



Computer Engineering Department
Course Name: Distributed Operating System

Instructor: : Dr. Samer Arendi	Lab2:Bazar.Com
Academic Year: 2022-2023	
Semester:1 st	

Students	
1-Randa Fadi Alawneh (11819413)	2-Amal Zetawi
Performed on:	Submitted on:



Objectives:

The purpose of this lab is to teach you Replication, Caching, Consistency and to teach concepts of multi-tier web design and micro-services.

Introduction:

In this part, we will add replication and caching to improve request processing latency. While the front-end server in lab 1 was a very simple component, in this part, we will add two types of functionality to the front-end node. First, we will add an in-memory cache that caches the results of recent requests to the catalog servers. Second, assume that both the order and catalog server are replicated - their code and their database files are replicated on multiple machines.

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 as load balancing algorithm.

We build Cache in-memory cache that integrated into the front-end server process, in which case, internal function calls are used to get and put items into the cache.

To ensure strong consistency guarantees, we implement a serverpush techniques where backend replicas send invalidate requests to the in-memory cache prior to making any writes to their database files. The invalidate request causes the data for that item to be removed from the cache. also we add other caching features such as a limit on the number of items in the cache, So we need a cache replacement policy ,we used LRU to replace older items with newer ones.

The replicas also use an internal protocol to ensure that any writes to their database are also performed at the other replica to keep them in sync with one another.

Procedure:

Here is an explanation of some of the steps of building the project:

- We need 3 computer ,So we use ubuntu as a front end server ,Windows Computer as as Catalog server ,another Windows Computer as Order Server , Every 3 computer connected on the same network.
- We use Node Framework, because Node provide Speed, Scalability, Productively. Also we use Visual Studio code as editor , In order to create and test the API we use POSTMAN.



- Order and Catalog Server need a database , So we use SQLITE which is a very simple database.
- We use SQLITE Studio to browse SQLITE database.

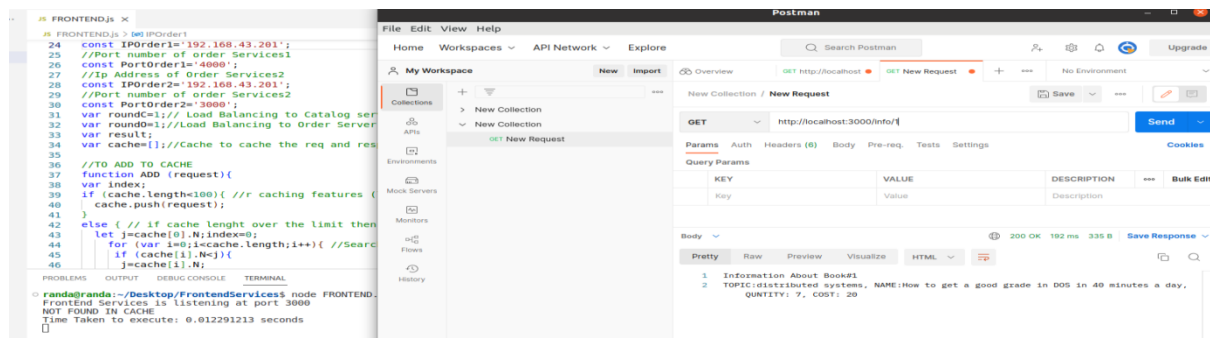
HOW to Run the BAZAR.COM:

- Each Folder of Different Services, Put it in different computer, and open this folder by using visual code studio.
- We need to install Some libraries to each Server Such : Express.js, SuperAgent, Sqlite, body parser . For example we need Express.js to create sever running at specific port and to build route, also We need SuperAgent to make request.
- In order to install library we need to write visual code terminal in each services :
npm install Eexpress, npm install superagent, npm install sqlite, npm install bodyparser.
- In order to start each services , we need to write at each services terminal node filename.
- open postman in front end services and start to create API and test the result.

Result :

We test result by send different request to frontend server:

When send REQUEST first time to ask about information about specific book,(of course the information about book does not found in cache) so they send REQUEST to one catalog server from 2 replica:





When send REQUEST another time to ask about information about specific book, the req should save in cache so we don't go to catalog server to find the response:

The screenshot displays a Node.js application running in a terminal window. The code defines a caching system for book information. The first request is served from the cache, as indicated by the terminal output: "Data Found In Cache: Information About Book#1". The Postman interface shows a GET request to `http://localhost:3000/info/1` with a response containing book details: "Information About Book#1", "TOPIC: distributed systems, NAME: How to get a good grade in DOS in 40 minutes a day, QUANTITY: 7, COST: 20".

Send another REQUEST to different book information to see the load balancing between 2 catalog server:

The screenshot displays a Node.js application running in a terminal window. The code defines a caching system for book information. The second request is served from the cache, as indicated by the terminal output: "Data Found In Cache: Information About Book#2". The Postman interface shows a GET request to `http://localhost:3000/info/2` with a response containing book details: "Information About Book#2", "TOPIC: distributed systems, NAME: RPCs for Noobs, QUANTITY: 56, COST: 30".



Load Balancing in catalog server :

```
17 console.log("Catalog Server is listening at port 5000");
18 });
19 // Connecting SQLite Database
20 const sqlite3 = require('sqlite3');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\AB\OneDrive\حظي\بتكمل\Catalog_Server1> node CatalogServer.js
Catalog Server is listening at port 5000
Catalog DataBase Connected
Information About Book#1
TOPIC:distributed systems, NAME:How to get a good grade in DOS in 40 minutes a day, QUNTY: 7, COST: 20

```
17 console.log("Error Occurred - " + err.message);
18 }
19 else {
20 console.log("Catalog DataBase Connected");
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\AB\OneDrive\حظي\بتكمل\Catalog_Server2> node CatalogServer.js
Catalog Server is listening at port 6000
Catalog DataBase Connected
Information About Book#2
TOPIC:distributed systems, NAME:RPCs for Noobs, QUNTY: 56, COST: 30

Send order REQUEST, when send a request to order server , then order server check no of items then if $no > 0$, we add the order to 2 server to achieve consistency between two replica , and we edit the amount value in two catalog server , and finally we send invalid REQ to front to remove the item information from cache:

```
24 const IPOrder1='192.168.43.201';
25 //Port number of order Services1
26 const PortOrder1='4000';
27 //Ip Address of Order Services2
28 const IPOrder2='192.168.43.201';
29 //Port number of order Services2
30 const PortOrder2='3000';
31 var roundC=1; // Load Balancing to Catalog server
32 var roundO=1; // Load Balancing to Order Server
33 var result;
34 var cache=[]; //Cache to cache the req and res
35
36 //TO ADD TO CACHE
37 function ADD(request){
38 var index;
39 if (cache.length<100){ //if caching features (
40 cache.push(request);
41 }
42 else { // if cache length over the limit then
43 let j=cache[0].N;index=0;
44 for (var i=0;i<cache.length;i++){ //Search
45 if (cache[i].N==j){
46 j=cache[i].N;
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\AB\OneDrive\حظي\بتكمل\FrontendServices> node FRONTEND.js
FrontEnd Services is listening at port 3000
Time Taken to execute: 0.012291213 seconds
Data Found In Cache:
Information About Book#1
TOPIC:distributed systems, NAME:How to get a good grade in DOS in 40 minutes a day, QUNTY: 7, COST: 20
NOT FOUND IN CACHE
Time Taken to execute: 0.000510356 seconds
Time Taken to execute: 0.000765283 seconds
Time Taken to execute: 0.001164085 seconds
0
[{ URI: 'info/2',
 Response: 'Information About Book#2\nTOPIC:distributed systems, NAME:RPCs for Noobs, QUNTY: 56, COST: 30',
 No: 0 }]
Overhead of cache consistency: 0.004187872 seconds

Home Workspaces API Network Explore

My Workspace

Collections

New Collection

New Request

POST http://localhost:3000/purchase/1

Params Auth Headers (7) Body Pre-req Tests Settings

Query Params

KEY VALUE DESCRIPTION Bulk Edit

Key Value Description

Body

Pretty Raw Preview Visualize HTML

200 OK 185 ms 244 B Save Response

1 The book has been successfully ordered.

Get started on Postman

Next: Save a request.

33%

Show me

Online Find and Replace Console

Cookies Capture requests Bootcamp Runner Trash



CatalogServer1:

```
17 | | console.log("Error Occurred - " + err.message);  
18 | }  
19 | else {  
20 | console.log("Catalog DataBase Connected");
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\AB\OneDrive\حظي\بتعلم\Catalog_Server2> node CatalogServer.js
Catalog Server is listening at port 6000
Catalog DataBase Connected
Information About Book#2
TOPIC:distributed systems, NAME:RPCs for Noobs, QUANTITY: 56, COST: 30
Update Done

CatalogServer2:

```
15 | let db = new sqlite3.Database("../Catalog2.db", (err) => {  
16 | | if(err) {  
17 | | | console.log("Error Occurred - " + err.message);  
18 | | }  
19 | | else {  
20 | | | console.log("Catalog DataBase Connected");
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\AB\OneDrive\حظي\بتعلم\Catalog_Server2> node CatalogServer.js
Catalog Server is listening at port 6000
Catalog DataBase Connected
Information About Book#2
TOPIC:distributed systems, NAME:RPCs for Noobs, QUANTITY: 56, COST: 30
Update Done



DB in 2 catalog server:

ID	Topic	Name	Amount	Cost
1	1 distributed systems	How to get a good grade in DOS in 40 minutes a day	6	20
2	2 distributed systems	RPCs for Noobs	56	30
3	3 undergraduate school	Xen and the Art of Surviving Undergraduate School	63	50
4	4 undergraduate school	Cooking for the Impatient Undergrad	56	100

OrderServer1:

```

File Edit Selection View Go Run Terminal Help
Order_Server1

EXPLORER
ORDER_SERVER1
  node_modules
  JS CREATE_DB.js
  JS Order_Server.js
  Order1.db
  package-lock.json
  package.json

JS Order_Server.js
13 Order_Server.js > IPFront
14 const PortCatalog1= 5000 ;
20 //IP Address of Catalog Service2
21 const IPCatalog2='192.168.43.112';
22 //Port number of Catalog Service2
23 const PortCatalog2='6000';
24 //Ip Address of Order Services2
25 const IPOrder2='192.168.43.201';
26 //Port number of order Services2
27 const PortOrder2='3000';
28 //IP Address of FrontEnd Server

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\s1181\Desktop\OrderServer\Order_Server1> node Order_Server.js
Order Server is listening at port 4000
DataBase Connected
The Book Available In Stock
  
```




OrderServer2:

The screenshot shows the Visual Studio Code editor with the file `Order_Server.js` open. The code in the file is as follows:

```

1 //The Express package is a structure for Node applications used for supporting the Node.js +
2 //Using Express to create server running on specific port + Express JS makes it super simple
3 const express = require("express");
4 const app = express(); //Create instance of Express to use it
5 const superagent = require('superagent');

```

The terminal output shows the following commands and results:

```

PS C:\Users\s1181\Desktop\OrderServer\Order_Server2> node Order_Server.js
Order2 Server is listening at port 3000
DataBase Connected
ADD DONE

```

DB in orderServer1:

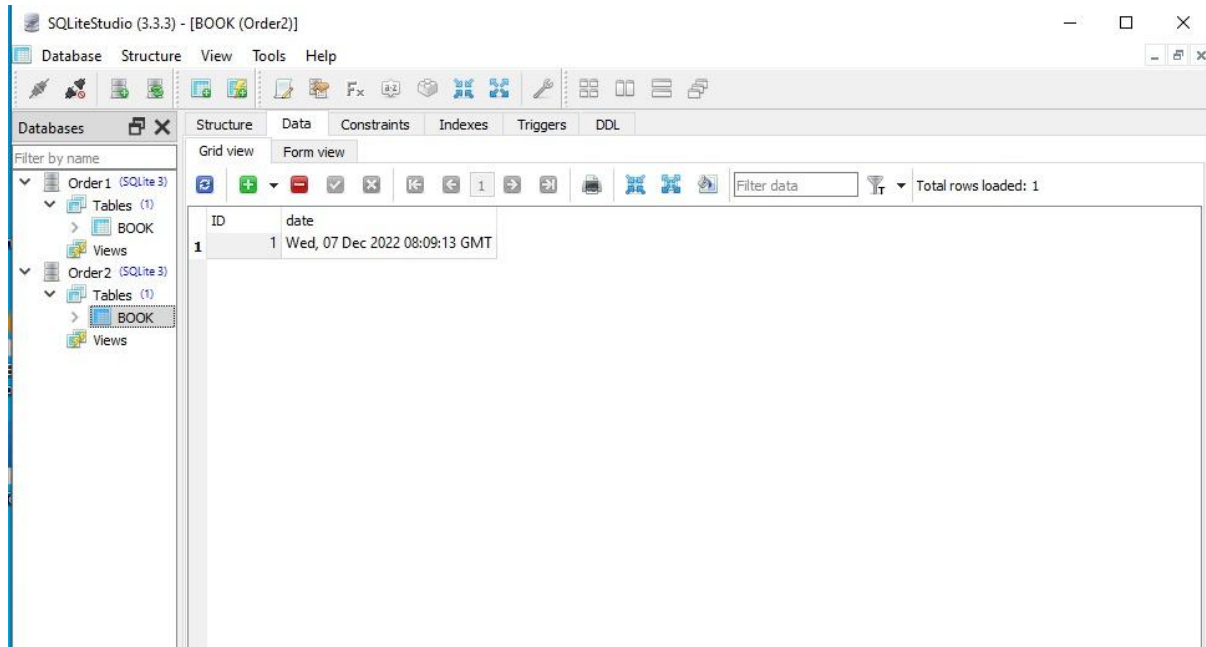
The screenshot shows the SQLiteStudio interface with the `BOOK` table selected. The table structure is as follows:

ID	date
1	Wed, 07 Dec 2022 08:09:13 GMT

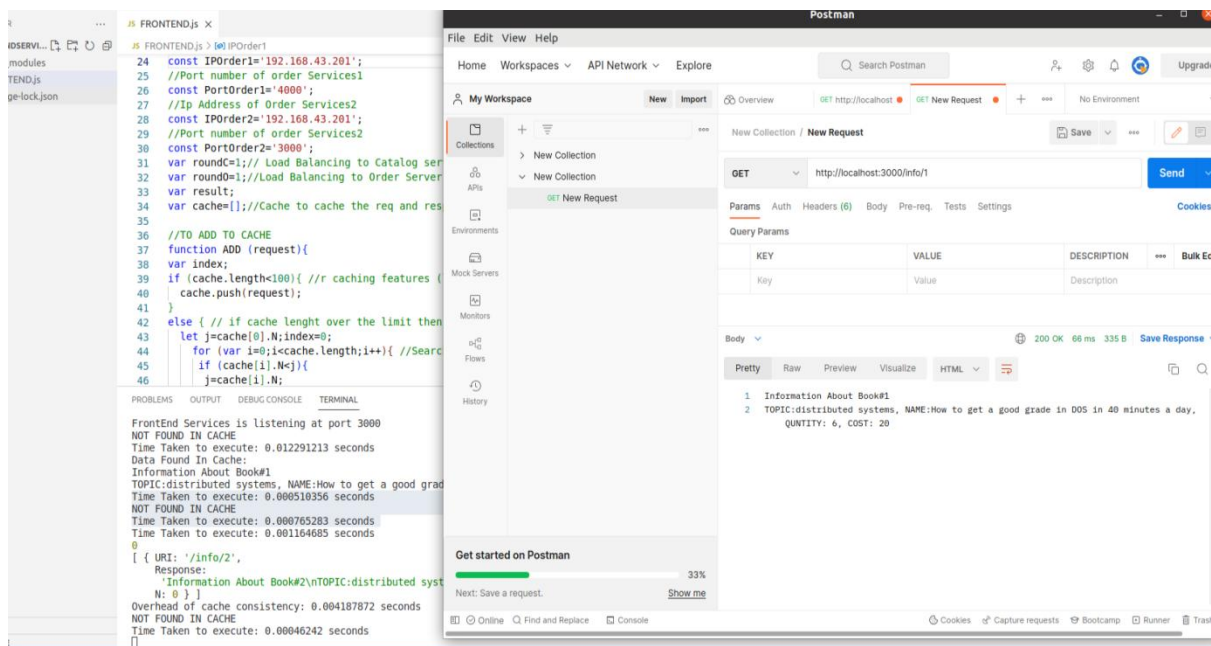
The table is named `BOOK` and is located in the `Order1` database. The data shows a single row with ID 1 and a date of Wed, 07 Dec 2022 08:09:13 GMT.



DB in orderServer2:



After purchase send info to Frontend to see that we don't find the value in cache and go to catalog server.





Experimental Evaluation and Measurements

Performance measurements or experiments to show/evaluate specific functionality:

- 1- Compute the average response time (query/buy) of your new systems. What is the response time with and without caching? How much does caching help?

Request Type	Response time with cache	Response time without cache
Info/1	0.000486421 seconds	0.000515828 seconds
Search/distributed systems	0.000360265 seconds	0.000486423 seconds
Info/2	0.000465991 seconds	0.000445246 seconds
Search/undergraduate school	0.000323171 seconds	0.00046743 seconds
Info/9	0.000373168 seconds	0.000453741 seconds

- 2- We Construct a simple experiment that issues orders or catalog updates (i.e., database writes) to invalidate the cache and maintain cache consistency. What are the overhead of cache consistency operations?

Request Type	Overhead of cache consistency
Purchase/3	0.000653568 seconds
Purchase/1	0.00827149s
Purchase/2	0.000803418 seconds
Purchase/4	0.000752267 seconds

As we noticed in general, the use of cache reduces the response time to the request. It is true that sometimes the data is not found in the cache and sometimes is wasted time for searching in the cache(cache miss), but this time is not large compared by the time taken if we sending the request directly to the catalog server.

Cache must be handled with caution, as some data becomes invalid in the cache and we need to delete it when a request arrives to the frontend server to delete it and this waste some time by maintaining the consistency of the information in the cache.

However, despite what was previously mentioned, cache has a very important benefit in reducing response time and maintaining performance.



CONCLUSIN:

We Learn from this Project : Replication, Caching, Consistency, and how load balancing between replication.