



Bazar Multi-tier Online Bookstore Part -2-
Distributed Operating Systems
10636456

10th. Dec. 2022

Masa koni 11820155

Lama Darawsheh 11819574

Program Design:

we modified part 1 to implement: replication, load balancing and caching.

- **Frontend server** responsible for forwarding requests to the two servers, and implements load balancing (using round-robin algorithm). It also has an in-memory cache to enhance the performance, for this we used a hash map for caching requests.
- To implement replication we made two replicas of both **Catalog server** and **order server**, and ran them on a VM with two catalog servers and two order server all with different ports.
- As for consistency, we used a push-based method, in which any of the catalog servers notify the other whenever a patch request for (quantity or cost) is received or whenever a book is purchased so that the other server make the changes to its database, too and this way both servers are always up to date.
- Also, whenever a change happen at any of the servers, it sends an invalidate request to the frontend server so that the frontend server removes the modified book from its cache in case it exists.

technologies: Node.js & express, mongoDB, Virtual box with Ubuntu running on it.

On MongoDB we made a database with a collection of: `_id`, `itemNumber`, `name`, `cost`, `topic` and `numberOfItems`.

Catalog servers:

- 192.168.56.101:3000
- 192.168.56.101:3005

Order servers:

- 192.168.56.101:3003
- 192.168.56.101:3004

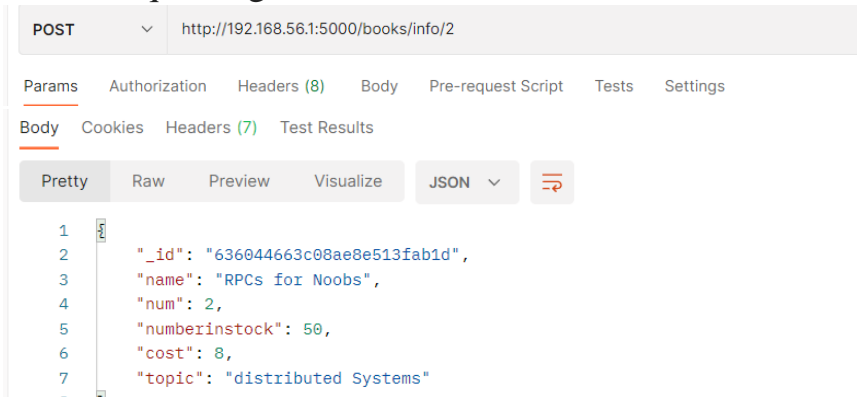
How it works:

- 1- The client only talks to the frontend server, and sends requests to localhost on port 5000 using a browser or postman.
<http://localhost:5000/>
- 2- If the frontend server receives:
 - a. A get requests and checks if it has the required information in cache, if it's not, it contacts one of the two replicas of the catalog servers and it chooses which server to contact using round robin.
 - b. A patch request, it forwards it to one of the replicas of the catalog server and it chooses which server to contact using round robin.
 - c. Purchase book request, it forwards it to one of the replicas of the order server and it chooses which server to contact using round robin.
- 3- If the catalog server replicas receive a get request, it returns the result from its database.
- 4- When any of the catalog server replicas receive a patch request (either from the frontend server or from the order server when a book is purchased), it makes the required changes to the database and then sends two requests:
 - A. notify the other replica to make the same change.
 - B. notify the frontend server to remove this book from the cache if it exists.
- 5- When any of the replicas of the catalog servers receives a notification message from the other replica, it applies the required changes

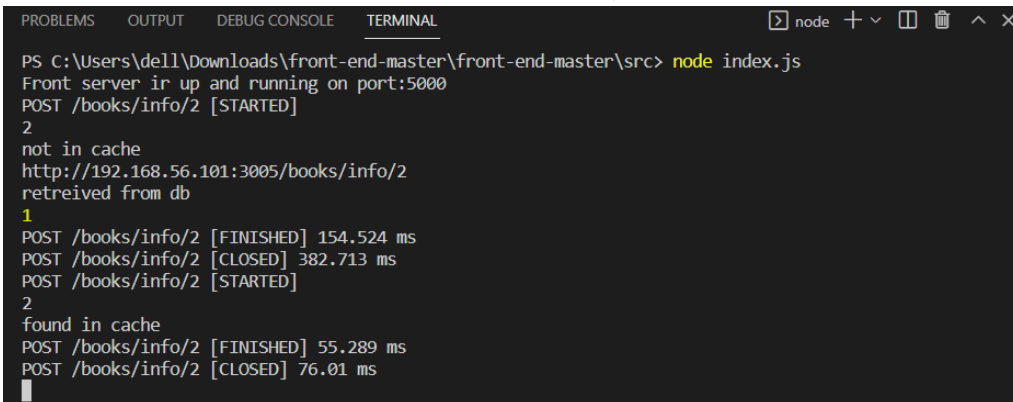
Here is a list of the available requests and their responses:

1. **Get information by id:** Request: localhost:5000/books/info/[id]

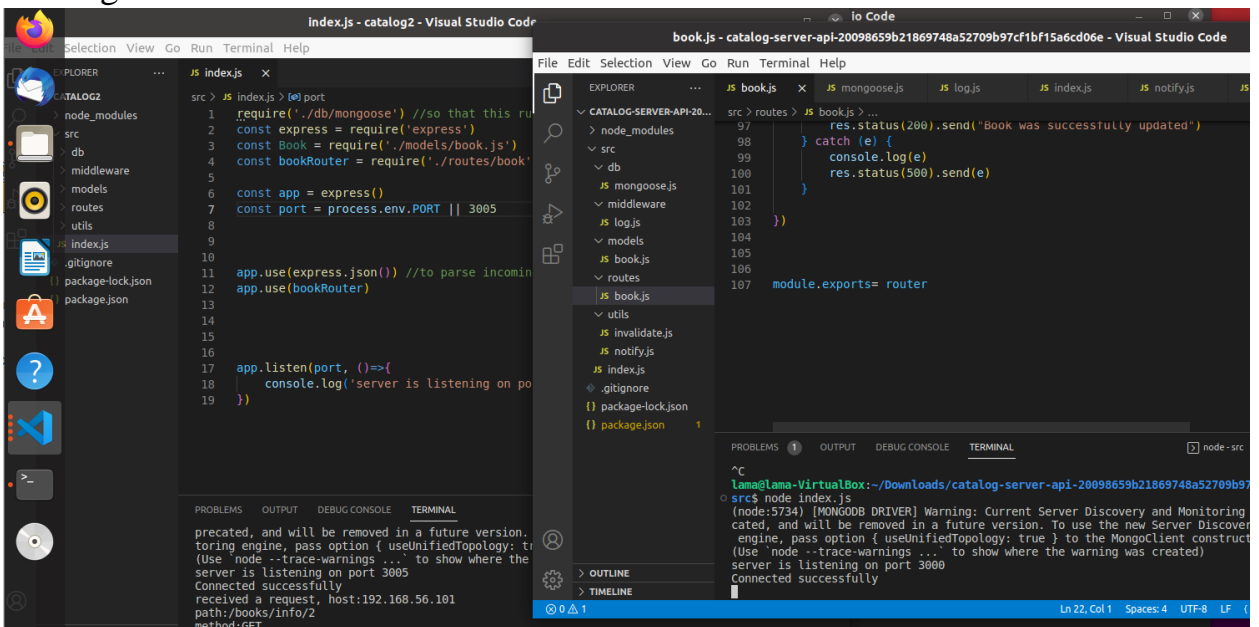
- Requesting information of book 2



Front-end server checks if book in cache, adds it if it is not



Catalog servers



2. Info of book 5, front server get it from catalog

POST	http://192.168.56.1:5000/books/info/5
Params	Authorization Headers (8) Body Pre-request Script
Query Params	
KEY	VALUE
Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "6390ca45d45c2c569e94eeb",
3   "name": "How to finish Project 3 on time",
4   "num": 5,
5   "numberinstock": 30,
6   "cost": 15,
7   "topic": "distributed Systems"
8 }
```

```
POST /books/info/2 [FINISHED] 55.289 ms
POST /books/info/2 [CLOSED] 76.01 ms
POST /books/info/5 [STARTED]
5
not in cache
http://192.168.56.1:3000/books/info/5
retrieved from db
2
POST /books/info/5 [FINISHED] 168.702 ms
POST /books/info/5 [CLOSED] 180.306 ms
```

One request got redirected to catalog 1 and the other to catalog 2

Visual Studio Code interface showing the development environment for a book catalog API. The left sidebar displays the Explorer with the project structure, including node_modules, src, db, middleware, models, routes, utils, index.js, .gitignore, package-lock.json, and package.json. The main editor shows the index.js file with code for connecting to MongoDB, setting up Express, and defining routes. The right sidebar shows the Explorer with the book.js file. The bottom panel shows the Output window with logs indicating the server is listening on port 3000 and has received a GET request for /books/info/2.

3. Request book 2 after adding it to cache.

```
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize JSON
1 [{"_id": "636044663c08ae8e513fab1d",
2   "name": "RPCs for Noobs",
3   "num": 2,
4   "numberinStock": 50,
5   "cost": 8,
6   "topic": "distributed Systems"}]
```

```
2
POST /books/info/5 [FINISHED] 168.702 ms
POST /books/info/5 [CLOSED] 180.306 ms
POST /books/info/2 [STARTED]
2
found in cache
POST /books/info/2 [FINISHED] 72.031 ms
POST /books/info/2 [CLOSED] 86.431 ms
```

frontend server

catalog servers did not receive any request

```
{} package.json 1
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
precated, and will be removed in a future version.
toring engine, pass option { useUnifiedTopology: tr
(Use 'node --trace-warnings ...' to show where the
server is listening on port 3005
Connected successfully
received a request, host:192.168.56.101
path:/books/info/2
method:GET
*****
```

4. Update numberinStock: frontend deletes book 2 from cache

```
PATCH http://192.168.56.1:5000/books/2
Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 [{"_id": "636044663c08ae8e513fab1d",
2   "name": "RPCs for Noobs",
3   "num": 2,
4   "numberinStock": 80,
5   "cost": 8,
6   "topic": "distributed Systems"}]
```

```
Body Cookies Headers (7) Test Results
Pretty Raw Preview Visualize HTML
1 Book was successfully updated
```

The image shows a VS Code editor with two files open: `index.js` and `book.js`. The `index.js` file defines the Express application, including database connection, middleware, and route definitions. The `book.js` file defines the `Book` Mongoose model and the `router` logic for handling book-related requests. The terminal at the bottom shows the application running on port 3005, with logs for incoming requests and responses.

```

src > .\index.js > {0} port
1 require('./db/mongoose') //so that this runs
2 const express = require('express')
3 const Book = require('./models/book.js')
4 const bookRouter = require('./routes/book')
5
6 const app = express()
7 const port = process.env.PORT || 3005
8
9
10
11 app.use(express.json()) //to parse incoming json to object.
12 app.use(bookRouter)
13
14
15
16
17 app.listen(port, ()=>{
18   console.log('server is listening on port ' + port)
19 })
20

```

```

src > routes > .\book.js > ...
97 res.status(200).send('Book was successfully updated')
98 }
99 catch (e) {
100   console.log(e)
101   res.status(500).send(e)
102 }
103 })
104
105
106
107 module.exports= router

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
http://192.168.56.101:3000/notify/books/2
http://192.168.56.101:3005/notify/books/2
received a request, host:192.168.56.101
path:/notify/books/2
method:PATCH
*****
got a notification to change, book id:2
received a request, host:192.168.56.101
path:/books/info/2
method:GET
*****
received a request, host:192.168.56.101
path:/notify/books/2
method:PATCH
*****
got a notification to change, book id:2

```

The screenshot shows the VS Code interface with a REST client request and response, and a terminal window.

REST Client Request:

```
GET http://192.168.56.1:5000/books/info/2
```

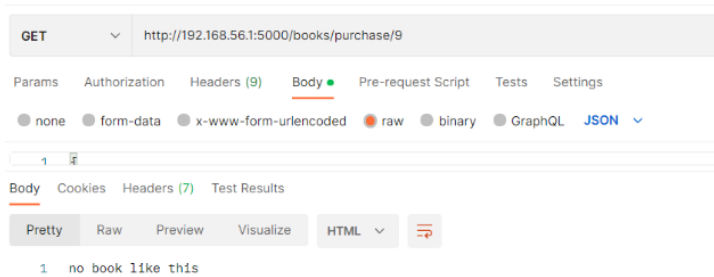
REST Client Response:

```
{
  "_id": "636944663c08ae8e513fabid",
  "name": "RPCs for Noobs",
  "num": 2,
  "numberinstock": 80,
  "cost": 8,
  "topic": "distributed Systems"
}
```

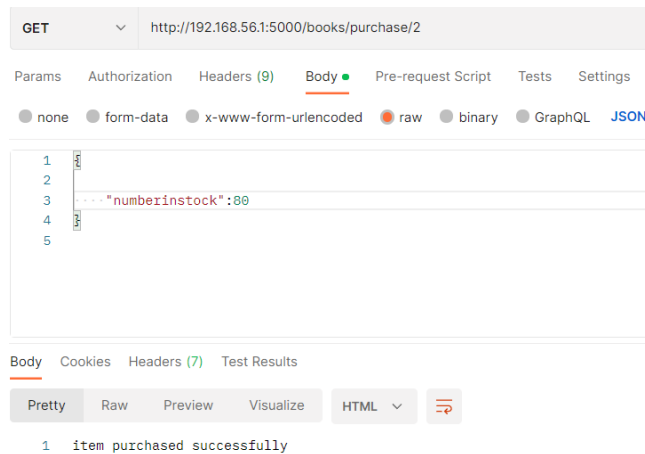
Terminal Output:

```
book changed, delete from cache
distributed Systems
current cache size:1
DELETE /invalidate/2 [FINISHED] 98.522 ms
DELETE /invalidate/2 [CLOSED] 106.774 ms
PATCH /books/2 [FINISHED] 392.997 ms
PATCH /books/2 [CLOSED] 401.265 ms
GET /books/info/2 [STARTED]
2
not in cache
http://192.168.56.101:3000/books/info/2
retrieved from db
2
GET /books/info/2 [FINISHED] 47.028 ms
GET /books/info/2 [CLOSED] 49.408 ms
GET /books/info/2 [STARTED]
```

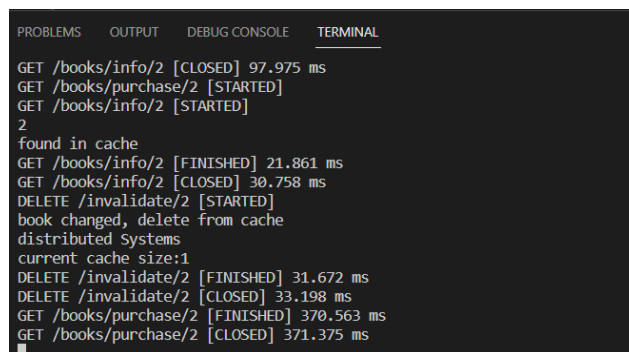
5. **Purchase book:** Request: localhost:5000/books/purchase/[id]
Before sending the purchase request.
If no book found with this id:



- Sending a request to buy book 2



Frontend




```
17 app.listen(port, ()=>{
18   console.log('server is listening on port ' + port)
19 })
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
http://192.168.56.101:3000/notify/books/2
http://192.168.56.101:3005/notify/books/2
received a request, host:192.168.56.101
path:/notify/books/2
method:PATCH
*****
got a notification to change, book id:2
received a request, host:192.168.56.101
path:/books/info/2
method:GET
*****
received a request, host:192.168.56.101
path:/notify/books/2
method:PATCH
*****
got a notification to change, book id:2
[ ]
JS index.js
.gitignore
package-lock.json
package.json 1
*****
received a request, host:192.168.56.101
path:/books/2
method:PATCH
*****
http://192.168.56.101:3000/notify/books/2
http://192.168.56.101:3005/notify/books/2
received a request, host:192.168.56.101
path:/notify/books/2
method:PATCH
*****
got a notification to change, book id:2
received a request, host:192.168.56.101
path:/books/info/2
method:GET
*****
Ln 22, Col 1 S
```

The image shows a VS Code interface. The foreground terminal window displays a REST client request and response. The request is a GET to 'http://localhost:3000/books/purchase/2' with a 'Content-Type: application/json' header. The response is a 200 OK with a JSON body: { msg: 'available', count: 80 }. The background terminal window shows the command 'node index.js' being executed, with output indicating a successful MongoDB connection and a warning about the deprecated ring engine.

- Searching for a book under title distributed systems



How to run Program:

1. Install node.js and Express framework on the two machines.
2. Build a database on monogodb with a collection containing the books.
3. Open the frontend terminal and start the server by typing: npm start
4. Open the catalog terminal and start the server by typing: npm start
5. Open the order terminal and start the server by typing: npm start
6. Start sending requests from postman/browser on localhost:5000.

Code:

Frontend server: <https://github.com/MasaKni/front-end>

Catalog server: <https://github.com/lamadarawsheh/CatalogServer>

Order server: <https://github.com/lamadarawsheh/orderServer>