

# DOS Project Part 2

ابراهيم عارضة 11819862  
احمد نبهان 11715158

**This part depends on the last part .**

In this part we added some features like caching and replication to improve request processing latency .

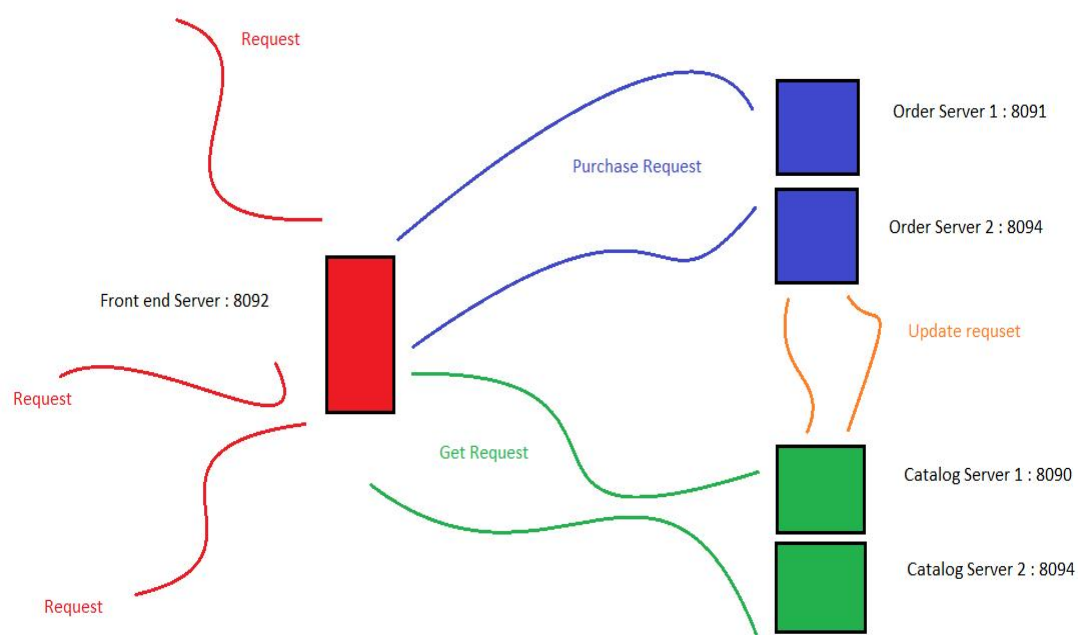
In part one our front end server was simple , it was for passing request without anymore features , but with this part it became more complex .

The Two main functions added to project :

1-caching .

2-Replication .

The figure below describe our project :



**Front end Server** at **port 8092** to handle requests and pass them to other servers depends on some operations .

**Order Server 1** at **port 8091** to listen to purchase requests from **front end server** .

**Other Order server** at **port 8094** to listen to purchase requests from **front end server** .

**Catalog Server 1** : at **port 8090** to handle request from Order and front servers .

**Catalog Server 2** : at **port 8094** to handle request from Order and front servers .

**\* we Replicate Order and catalog server to achieve replication feature .**

## Some Results :

Before caching the Response time for request ( findById ) was

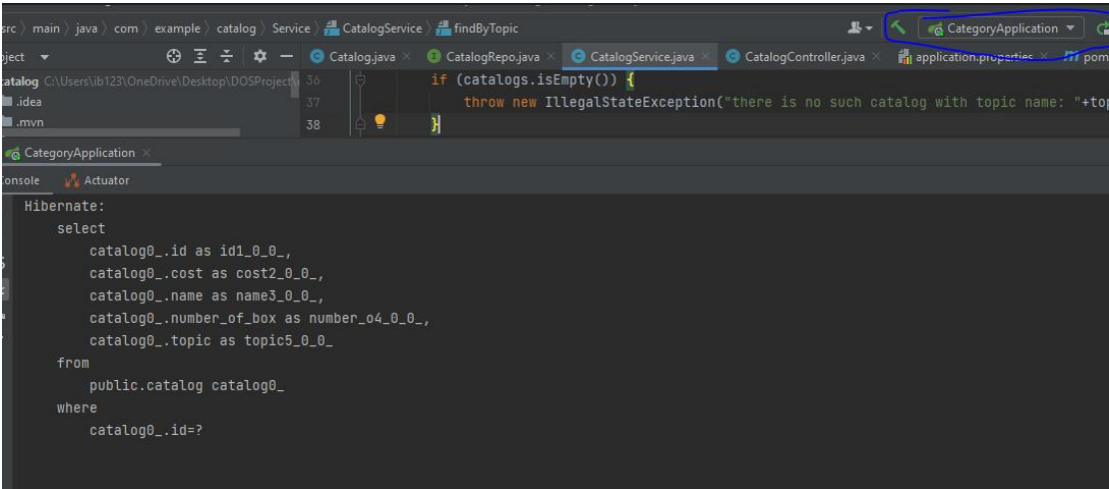
**3.08 s**

The screenshot shows a REST client interface with a GET request to `http://localhost:8092/api/v2/catalog/findById/1`. The response status is 200 OK, and the response time is 3.08 s, which is circled in blue. The response body is displayed in JSON format, showing an object with fields: `id`, `name`, `topic`, `numberOfBox`, and `cost`.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

```
1 {
2   "id": 1,
3   "name": "How to get a good grade in DOS in 40 minutes a day",
4   "topic": "distributedSystem",
5   "numberOfBox": 0,
6   "cost": 10.0
7 }
```

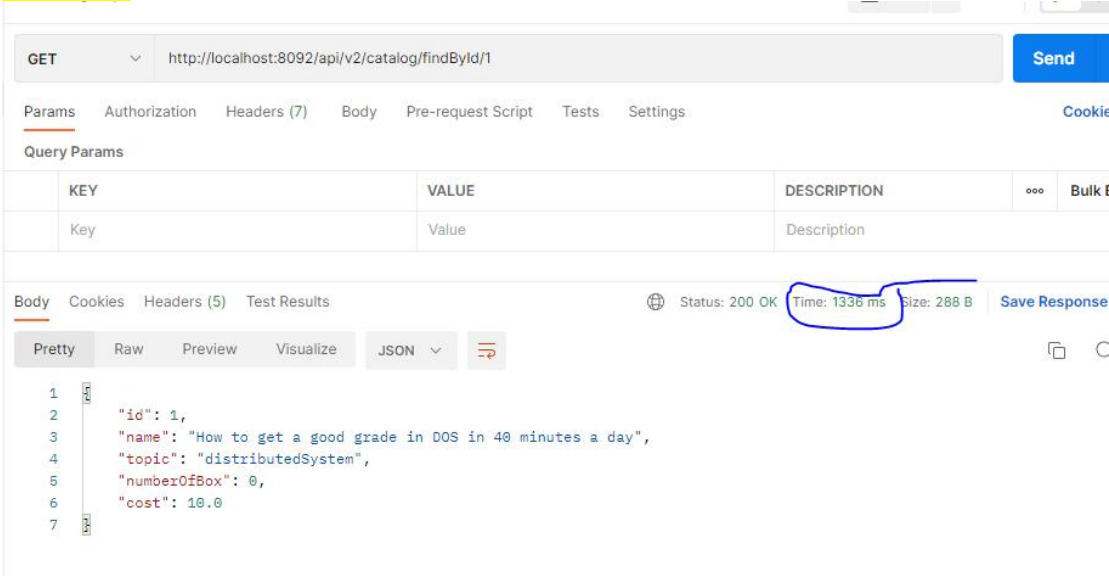
Handling first request in Catalog server .



The screenshot shows an IDE with several tabs open: Catalog.java, CatalogRepo.java, CatalogService.java, CatalogController.java, application.properties, and pom.xml. The CatalogService.java tab is active, showing a method findByTopic. The console window at the bottom displays a Hibernate SQL query: 

```
select catalog0_.id as id1_0_0_, catalog0_.cost as cost2_0_0_, catalog0_.name as name3_0_0_, catalog0_.number_of_box as number_04_0_0_, catalog0_.topic as topic5_0_0_ from public.catalog catalog0_ where catalog0_.id=?
```

After caching the Response time for request( findById) was 1336ms



The screenshot shows a REST client interface. The top bar shows a GET request to `http://localhost:8092/api/v2/catalog/findById/1`. The 'Send' button is visible. Below the bar, the 'Query Params' section is empty. The 'Body' section shows the response in JSON format: 

```
{  "id": 1,  "name": "How to get a good grade in DOS in 40 minutes a day",  "topic": "distributedSystem",  "numberOfBox": 0,  "cost": 10.0}
```

 The status bar at the bottom indicates 'Status: 200 OK', 'Time: 1336 ms', and 'Size: 288 B'. The 'Time: 1336 ms' is circled in blue.

Before caching	After caching
3.08 s	1336 ms

## Why ?

The first request of ( findById/1) reached to front end server to get information about the book with ID =1 , at this point there was no information about this book in cache memory in front end server , so front end server pass this request to Catalog server to get information . So this request will take long time ( 3.08 S).

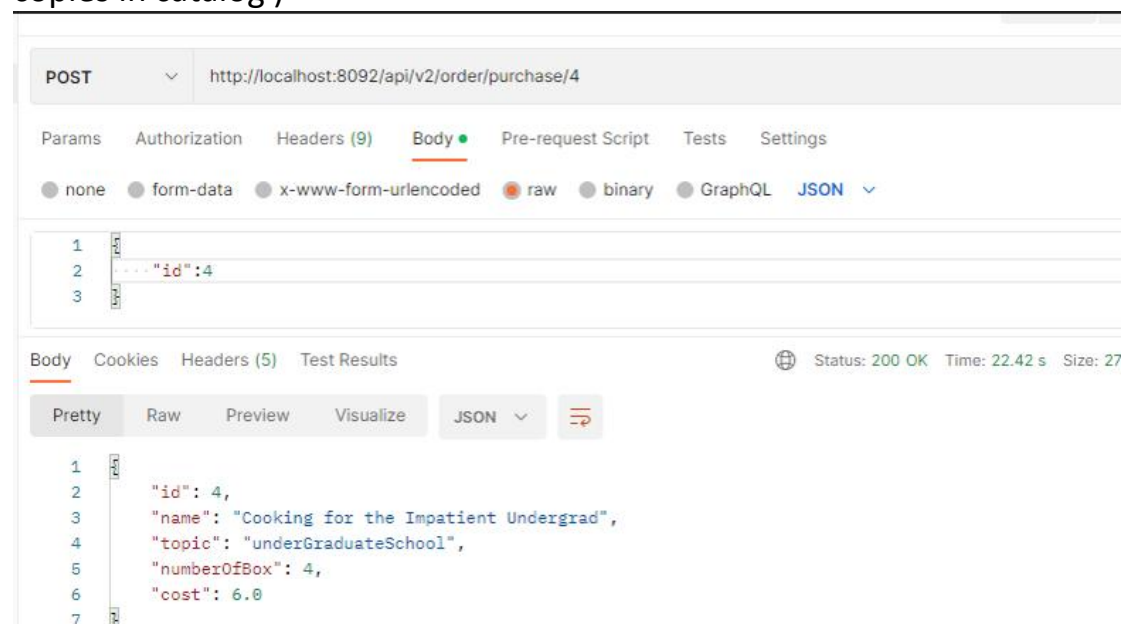
After this request information about this book will be stored in cache memory .

The second time of the same request (findById/1) will be handled in Front end server , So this will take less time rather than First time. (1336 ms)

## What happens if data stored in Catalog servers is updated ?

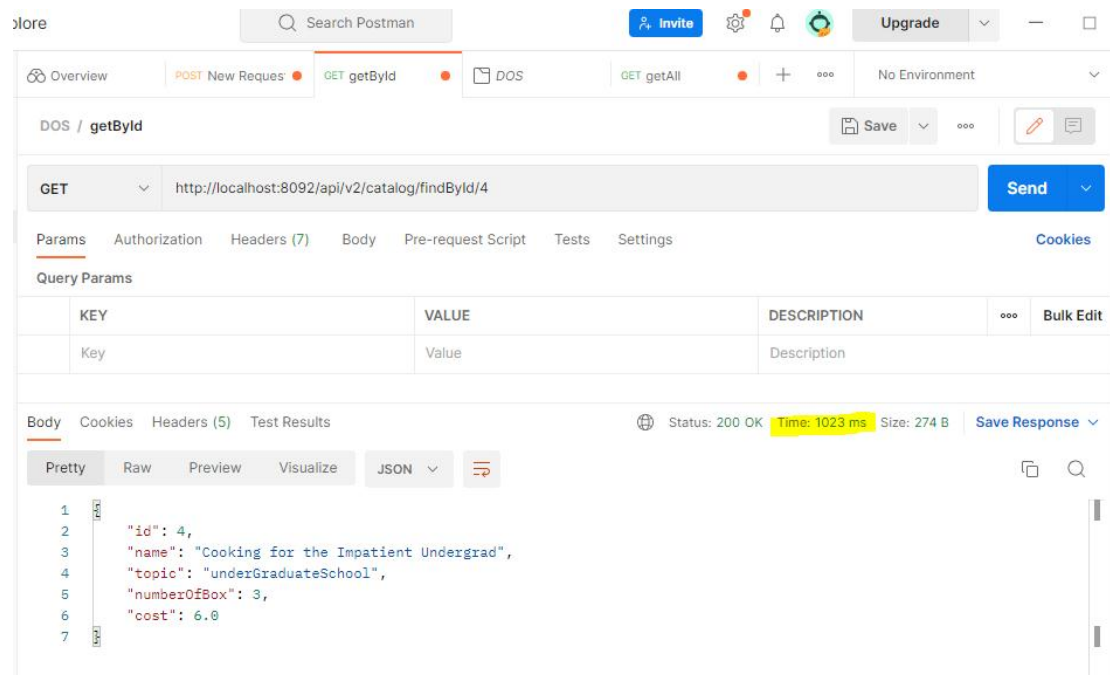
This update will be reflect to cache memory in front-end server . this request is to purchase book with Id 4 (Buy one copy)

So number of copies of this book will decremented by 1 (it will be now 3 copies in catalog )



If I make get request (findByld/4) for the same book , it will be handled in front end server and will return response with information stored in cache memory .

Notice that the number of copies of this book is 3 which is the same as information stored in catalog server .



This evidence that any changes happens in Catalog reflect to cache memory .....

## How we made caching with Spring Boot environment:

By using annotation `@EnableCaching`

`@EnableCaching` annotation is the annotation-driven cache management feature in the spring framework.

expression used for making the method caching conditional.

**@Cacheable(cacheNames = "Books", key = "#id")**

With the **@CachePut** annotation, we can update the content of the cache

**@CachePut(cacheNames = "Books", key = "#book.id")**

This is all about Caching .....

## Replication and Load Balancing

To deal with this replication, the front end node needs to implement a load balancing algorithm that takes each incoming request and sends it to one of the replicas. We used Round Robin algorithm to achieve load balancing .

**Round - robin load balancing** is one of the simplest methods for distributing client requests across a group of servers. Going down the list of servers in the group, the round- robin load balancer forwards a client request to each server in turn. When it reaches the end of the list, the load balancer loops back and goes down the list again .

## Results :

The first request handled by first order server at port 8091 .

```
[Server:localhost:8091; Zone:UNKNOWN; Total Requests:0; Successive connection failure:0; Total blackout seconds:0; Last connection made:Thu
```

The second request handled by other order server at port 8094 .

```
2022-12-12 18:55:52.240 INFO 7124 --- [nio-8094-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-12-12 18:55:52.702 INFO 7124 --- [nio-8094-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 457 ms
```

## How we made load Balancing with Spring Boot environment:

Used for configuring your Ribbon client(s).

```
@RibbonClients({ @RibbonClient(name = "catalog-service"),
    @RibbonClient(name = "order-service")})
```

Used as a marker annotation indicating that the annotated RestTemplate should use a RibbonLoadBalancerClient for interacting with your service(s).

```
@LoadBalanced
```

**The End of Report.....**







