
all
end
```

```
-----
echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\nall\r\nend\r\n\r\n" | nc -q 1
localhost 8888 > response.txt
```

4. **DELETE method:** the content must have the following form: Id.

```
-----
DELETE /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2Rvcn8=
Connection: Keep-Alive
```

```
6
end
```

```
echo -ne "DELETE /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout$ echo -ne "DELETE /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
doro@ubuntu:~/2checkout$ cat response.txt
HTTP/1.1 200 OK
Content-Length: 24
Connection: close
Content-Type: text/plain

DELETE done (success).

doro@ubuntu:~/2checkout$
```

```
doro@ubuntu:~/2checkout$ ./application
DELETE /sql/product_catalog.db HTTP/1.1
Host: localhost:8888
User-Agent: OpenBSD netcat
Accept: */*
Authorization: Basic ZG9yb3pwYXNzX2Rvcn8=
Connection: Keep-Alive

6
end
```

```
doro@ubuntu:~/2checkout/sql$ sqlite3 product_catalog.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> SELECT * FROM product_catalog;
sqlite>
```

5. Incorrect database operations

There are also cases in which the request format is correct, but the desired operation on the database is not correct. The error will appear in response. For example, inserting a product with the same Id as another existing product (primary key constraint).

```
doro@ubuntu:~/2checkout/sql$ echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n6,Shirt,19.99,Clothes\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
doro@ubuntu:~/2checkout/sql$ cat response.txt
HTTP/1.1 200 OK
Content-Length: 22
Connection: close
Content-Type: text/plain

POST done (success).

doro@ubuntu:~/2checkout/sql$ echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n6,Suit,29.99,Clothes\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
doro@ubuntu:~/2checkout/sql$ cat response.txt
HTTP/1.1 200 OK
Content-Length: 59
Connection: close
Content-Type: text/plain

POST done (UNIQUE constraint failed: product_catalog.Id).

doro@ubuntu:~/2checkout/sql$
```


6. Incorrect request format

```
echo -ne "METHOD /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9ybzwYXNzX2RvcM8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout/sql$ echo -ne "METHOD /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9ybzwYXNzX2RvcM8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
doro@ubuntu:~/2checkout/sql$ cat response.txt
HTTP/1.1 400 Bad Request
doro@ubuntu:~/2checkout/sql$
```

7. Unauthorized user

```
echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic 12345678\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout/sql$ echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic 12345678\r\n\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
doro@ubuntu:~/2checkout/sql$ cat response.txt
HTTP/1.1 401 Unauthorized
doro@ubuntu:~/2checkout/sql$
```

8. Request rate above the allowed limit (for example, repeat the request several times)

```
echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic 12345678\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout$ echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9ybzwYXNzX2RvcM8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
doro@ubuntu:~/2checkout$ cat response.txt
HTTP/1.1 429 Too Many Requests
doro@ubuntu:~/2checkout$
```

9. Resource not found

```
echo -ne "GET /sql/non_existent_database.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic 12345678\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout$ echo -ne "GET /sql/non_existent_database.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9ybzpWYXNzX2RvcM8=\r\nConnection: Keep-Alive\r\n\r\n6\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
doro@ubuntu:~/2checkout$ cat response.txt
HTTP/1.1 404 Not Found

doro@ubuntu:~/2checkout$
```

In my application, I left the database created and populated with the entries below for testing help. But the database can be deleted at any time, because when the server starts, a new one will be created if it does not already exist. The example is also in the resources directory.

```
echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-
Alive\r\n\r\n0,Computer,999.99,IT\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n1,Laptop,499.99,IT\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwYXNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\n2,Banana,1.11,Food\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9ybzpwYXNzX2Rvcn8=\r\nConnection: Keep-
Alive\r\n\r\n3,Orange,2.22,Food\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9yb3pwYXNzX2Rvc m8=\r\nConnection: Keep-Alive\r\n\r\n4,C Programming
Language,10.00,Book\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
echo -ne "POST /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic
ZG9ybzpwYXNzX2Rvc m8=\r\nConnection: Keep-Alive\r\n\r\n5,Design
Patterns,12.00,Book\r\nend\r\n\r\n" | nc -q 1 localhost 8888 > response.txt
```

```
echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localhost:8888\r\nUser-Agent:
OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwY
XNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\nall\r\nend\r\n\r\n" | nc -q 1
localhost 8888 > response.txt
```

```
doro@ubuntu:~/2checkout$ echo -ne "GET /sql/product_catalog.db HTTP/1.1\r\nHost: localh
ost:8888\r\nUser-Agent: OpenBSD netcat\r\nAccept: */*\r\nAuthorization: Basic ZG9yb3pwY
XNzX2Rvcn8=\r\nConnection: Keep-Alive\r\n\r\nall\r\nend\r\n\r\n" | nc -q 1 localhost 88
88 > response.txt
doro@ubuntu:~/2checkout$ cat response.txt
HTTP/1.1 200 OK
Content-Length: 758
Connection: close
Content-Type: text/plain

Id = 0
Name = Computer
Price = 999.99
Category = IT
CreateDate = 09-06-2020 03:19:41
UpdatedDate = 09-06-2020 03:19:41

Id = 1
Name = Laptop
Price = 499.99
Category = IT
CreateDate = 09-06-2020 03:19:55
UpdatedDate = 09-06-2020 03:19:55

Id = 2
Name = Banana
Price = 1.11
Category = Food
CreateDate = 09-06-2020 03:20:07
UpdatedDate = 09-06-2020 03:20:07

Id = 3
Name = Orange
Price = 2.22
Category = Food
CreateDate = 09-06-2020 03:20:19
UpdatedDate = 09-06-2020 03:20:19

Id = 4
Name = C Programming Language
Price = 10
Category = Book
CreateDate = 09-06-2020 03:20:29
UpdatedDate = 09-06-2020 03:20:29

Id = 5
Name = Design Patterns
Price = 12
Category = Book
CreateDate = 09-06-2020 03:20:39
UpdatedDate = 09-06-2020 03:20:39

GET done (success).
doro@ubuntu:~/2checkout$
```

The RESTful API application could be extended with more functionalities, but for now, these are the main points of my application written in C on the Linux platform, following the tasks (**CRUD, authentication, rate limit**) and the final observations of the problem statement: **working without a framework, using proper HTTP verbs and status codes, error handling.**