ابراهيم عارضة 11819862
احمد نبهان 11715158

DOS Project Part 2

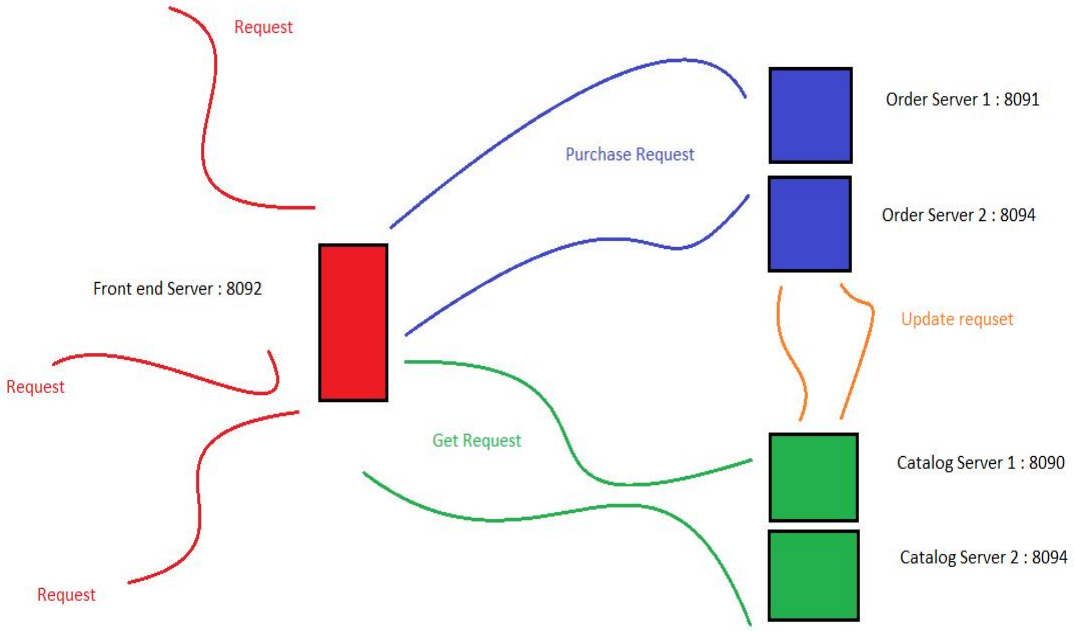
This part depends on the last part .

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** .

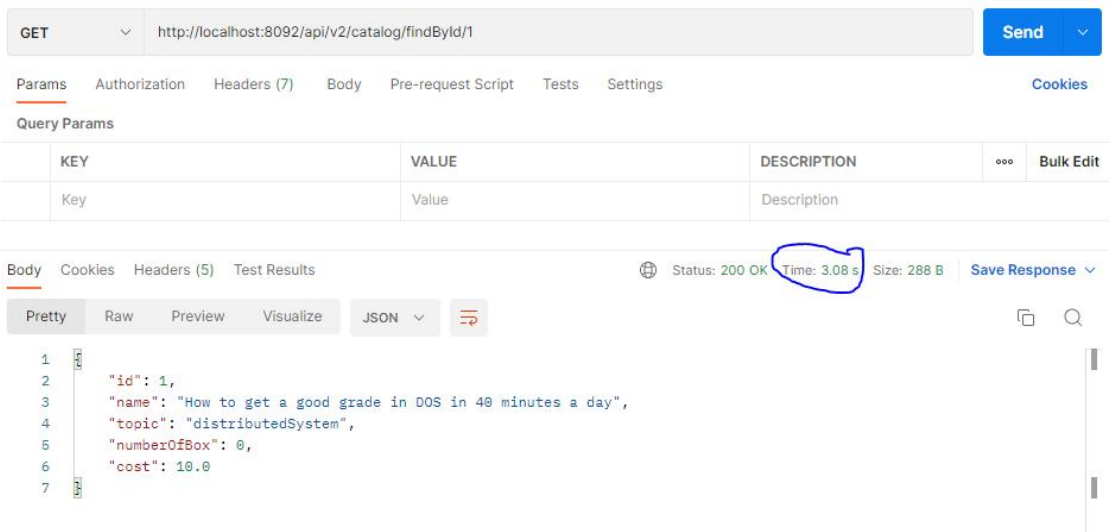
Catalog Server 1 : at **port 8090** to handle request from Order and front servers .

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 he did not find data in memory cache (miss cache) .

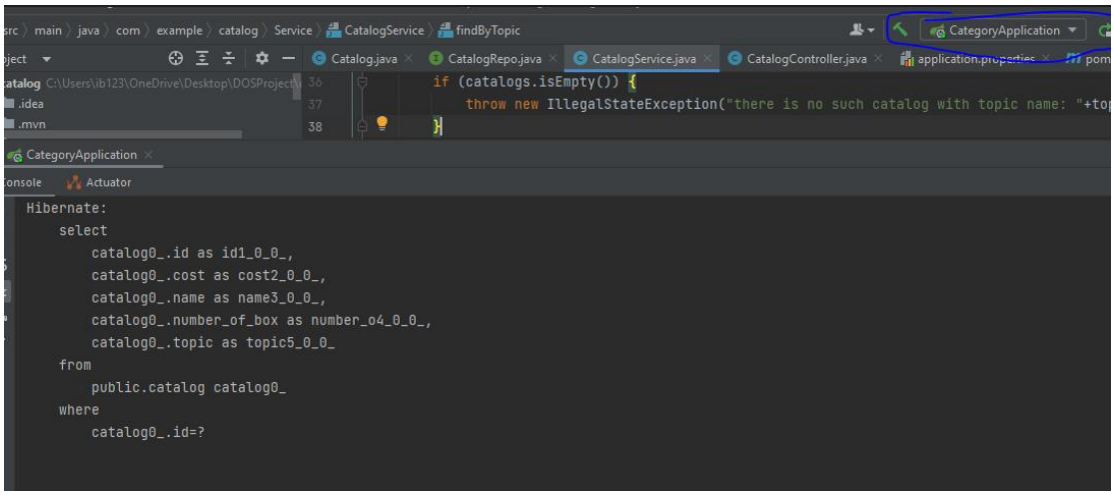


So ,Handling first request in Catalog server . so this spent long time .

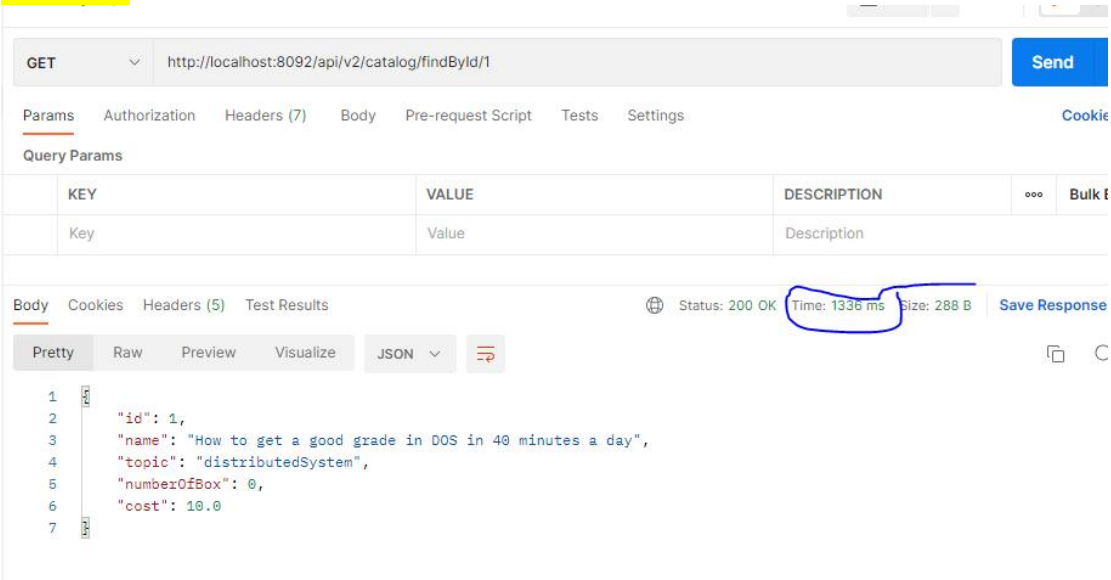
What is the latency of a subsequent request that sees a cache miss?
3.08 seconds

What are the overhead of cache consistency operations?

If miss cache happen in many times this will put the system in overhead state , it will spend more time to get information and save them into memory cache …



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



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)

POST http://localhost:8092/api/v2/order/purchase/4

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "id":4
}
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 22.42 s Size: 27

Pretty Raw Preview Visualize JSON

```
{
    "id": 4,
    "name": "Cooking for the Impatient Undergrad",
    "topic": "underGraduateSchool",
    "numberOfBox": 4,
    "cost": 6.0
}
```

If I make get request (findById/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 .

Search Postman Invite Upgrade

Overview POST New Reques GET getById DOS GET getAll No Environment

DOS / getById Save

GET http://localhost:8092/api/v2/catalog/findById/4 Send

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

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

Body Cookies Headers (5) Test Results Status: 200 OK Time: 1023 ms Size: 274 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "id": 4,
    "name": "Cooking for the Impatient Undergrad",
    "topic": "underGraduateSchool",
    "numberOfBox": 3,
    "cost": 6.0
}
```

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……