

DOS Project Report Part 2

Student Name: Muna Khwaireh Stu#: 12028473

Student Name: Samah Qaradeh Stu#: 12028923

Introduction:

This project aims to improve Bazar.com's online bookstore by reducing response times and handling higher user demand. To achieve this, we incorporated **caching** and **replication** to enhance performance and ensure data consistency, especially during frequent data updates. Additionally, we used **Dockerization** to simplify deployment and management of the bookstore's microservices as containerized applications.

By implementing caching for frequently accessed data and distributing requests across replicated servers, we reduced backend load and improved response times. This report outlines the updated system's design, evaluates performance gains, and demonstrates how these enhancements support a more scalable and responsive application.

Part1: Cache Consistency & load balancing:

```
// Route to get book information by item ID
app.get('/info/:id', async (req, res) => {
  const itemId = req.params.id;
  try {

    // Forward the info request to the catalog service
    const response1=await axios.get(`http://catalogcash:3004/info/${itemId}`);
    const msg1=response1.data.message;
    const specificMessage1 = 'Book from small cash'; ←

    if (msg1 === specificMessage1) {
      |   |   res.json(response1.data);
    } else {

      const response2 = await axios.get(`http://catalog:3002/info/${itemId}`); ↑

      if(response2.data.message==="Book from big memory"){
        |   |   const response3 = await axios.post(`http://ordercash:3005/add_new_book`, response2.data);
        |   |   res.json(response2.data);
      } else {

        |   |   res.json({
        |   |     message: "Book not found in big memory and not found in small cash"
      });
    }
  }
});
```

Note: we has 5 operations: search,info, purchase, update cost and update quantity.

- When i send **GET** request the **first time, before Caching the data:**

So the data will come from the memory

GET searchtopic

HTTP New Collection / **searchtopic**

GET http://localhost:3001/search/distributed systems

Params Authorization Headers (6) Body Scripts Tests Settings

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1  "message": "we found 2 books in big memory",
2  "data": [
3      {
4          "id": 1,
5          "title": "How to get a good grade in DOS",
6          "topic": "distributed systems",
7          "price": 711,
8          "quantity": 9
9      },
10     {
11         "id": 2,
12         "title": "RPCs for Noobs",
13         "topic": "distributed systems",
14         "price": 55,
15         "quantity": 68
16     }
17 ]
18
19

```

200 OK 98 ms 483 B Save Response Cookies

GET searchtopic

HTTP New Collection / **searchtopic**

GET http://localhost:3001/info/2

Params Authorization Headers (6) Body Scripts Tests Settings

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2      "id": 2,
3      "title": "RPCs for Noobs",
4      "topic": "distributed systems",
5      "price": 55,
6      "quantity": 68,
7      "message": "Book from big memory"
8  }

```

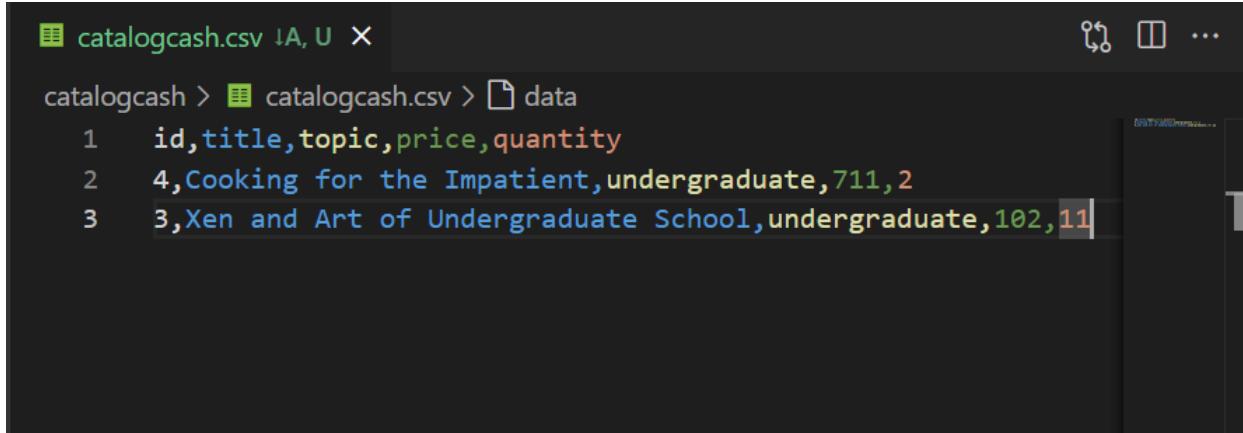
200 OK 297 ms 357 B Save Response Cookies

Postbot

Ctrl Alt P

Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

Our cache capacity is 2 books:



```
catalogcash.csv !A, U X
catalogcash > catalogcash.csv > data
1 id,title,topic,price,quantity
2 4,Cooking for the Impatient,undergraduate,711,2
3 3,Xen and Art of Undergraduate School,undergraduate,102,11
```

Q1) Compute the average response time (query/buy) of your new systems.

What is the response time with and without caching?

- Answers
 - for info: **297ms** for
 - search: **98ms**

after the above requests, the book will be added to the cache , and the below requests shows that the result come from the cache 😊

with cash:

GET searchtopic

New Collection / **searchtopic**

GET http://localhost:3001/info/2

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	...

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 2,
3   "title": "RPCs for Noobs",
4   "topic": "distributed systems",
5   "price": 55,
6   "quantity": 68,
7   "message": "Book from small cash"
8 }

```

Postbot Ctrl Alt P

200 OK 23 ms 357 B Save Response ...

GET searchtopic

New Collection / **searchtopic**

GET http://localhost:3001/search/distributed systems

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "we found 2 books in small cash",
3   "data": [
4     {
5       "id": 2,
6       "title": "RPCs for Noobs",
7       "topic": "distributed systems",
8       "price": 55,
9       "quantity": 68
10      },
11      {
12        "id": 1,
13        "title": "How to get a good grade in DOS",
14        "topic": "distributed systems",
15        "price": 711,
16        "quantity": 9
17      }
18    ]
19 }

```

Postbot Ctrl Alt P

200 OK 37 ms 483 B Save Response ...

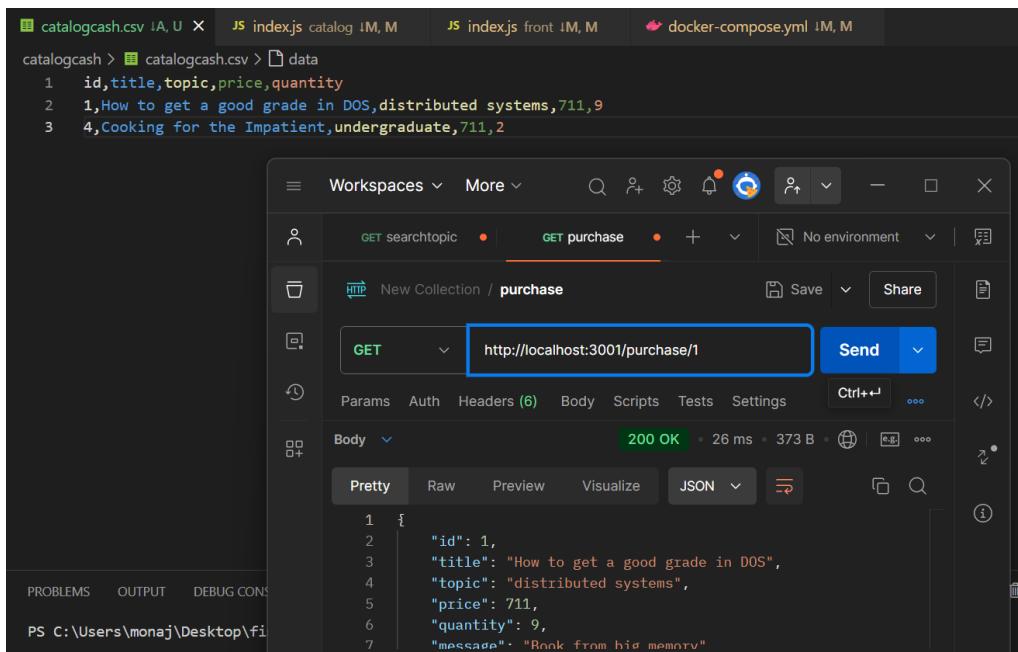
Q2) How much does caching help?

- Answers for info: **23ms**, **297/23 → 12.9 Faster than without cache**
 - search: **37ms**, **98/37→ 2.65 Faster than without using cache**
 -

Invalidate Message

At the first I will purchase a book that is already at the cache:

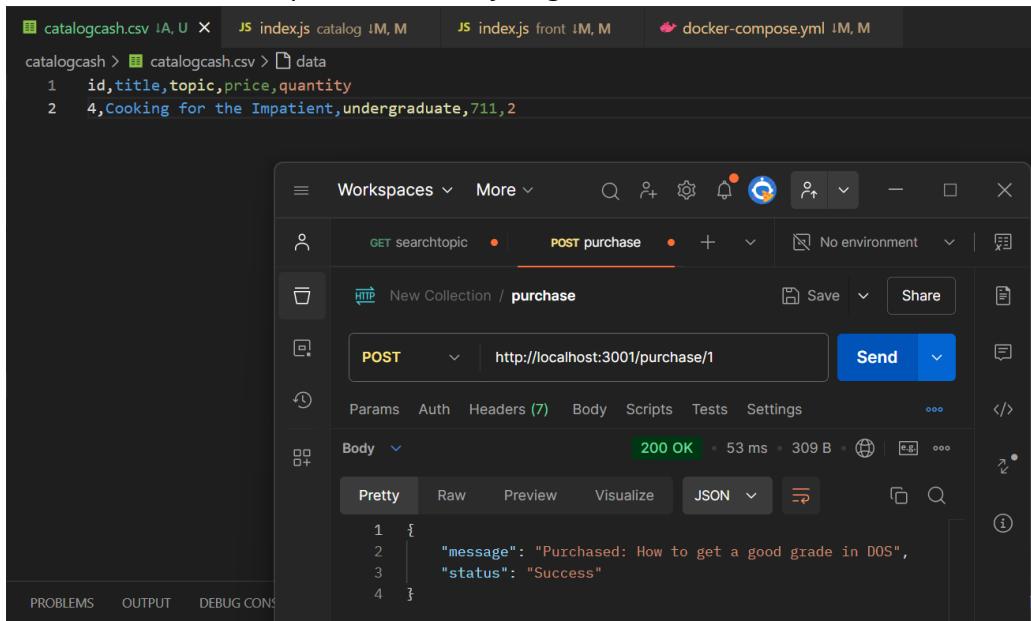
This is before make the request:



The screenshot shows the Postman interface with a collection named "catalogcash". A GET request is made to `http://localhost:3001/purchase/1`. The response is a 200 OK status with a JSON body containing a book's details:

```
1 {  
2   "id": 1,  
3   "title": "How to get a good grade in DOS",  
4   "topic": "distributed systems",  
5   "price": 711,  
6   "quantity": 9,  
7   "message": "Book from his memory"  
}
```

And here after the request, the book just get out of the cache:



The screenshot shows the Postman interface with a collection named "catalogcash". A POST request is made to `http://localhost:3001/purchase/1`. The response is a 200 OK status with a JSON body indicating the purchase was successful:

```
1 {  
2   "message": "Purchased: How to get a good grade in DOS",  
3   "status": "Success"  
4 }
```

And if I search at cache for it:

catalogcash.csv IA, U X JS index.js catalog !M, M JS index.js front !M, M docker-compose.yml !M, M

catalogcash > catalogcash.csv > data

```
1 id,title,topic,price,quantity
2 4,Cooking for the Impatient,undergraduate,711,2
```

The screenshot shows the Postman application interface. At the top, there are tabs for 'catalogcash.csv' (IA, U X), 'index.js catalog' (!M, M), 'index.js front' (!M, M), and 'docker-compose.yml' (!M, M). Below the tabs, it says 'catalogcash > catalogcash.csv > data'. Underneath, there is a code snippet with two rows: '1 id,title,topic,price,quantity' and '2 4,Cooking for the Impatient,undergraduate,711,2'. The main area of the Postman window shows a collection named 'purchase'. A GET request is selected with the URL 'http://localhost:3004/info/1'. The response status is '200 OK' with a time of '13 ms' and a size of '277 B'. The response body is JSON, displayed as follows:

```
1 {
2   "message": "Book not found in small cash"
3 }
```

Expiration Run	Without Cache	With Cache	#of Times when using Cache speed
1	46	9	$46/9 = 5.11$ Times
2	26	7	$26/7 = 3.71$ Times
3	27	9	$27/7 = 3.85$ Times

For Replication Services & Database:

The screenshot displays a file comparison interface comparing two versions of a project structure under the root directory **BAZAR.COM**.

Left Tree (Initial State):

- catalog**: Contains `node_modules`, `catalog.csv` (marked `↓M, M`), `dockerfile` (marked `↓R`), `index.js` (marked `↓M, M`), `package-lock.json` (marked `↓R`), and `package.json` (marked `↓R`).
- catalogcash**: Contains `node_modules`, `catalogcash.csv` (marked `↓A, U`), `dockerfile` (marked `U`), `index.js` (marked `↓A, U`), `package-lock.json` (marked `U`), and `package.json` (marked `U`).
- front**: Contains `node_modules`.
- order**: Contains `node_modules`.
- ordercash**: Contains `bazar.com.code-workspace` (marked `U`), `docker-compose.yml` (marked `↓M, M`), `package-lock.json`, and `package.json`.

Right Tree (Merged State):

- catalog**: Contains `node_modules`, `catalog.csv` (marked `↓M, M`), `front`, `node_modules`, and `order`.
- catalogcash**: Contains `node_modules`, `front`, `node_modules`, and `order`.
- order**: Contains `node_modules`.
- ordercash**: Contains `node_modules`, `dockerfile` (marked `↓A, U`), `index.js` (marked `↓A, U`), `ordercash.csv` (marked `↓A, U`), `package-lock.json` (marked `U`), `package.json` (marked `↓A, U`), `bazar.com.code-workspace` (marked `U`), `docker-compose.yml` (marked `↓M, M`), `package-lock.json`, and `package.json`.

Part2: Dockerize your Application (Optional part)

I Construct my project Dockerized from scratch, i create docker container (image) for each service, and each service has it's own port, and i use volume for sharing data between the Guest OS (Docker Containers) & Host OS , and i create docker-compose file to run all containers at the same time with 1 simple command .

My compose.yml file:

```
version: '3.8'

services:
  frontend:
    build: ./front
    ports:
      - "3001:3001"
    depends_on:
      - catalog
      - order
      - catalogcash
      - ordercash
    networks:
      - app-network

  catalog:
    build: ./catalog
    ports:
      - "3002:3002"
    volumes:
      - ./catalog:/catalog # Mount the catalog directory
    networks:
      - app-network

  order:
    build: ./order
    ports:
      - "3003:3003"
    volumes:
      - ./catalog:/catalog # Mount the same catalog directory for access
      - ./order:/order
    networks:
      - app-network
```

```

catalogcash:
  build: ./catalogcash
  ports:
    - "3004:3004"
  depends_on:
    - catalog
    - order
  volumes:
    - ./catalogcash:/catalogcash # Mount the catalog directory
  networks:
    - app-network

ordercash:
  build: ./ordercash
  ports:
    - "3005:3005"
  depends_on:
    - catalog
    - order
  volumes:
    - ./catalogcash:/catalogcash # Mount the same catalog directory for
access
  - ./ordercash:/ordercash
  networks:
    - app-network

networks:
  app-network:
    driver: bridge

```

Docker Containers(images) while running:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c7323d23e529	bazarcom-frontend	"docker-entrypoint.s..."	3 hours ago	Up 3 hours	0.0.0.0:3001->3001/tcp	bazarcom-frontend-1
74cb6116a58	bazarcom-catalogcash	"docker-entrypoint.s..."	3 hours ago	Up 3 hours	0.0.0.0:3004->3004/tcp	bazarcom-catalogcash-1
6eb982aff944	bazarcom-ordercash	"docker-entrypoint.s..."	3 hours ago	Up 3 hours	0.0.0.0:3005->3005/tcp	bazarcom-ordercash-1
a58eb8934a12	bazarcom-order	"docker-entrypoint.s..."	3 hours ago	Up 3 hours	0.0.0.0:3003->3003/tcp	bazarcom-order-1
1b415185698d	bazarcom-catalog	"docker-entrypoint.s..."	3 hours ago	Up 3 hours	0.0.0.0:3002->3002/tcp	bazarcom-catalog-1

For Running The Project, it's the same for Part1:

```
docker-compose up -d -- build
```