Before we start with listing the output scenarios, here is a list of the IPs of the servers in our system:

Front-end server: 192.168.1.107:5000
Catalog server: 192.168.1.108:5000
Order server: 192.168.1.114:5000
Catalog server 2: 192.168.1.109:5000
Order server 2: 192.168.1.115:5000

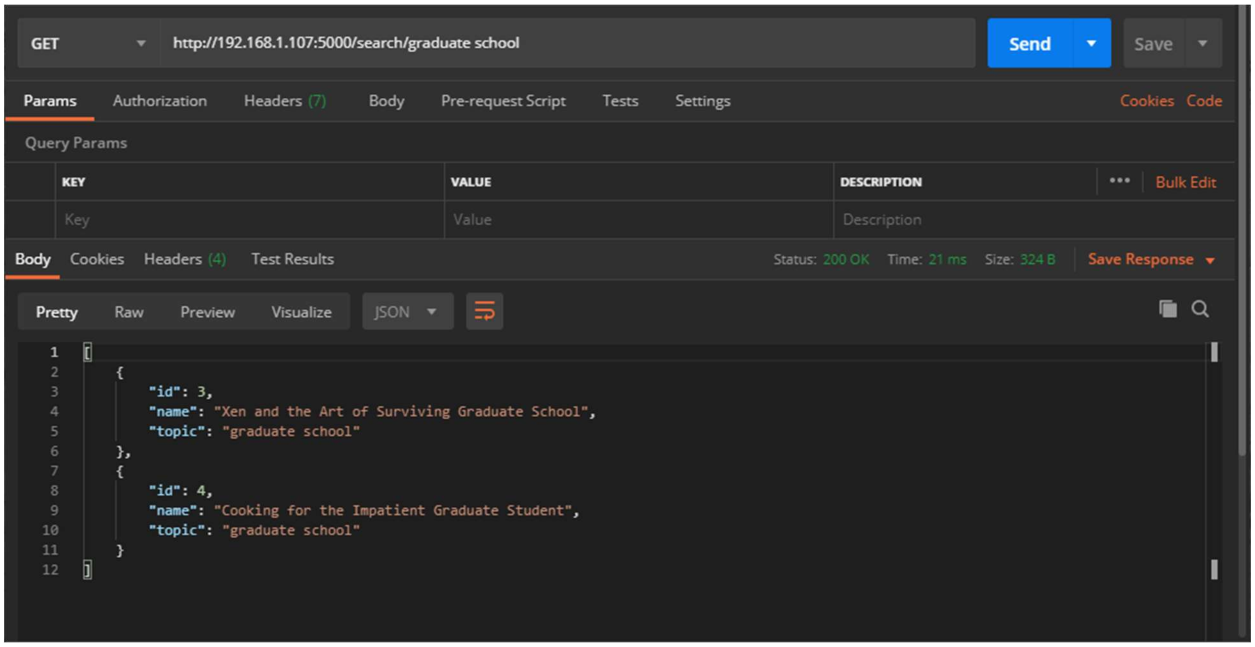
The following are some outputs of some scenarios that I have created to test if the new features work probably

But before that let’s check the outputs of the previous part.

Scenario 1: Sending one http request to get a list of books with a specific topic.

Request: http://192.168.1.107:5000/search/graduate school

Output:



http://192.168.1.107:5000/search/graduate school

Status: 200 OK Time: 21 ms Size: 324 B

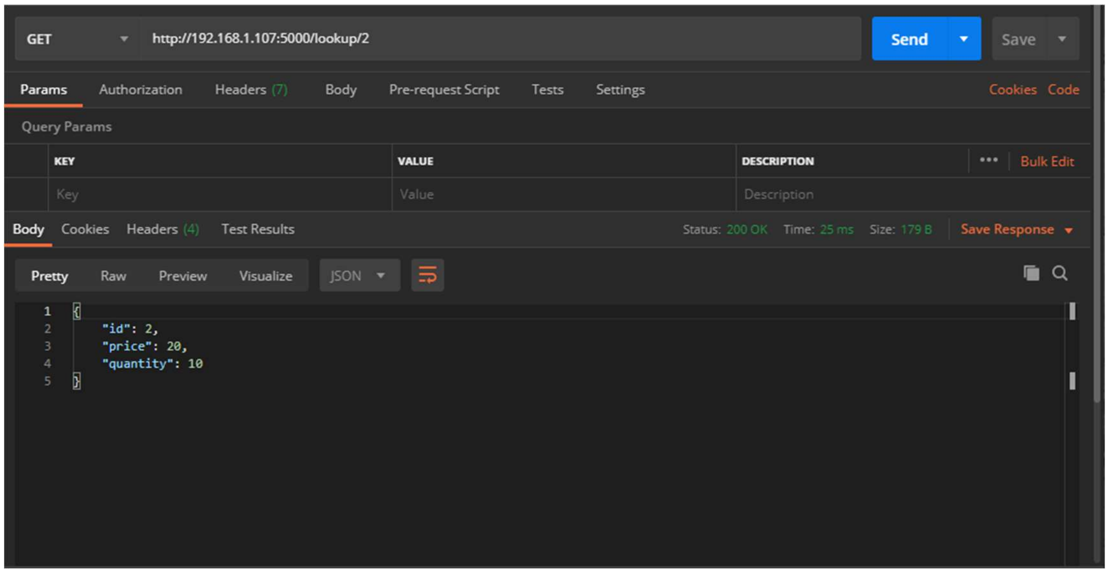
```
[
    {
        "id": 3,
        "name": "Xen and the Art of Surviving Graduate School",
        "topic": "graduate school"
    },
    {
        "id": 4,
        "name": "Cooking for the Impatient Graduate Student",
        "topic": "graduate school"
    }
]
```

Figure 1: Search topic output

Scenario 2: Sending one http request to get a book with a specific id.

Request: http://192.168.1.107:5000/lookup/2

Output:



http://192.168.1.107:5000/lookup/2

Status: 200 OK Time: 25 ms Size: 179 B

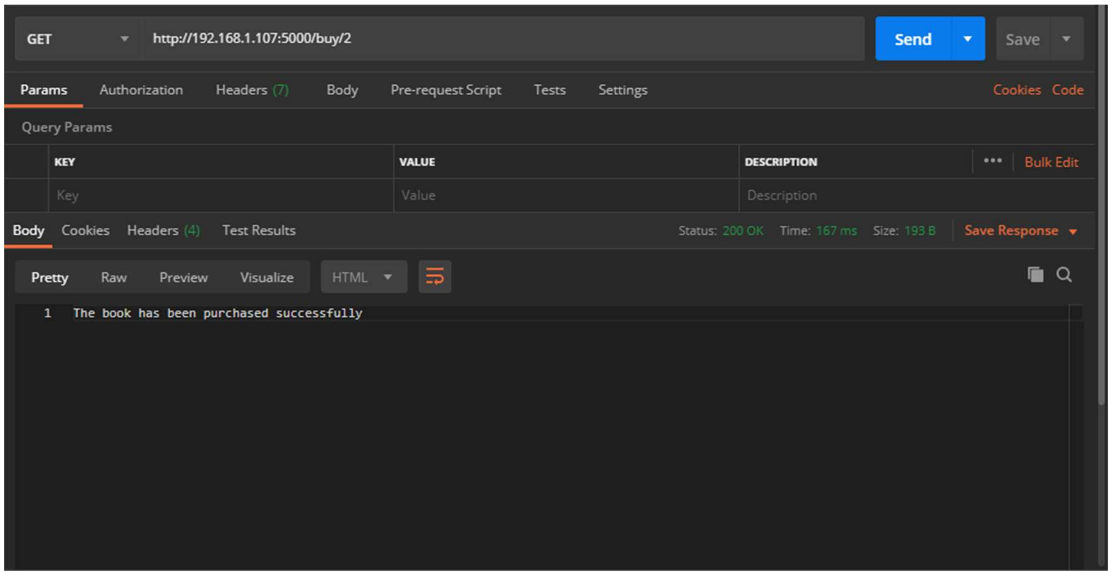
```
{
    "id": 2,
    "price": 20,
    "quantity": 10
}
```

Figure 2: Lookup id output

Scenario 3: Sending one http request to buy a book with a specific id.

Request: http://192.168.1.107:5000/buy/2

Output:



http://192.168.1.107:5000/buy/2

Status: 200 OK Time: 167 ms Size: 193 B

```
The book has been purchased successfully
```

Figure 3: Buy output

Now let’s show the new features.

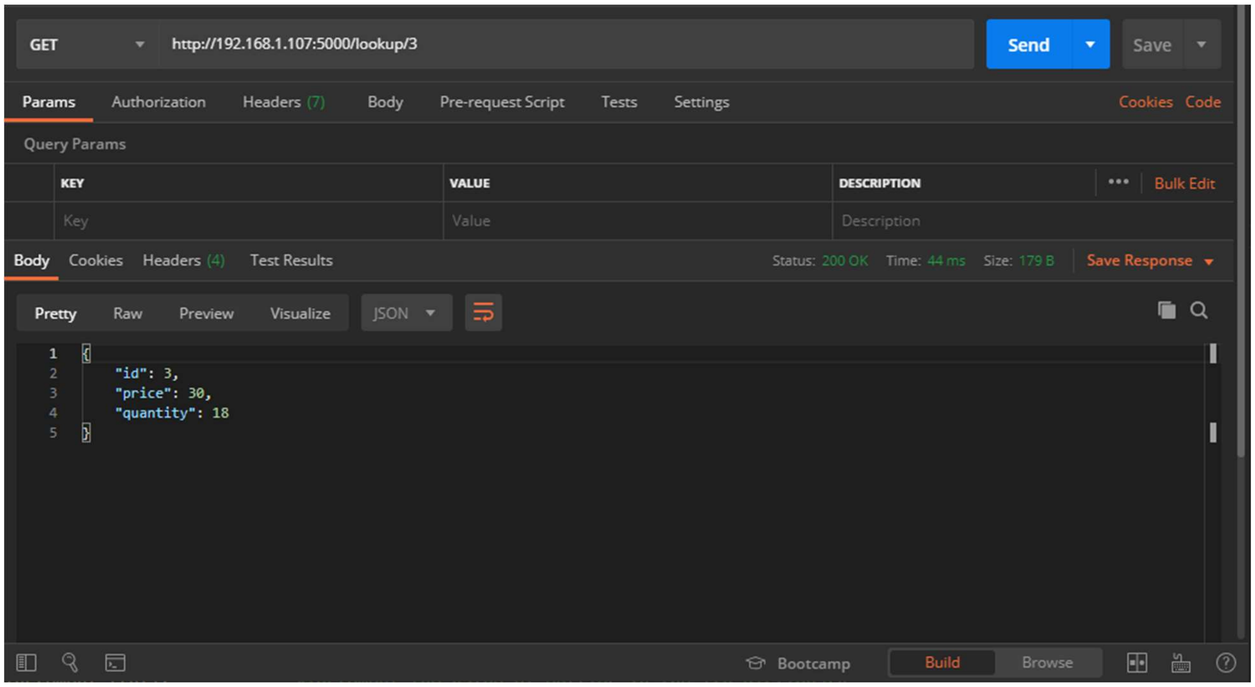
Scenario 4: testing load balancing, sending two “lookup” requests for different ids.

Request1: http://192.168.1.107:5000/lookup/3

Request2: http://192.168.1.107:5000/lookup/4

The first request will look up for book with id 3, we have an output from Postman, and from the terminal of the first catalog server

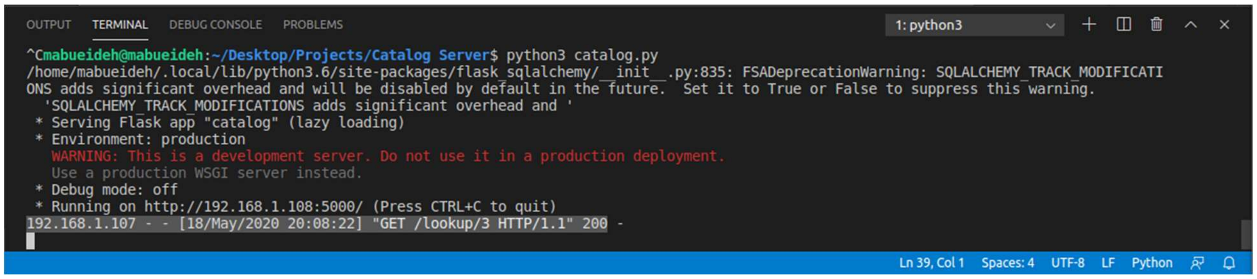
Postman output:



```
GET http://192.168.1.107:5000/lookup/3
Status: 200 OK   Time: 44 ms   Size: 179 B
{
    "id": 3,
    "price": 30,
    "quantity": 18
}
```

Figure 4: Lookup request for id 3 output - Load balancing testing

Catalog Server 1 output:



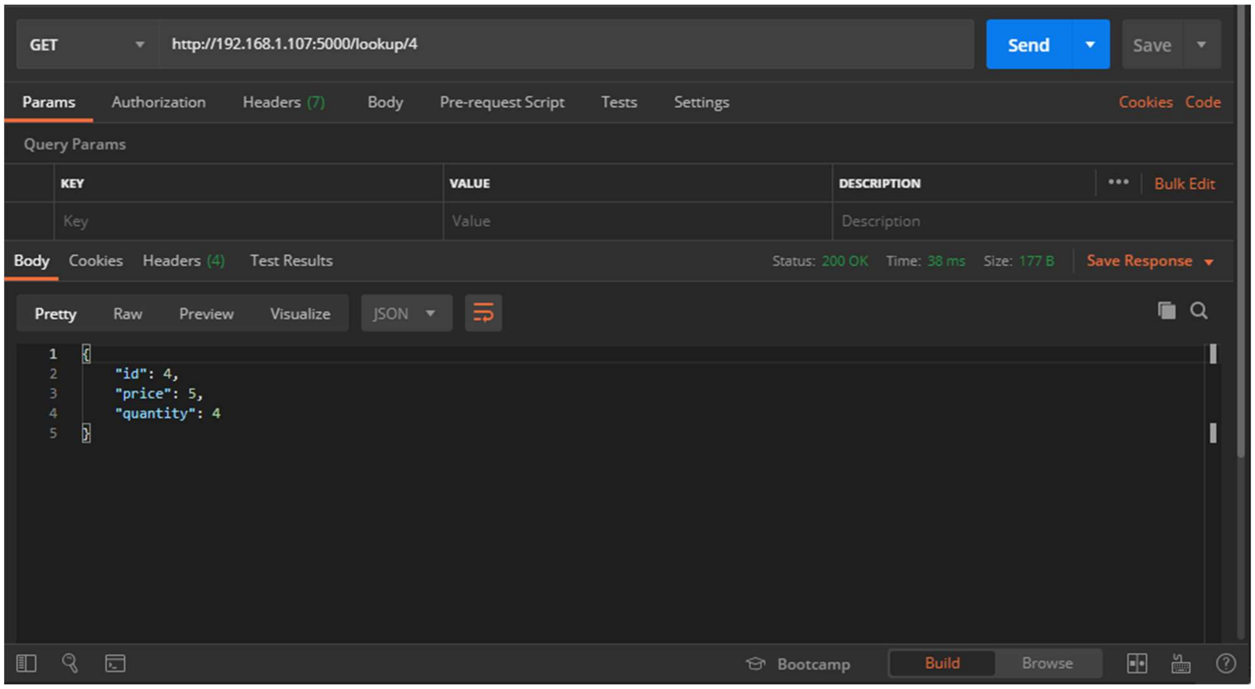
```
^Cmabueideh@mabueideh:~/Desktop/Projects/Catalog Server$ python3 catalog.py
/home/mabueideh/.local/lib/python3.6/site-packages/flask_sqlalchemy/__init__.py:835: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATI
ONS adds significant overhead and will be disabled by default in the future.  Set it to True or False to suppress this warning.
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
 * Serving Flask app "catalog" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://192.168.1.108:5000/ (Press CTRL+C to quit)
192.168.1.107 - - [18/May/2020 20:08:22] "GET /lookup/3 HTTP/1.1" 200 -
```

Figure 5 Catalog server 1 terminal output - Load balancing testing,

Notice the highlighted line which resembles the http request, and from the IP address you can see that this is catalog server 1.

The second request will look up for book with id 4, we have an output from Postman, and from the terminal of the second catalog server

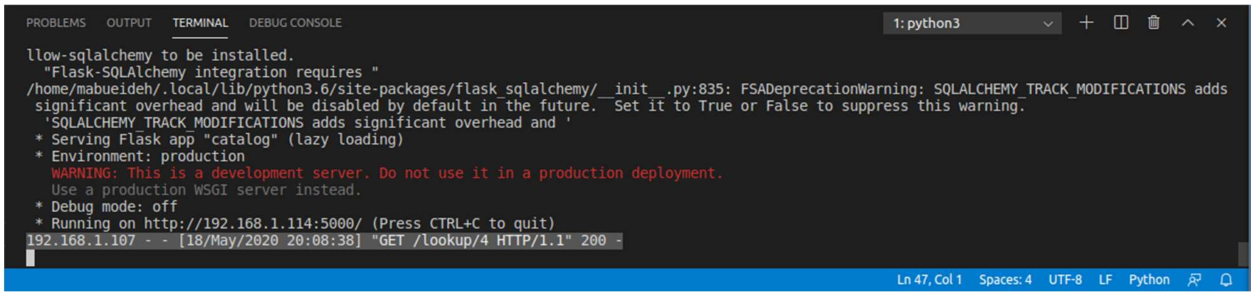
Postman output:



```
{
    "id": 4,
    "price": 5,
    "quantity": 4
}
```

Figure 6: Lookup request for id 4 output - Load balancing testing

Catalog Server 2 output:



```
llow-sqlalchemy to be installed.
  "Flask-SQLAlchemy integration requires "
/home/mabueideh/.local/lib/python3.6/site-packages/flask_sqlalchemy/__init__.py:835: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds
 significant overhead and will be disabled by default in the future.  Set it to True or False to suppress this warning.
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
 * Serving Flask app "catalog" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://192.168.1.114:5000/ (Press CTRL+C to quit)
192.168.1.107 - - [18/May/2020 20:08:38] "GET /lookup/4 HTTP/1.1" 200 -
```

Figure 7: Catalog server 2 terminal output - Load balancing testing

Notice the highlighted line which resembles the http request, and from the IP address you can see that this is catalog server 2.

As you can see each of the catalog servers handled only one request, which means that the requests were handled in a round-robin fashion, which achieves the load balancing mechanism we want

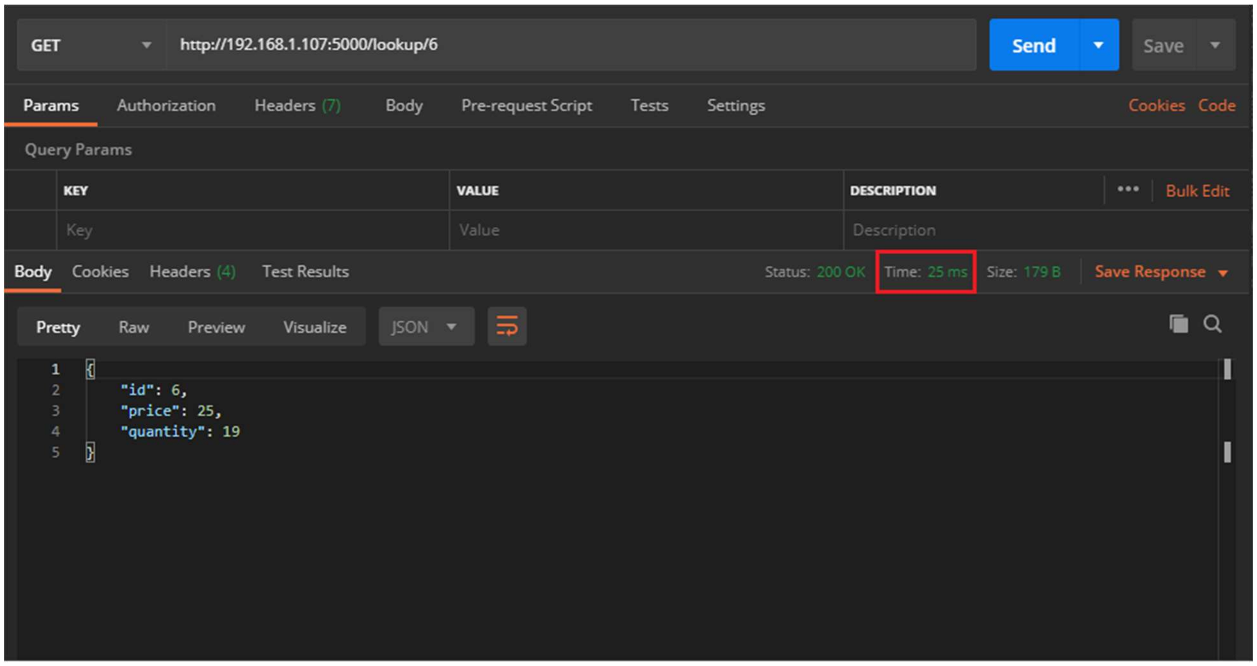
Scenario 5: testing caching, this will consist of two parts. Seeing the latency difference which will be the job of the first two requests, and keeping the cache contents valid which will be the job of last two requests.

Request1: http://192.168.1.107:5000/lookup/3

Request2: http://192.168.1.107:5000/lookup/3

The first request will look up for book with id 3, this will be a cache miss and will take a lot of time, we have an output from Postman which is the contents of book.

Output:



http://192.168.1.107:5000/lookup/6

Status: 200 OK Time: 25 ms Size: 179 B

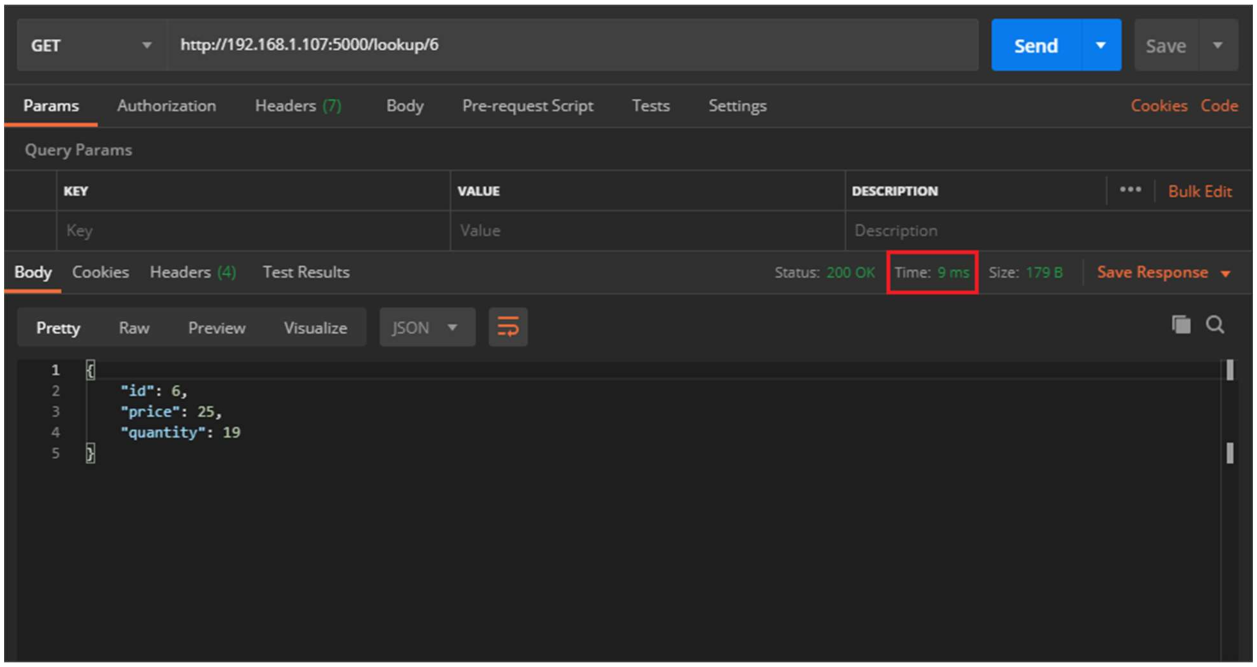
```
{
    "id": 6,
    "price": 25,
    "quantity": 19
}
```

Figure 8: First lookup request output - Cache testing

Notice that it took 25 ms to get the book for the first time, since we needed to go to the catalog server to get it.

The second request will also look up for book with id 3, but this time it will be a cache hit and will take a shorter amount of time, we have an output from Postman which is the contents of book.

Output:



http://192.168.1.107:5000/lookup/6

Status: 200 OK Time: 9 ms Size: 179 B

```
{
    "id": 6,
    "price": 25,
    "quantity": 19
}
```

Figure 9 Second lookup request output - Cache testing.png

Notice that it took 9 ms to get the book, which is much faster than before. Which proves that the cache works.

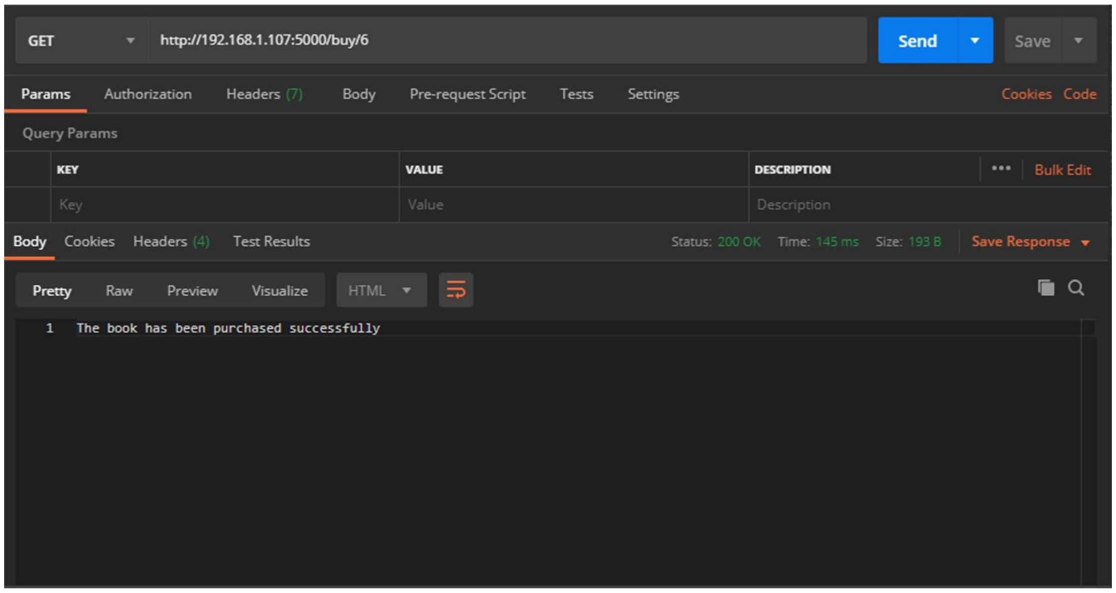
Now let's check the consistency of the cache using these requests.

Request3: http://192.168.1.107:5000/buy/3

Request4: http://192.168.1.107:5000/lookup/3

The third request asks to buy book with id 3, which will cause the catalog server to send an "invalidate" request which will cause the book with id 3 to be flushed from the cache, , we have an output from Postman which is the success message of buying the book.

Output:



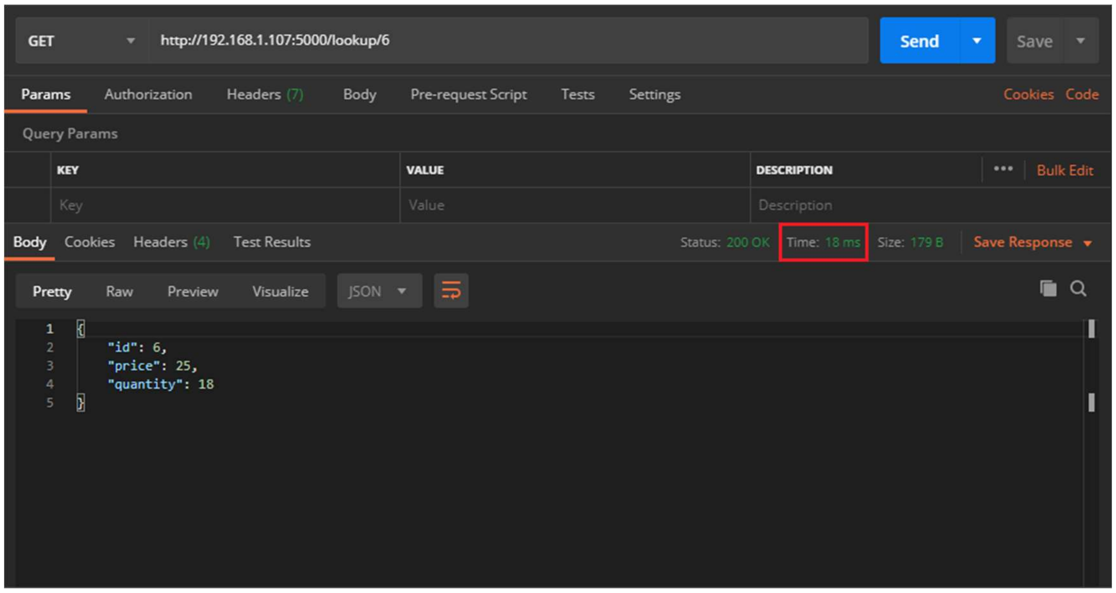
http://192.168.1.107:5000/buy/6

Status: 200 OK Time: 145 ms Size: 193 B

```
The book has been purchased successfully
```

Figure 10: Buy request output - Cache testing

Now the fourth request will look up for book with id 3 again, but since a “buy” request has been initiated for this book, this will be a cache miss and will take a lot of time, we have an output from Postman which is the contents of book.
Output:



http://192.168.1.107:5000/lookup/6

Status: 200 OK Time: 18 ms Size: 179 B

```
{
    "id": 6,
    "price": 25,
    "quantity": 18
}
```

Figure 11 Third lookup request output - Cache testing

As you can see, it took 18 ms to process the last request which means that the entry isn’t in the cache any more.

Scenario 6: testing database consistency, to check the consistency between two data bases correctly I am going to run the front-end server that doesn't have a cache consistency. We are going to run a list of requests that will prove that the database will stay consistent.

Request1: http://192.168.1.107:5000/lookup/1

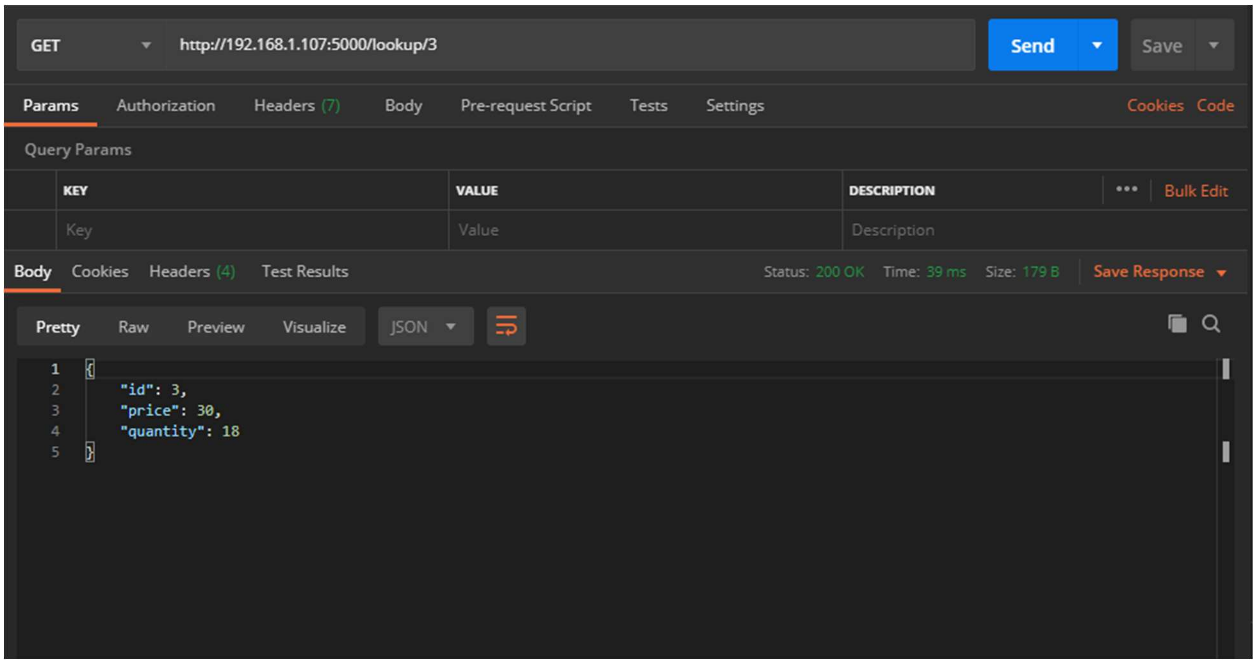
Request2: http://192.168.1.107:5000/buy/1

Request3: http://192.168.1.107:5000/lookup/1

Request4: http://192.168.1.107:5000/lookup/1

The first request book will get book with id 1, this request purpose is to know the quantity and compare it with the end result.

Output:



http://192.168.1.107:5000/lookup/3

Status: 200 OK Time: 39 ms Size: 179 B

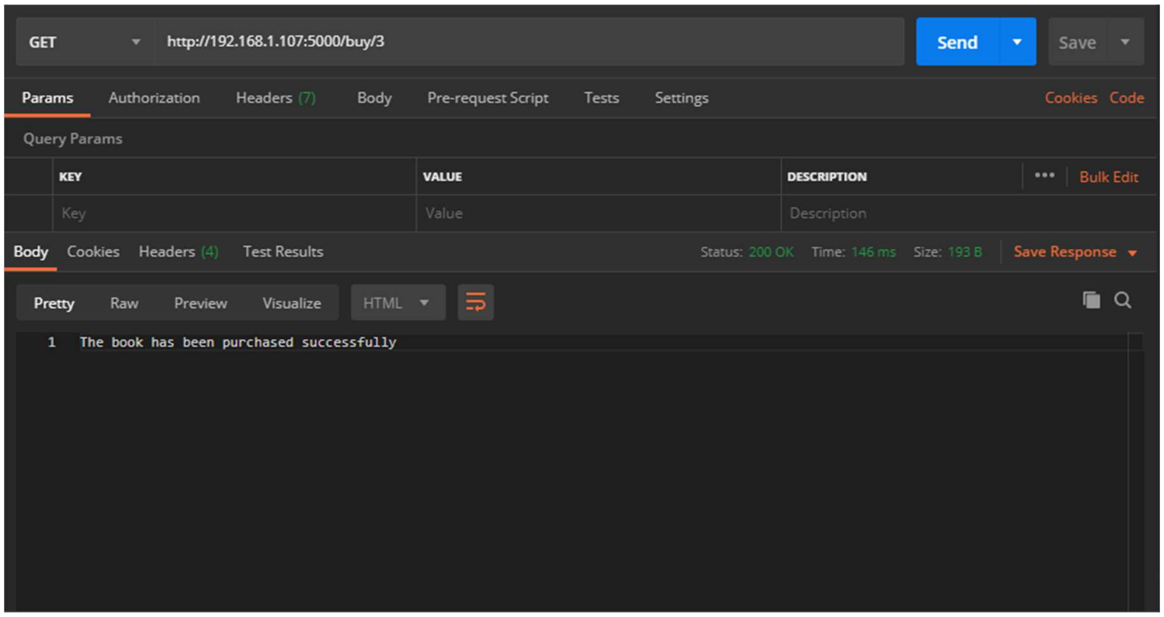
```
{
    "id": 3,
    "price": 30,
    "quantity": 18
}
```

Figure 12: First lookup request output - Consistency check

From the request we can see that there are 18 books in the inventory.

The second request is to buy the book with id 1, which will update the database.

Output:


GET http://192.168.1.107:5000/buy/3

Status: 200 OK Time: 146 ms Size: 193 B

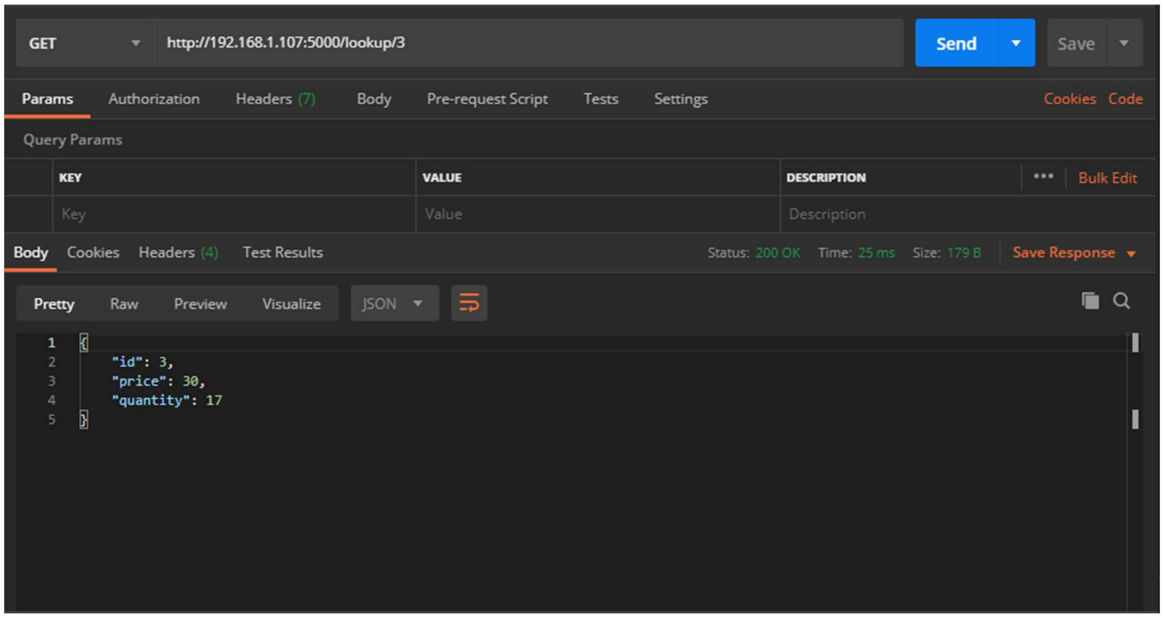
```
The book has been purchased successfully
```

Figure 13 Buy request output - Consistency check

After this request the quantity should be at 17.

Now we are going to send two lookup requests, and remember that cache isn't implemented. And since we know that our system implements load balancing then each catalog server will handle a request.

The third request output:

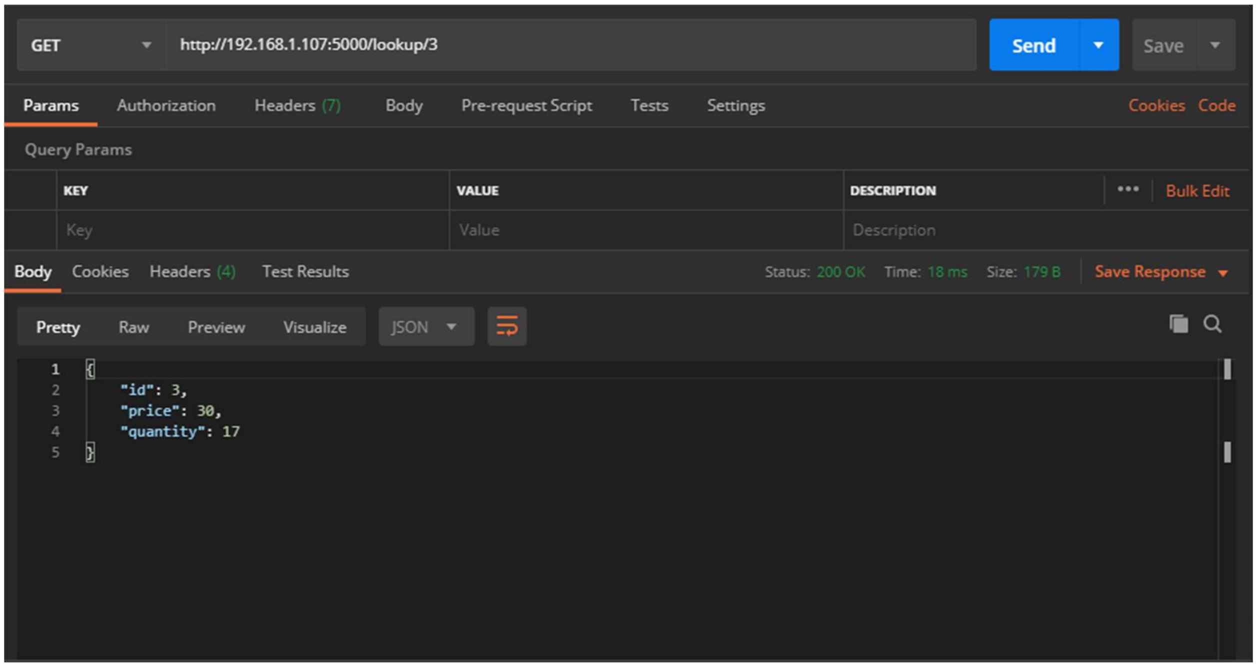

GET http://192.168.1.107:5000/lookup/3

Status: 200 OK Time: 25 ms Size: 179 B

```
{
    "id": 3,
    "price": 30,
    "quantity": 17
}
```

Figure 14 Second lookup request output - Consistency check

The fourth request output:



http://192.168.1.107:5000/lookup/3

Status: 200 OK Time: 18 ms Size: 179 B

```
{
    "id": 3,
    "price": 30,
    "quantity": 17
}
```

Figure 15 Third lookup request output - Consistency check

From the third and fourth requests output, we can see that they both agree that the quantity is 17, which means that the database has accomplished consistency

All the images in this document can be found in the repository.