**Distributed operating systems**

**Lab2**

**Mustafa Ramahi**

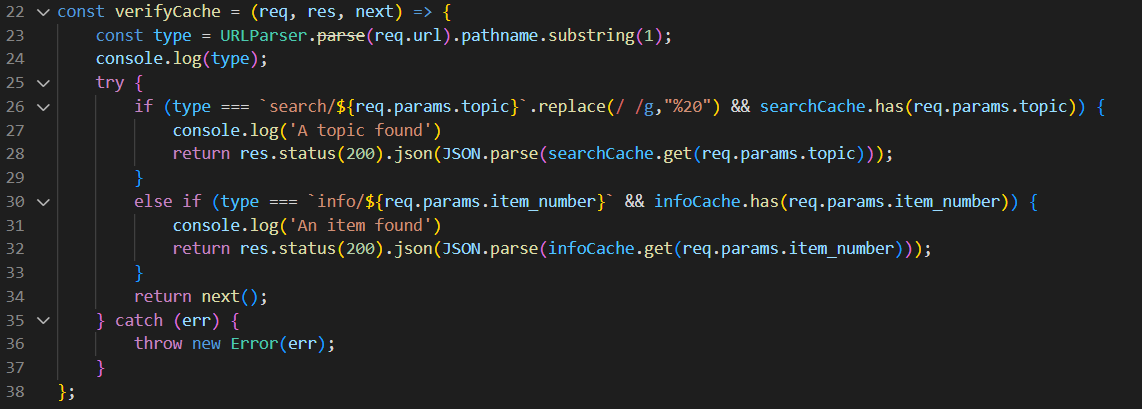
**Id:11923890**

* **Code explanation:**

1. **Front-end server**

In our NodeJS application, we make use of two important modules - "round-robin-js" for load balancing using the round-robin technique and "Node-cache" for caching requests on our front-end server. The load balancing helps distribute incoming requests evenly among different servers, and the caching optimizes response times by storing and retrieving data from memory.

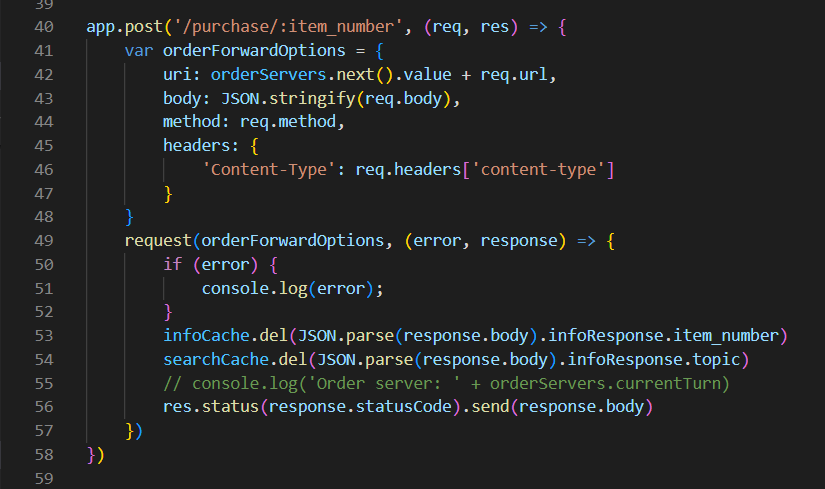
The following code snippet shows a middleware that used to check if the request is already on the cache memory or not; if it's, the response data will be returned from the data in the cache.



The following code shows a middleware for the purchasing, once a book purchase is done, the data related to that book will be remove from the caches.

Also the order server which the request will be forwarded to is chosen when this line is executed:

orderServers.next().value



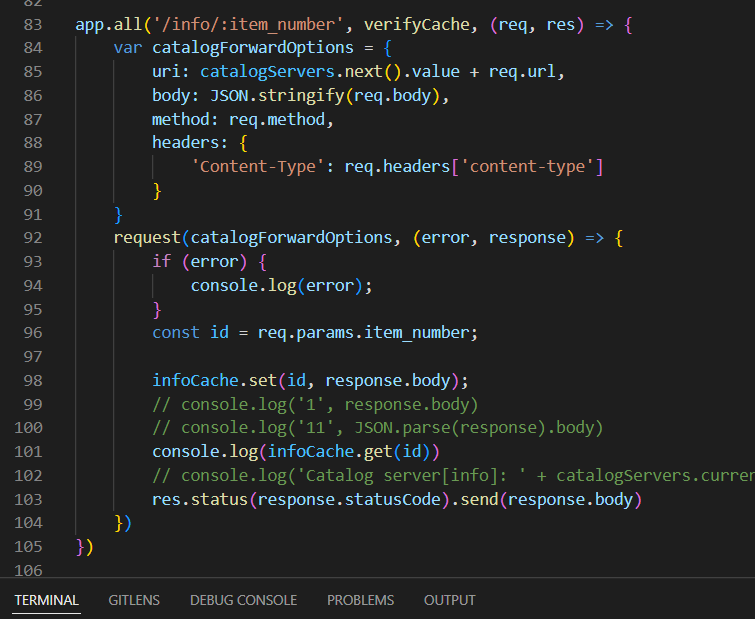
The next figure shows the search operation which is called only if the data was not in the cache,

The catalog server is chosen when line 63 of the code below is executed.

Saving the response data for the searching request is done in line 75.



The next figure shows the same technique but this time it is for searching for a specific book:



1. **Order server:**

**Our system is set up to handle book purchases. Here's a simplified explanation of the process:**

**1.First, we check if a book is available by sending an information request.**

**2.If the book is available, we proceed to update its stock in both catalog servers.**

**3. and when I ordered book I store the information for this process in order.csv**

**3.If the stock update is successful, we provide a response containing the order details. As shown below:**

app.post('/purchase/:item\_number', (req, res) => {

    console.log('ord1')

    var item\_number = req.params.item\_number

    var clientServerOptions = {

        uri: catalogIP + 'info/' + item\_number,

        body: '',

        method: 'GET',

        headers: {

            'Content-Type': 'application/json'

        }

    }

    request(clientServerOptions, (error, infoResponse) => {

        if (infoResponse && infoResponse.statusCode == 200) {

            clientServerOptions = {

                uri: catalogIP + 'update/' + item\_number,

                body: JSON.stringify({ itemsInStock: -1 }),

                method: 'PUT',

                headers: {

                    'Content-Type': 'application/json'

                }

            }

            request(clientServerOptions, (error, UpdateResponse) => {

                if (UpdateResponse.statusCode == 200) {

                    clientServerOptions.uri = catalog2IP + 'update/' + item\_number;

                    request(clientServerOptions, (error, UpdateResponse) => {

                        if (UpdateResponse.statusCode == 200) {

                            const orderRequest = {

                                uri: order2IP + 'addOrder/' + item\_number,

                                body: '',

                                method: 'POST',

                                headers: {

                                    'Content-Type': 'application/json'

                                }

                            }

                            request(orderRequest, (error, UpdateResponse) => {

                                if (UpdateResponse.statusCode == 200) {

                                    res.send({ orderID: orders.addOrder(item\_number), infoResponse: JSON.parse(infoResponse.body) })

                                } else {

                                    res.status(400).send()

                                }

                            })

                        } else {

                            res.status(400).send()

                        }

                    })

                }

                else {

                    res.status(400).send()

                }

            })

        }

        else {

            res.status(400).send()

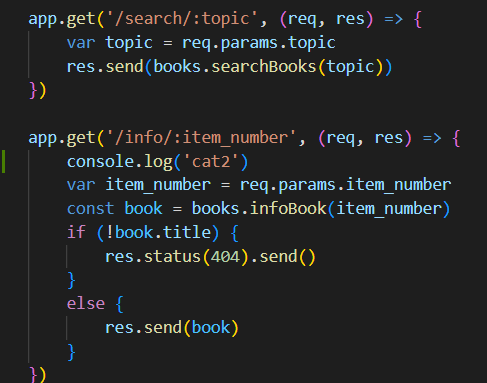
        }

    });

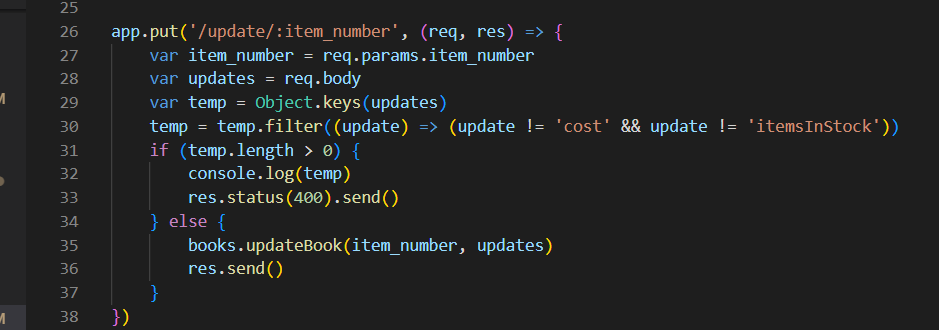
})

1. **Catalog server**

The searching requests (info or search about a topic) are shown in the next snapshot:



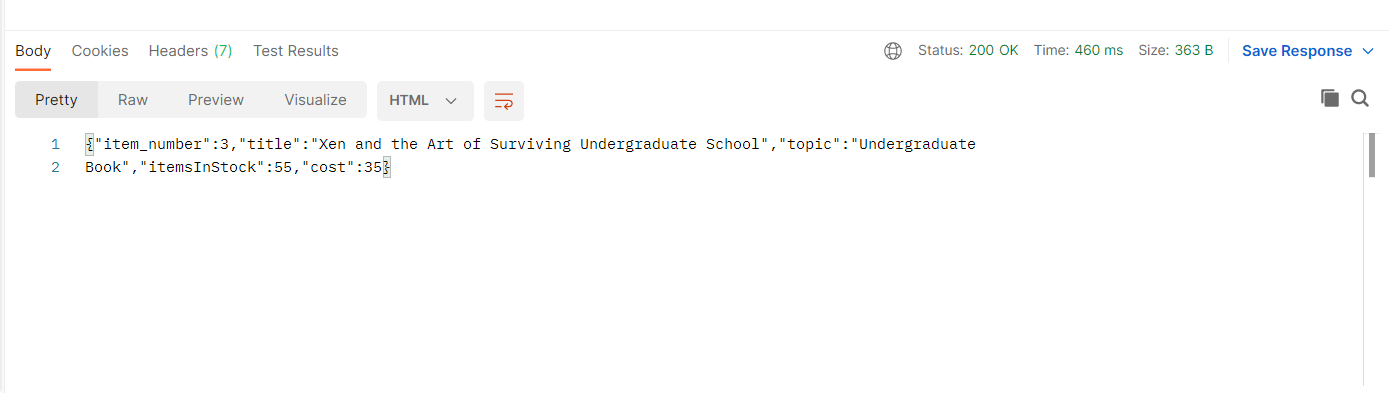
And to update a book we used the following middleware:



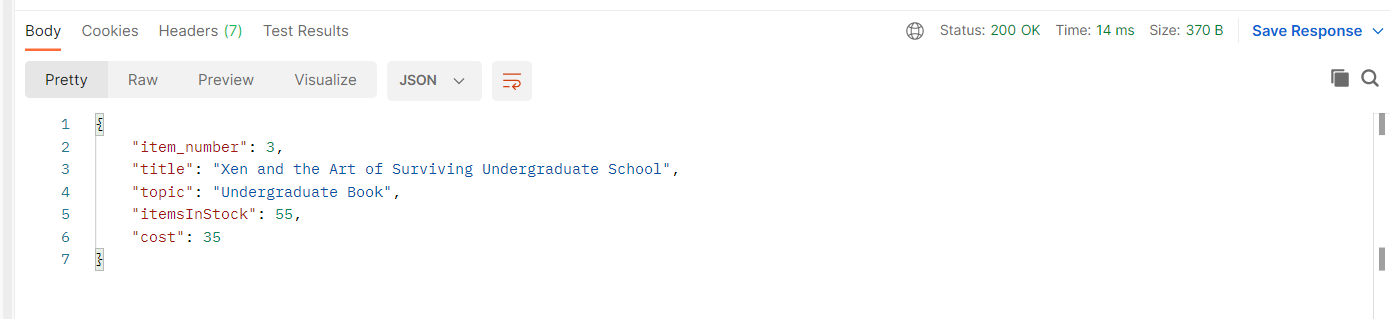
* **Results**

The response time for a getting request for the first time was  **higher** than when the same request was stored in a cache.

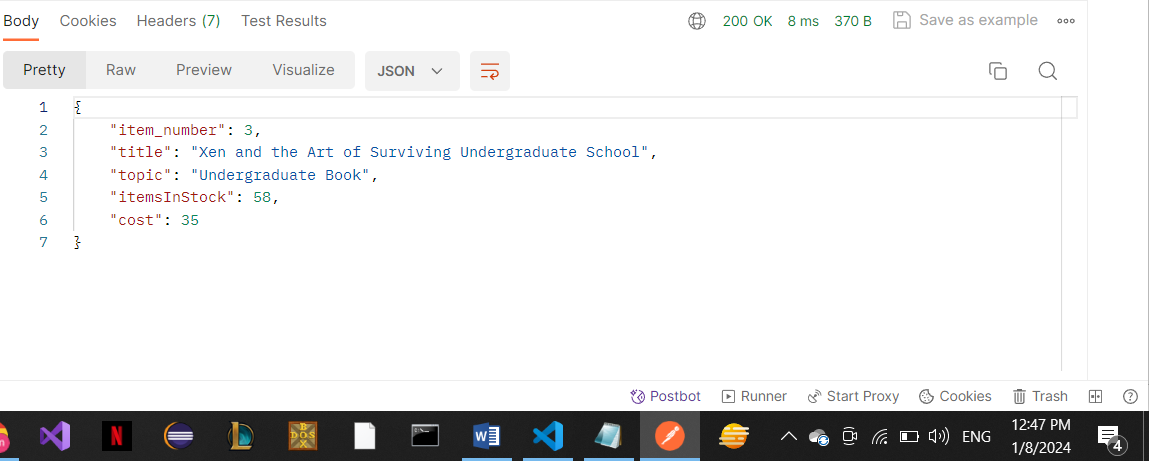
1. **Response time before caching:**

****

1. **Response time after caching**

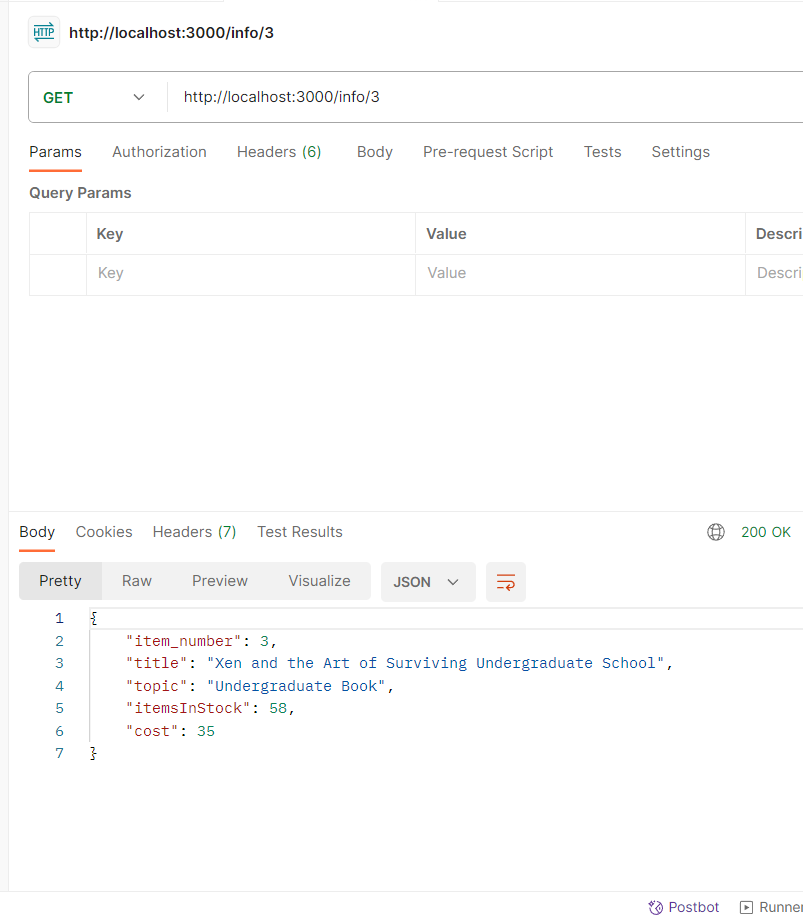
****

**When I try more then one:**

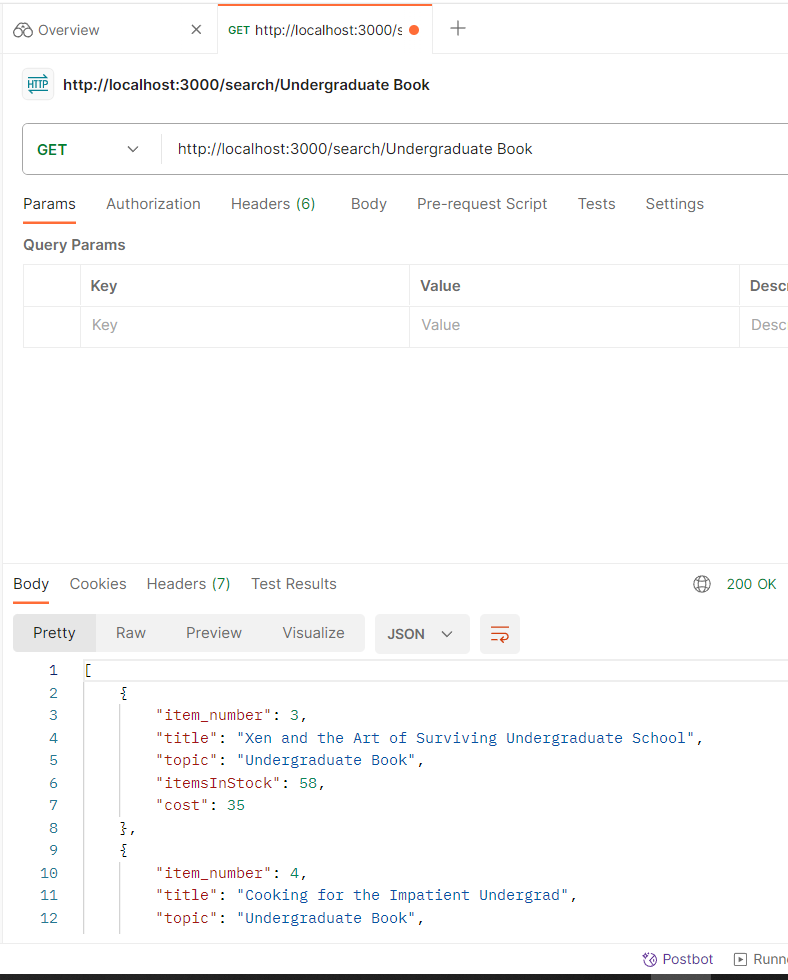
****

**My output :**

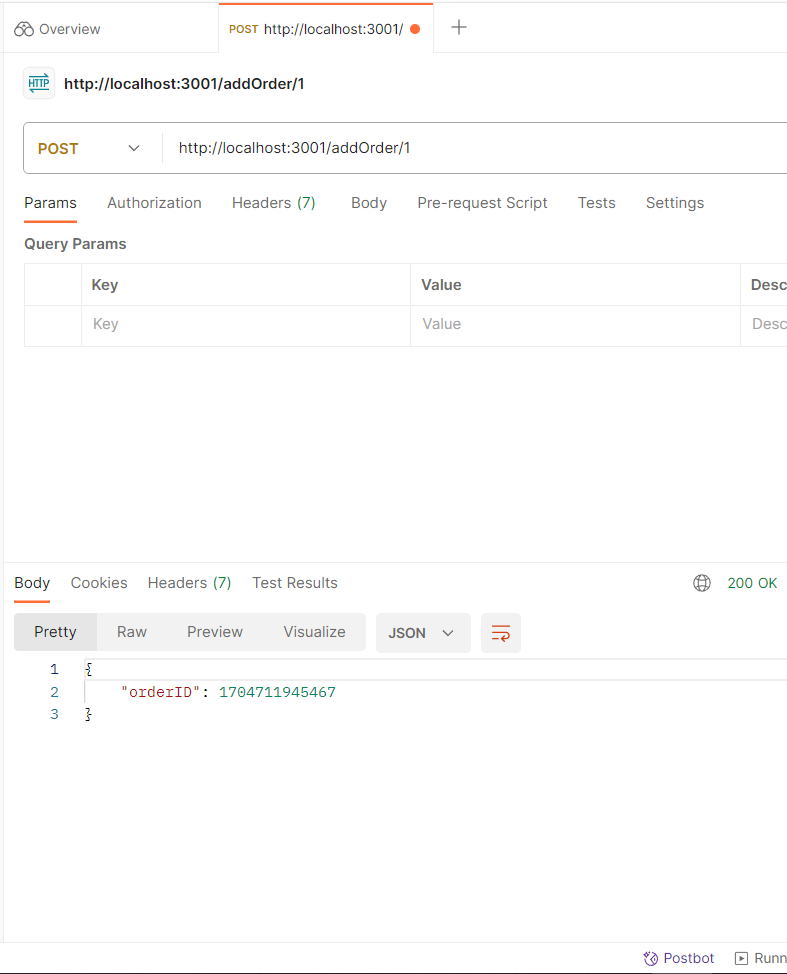
**Info:**

****

**Search:**

****

**Add Order:**

****