**Distributed Operating Systems**

**Turning the Bazar into an Amazon: Replication, Caching and Consistency Report**

Student: **Aseel BniFadel**

**Dania Tubaileh**

We have started building the design **locally.** The codes were completely written using Java on eclipse IDE. The design depends on Spark dependencies for apache maven project**.** We use a CSV file to be considered as thedatabase in our project which is shared between the catalog and order server.

The process follows one sequence of events where a user starts by writing a request as HTTP request (*e.g.: http:// [IPAddress]:[Port]/ServiceName/parameter*) , the HTTP request will be caught by the front-end server which will filter the requests and route them to the right back-end server. In the back-end servers, each server can access the database and do the request as it was programmed for it to do (more explaining will be below).

The front-end server is the last server that we run once we want to run the whole system, because we want to make sure that the ports numbers it has are the right ones so we assign the backend servers ports, run them then run the front end.

The code of the front-end server contains a Round-Robin algorithm for load balancing and in-memory-cache integrated.

The back-end server (catalog) which is responsible for lookup and search requests has a class for handling the management of the database , where it only reads as requested from the database.

The back-end server (order) is responsible for one request which is the buy request which is supposed to modify the database by decrementing the specific book amount by one.

Our program expected scenario:

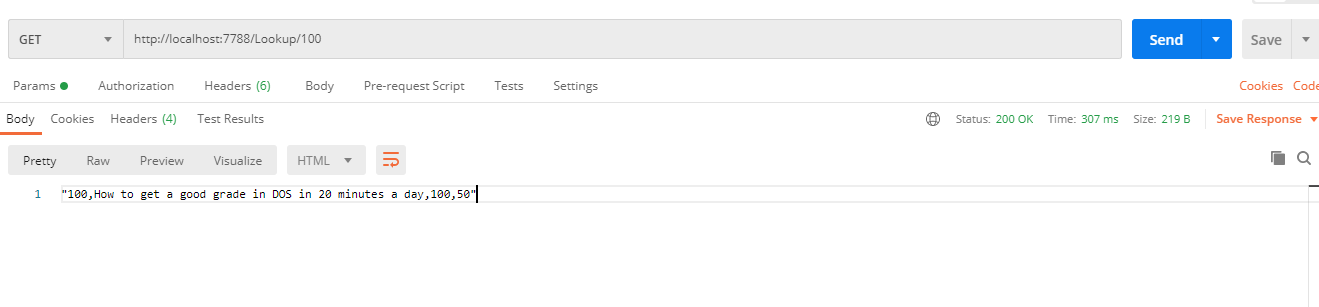
1- We run the backend servers first after giving them 2 available ports through the arguments variable.

2-Then, we run the FrontEnd Server on an available port. (7788) in our code;

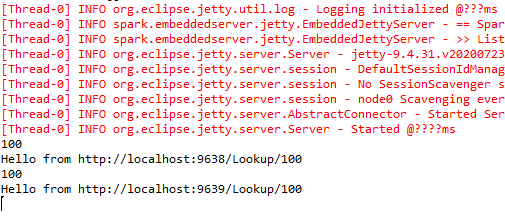
3- We put the HTTP request URL on Postman, then it will display the result as in the screenshots below.

**Performance results of your measurements/experiments:**

Load Balancing using Round Robin Algorithm:



We request the Lookup service two times from Frontend server IP in sequence. In each time, we printed on the eclipse console the IP of the server that handle the request as it’s shown below:



This line in the code which prints the ip:

String url = roundRobinLoadBalancer.getIp()+"Lookup/"+ id;

**In-Memory Cache:**

We implemented the cache in a simple way, which is an ArrayList as a member field in the Frontend server; it has a fixed small size.

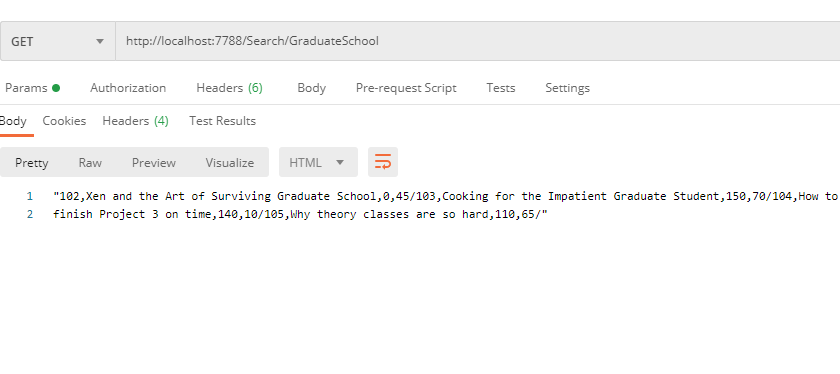
When a Lookup Request happen or Search request, the resultant books will be added to the list, when it becomes full, the least used book will be removed from the list.

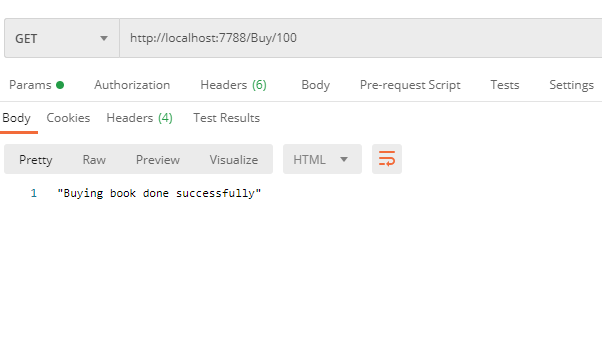
The following table will illustrate the response time of the requests.

|  |  |  |  |
| --- | --- | --- | --- |
| Request type | Without Caching | With Caching ( request first time) | With Caching ( after a while) |
| Look up | 251ms | 211 ms | 7 ms |

**Some Images of the output:**

Search Service:

****Buy Service for available book(amount is not 0):



Buy Service for an out of stock book (amount is 0): 