**Distributed operating systems**

**Part 2**

**Ahmad Masri**

**Mohammad Kukhun**

* **Code explanation:**

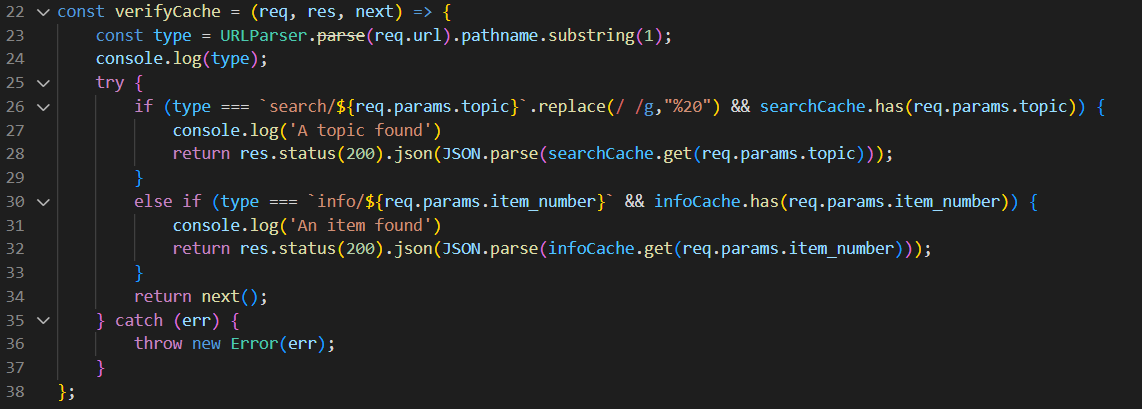
We have 5 machines, one for each server and replica.

1. **Front-end server**

NodeJS supports library called **round-robin-js** that used to implement load balancing using round robin technique. We used it.

In addition to **round-robin-js** module, NodeJS also supports **Node-cache** module, it used for caching the requests in the front-end server. We used two instances from it, one instance for order server and another for catalog server.

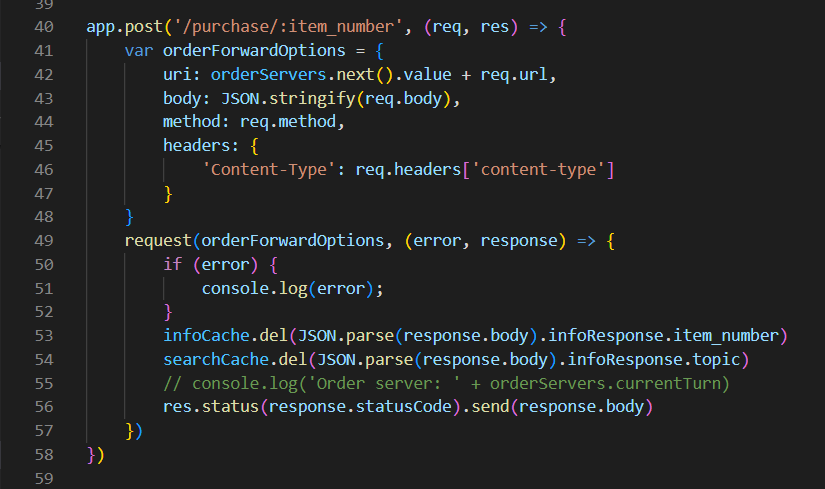
The following code snippet shows a middleware that used to check if the request is already on the cache memory or not; if it's, the response data will be returned from the data in the cache.



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



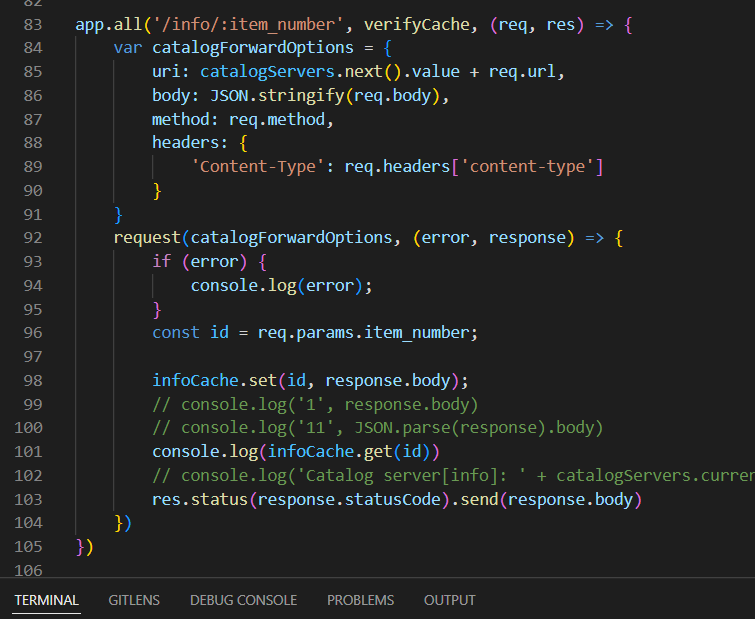
The next figure shows the search operation which is called only if the data was not in the cache,

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:**

**Supports handling book purchasing,**

**First it sends an ifo reques to figure if the book is available or not, if it's, it will send a request to update that book in the stock in both catalog servers, if that was done successfully will return a response of the order details. As shown in the code 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()

        }

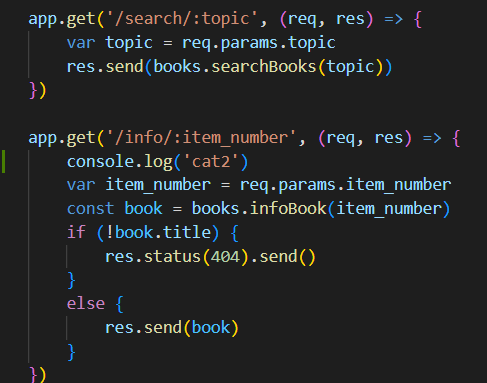
    });

})

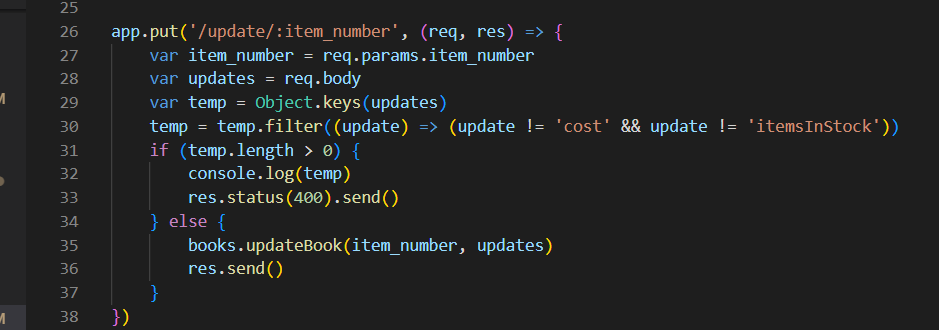
The previous code is the same for the second order server

1. **Catalog server**

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



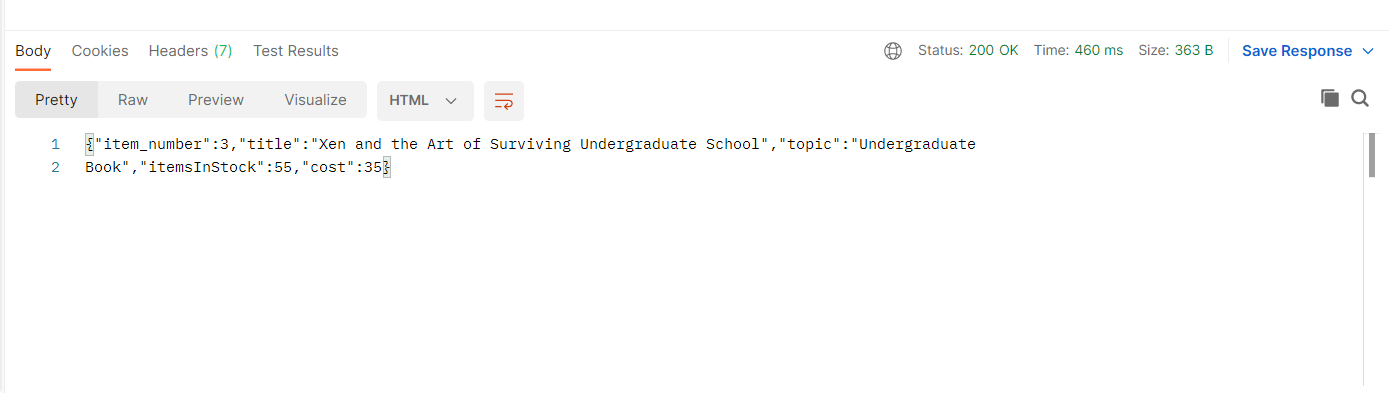
And to update a book we used the following middleware:



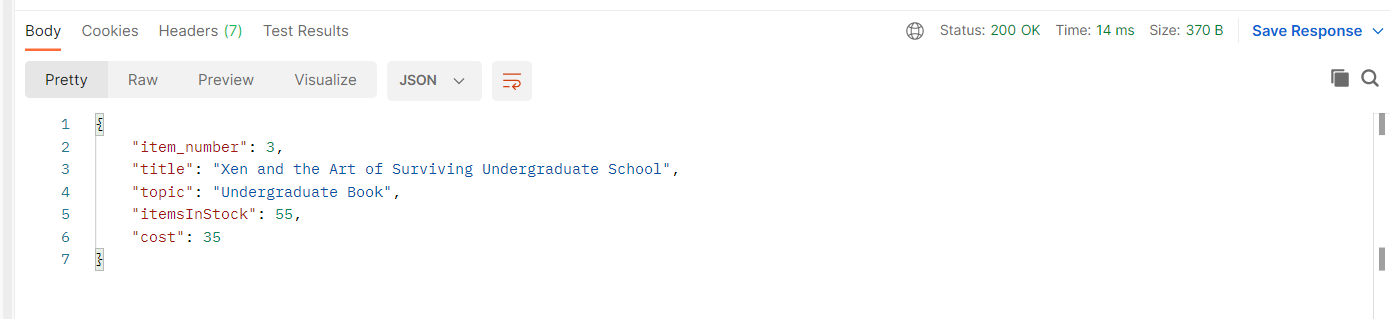
* **Results**

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**

****