DOS Project Report Part 2

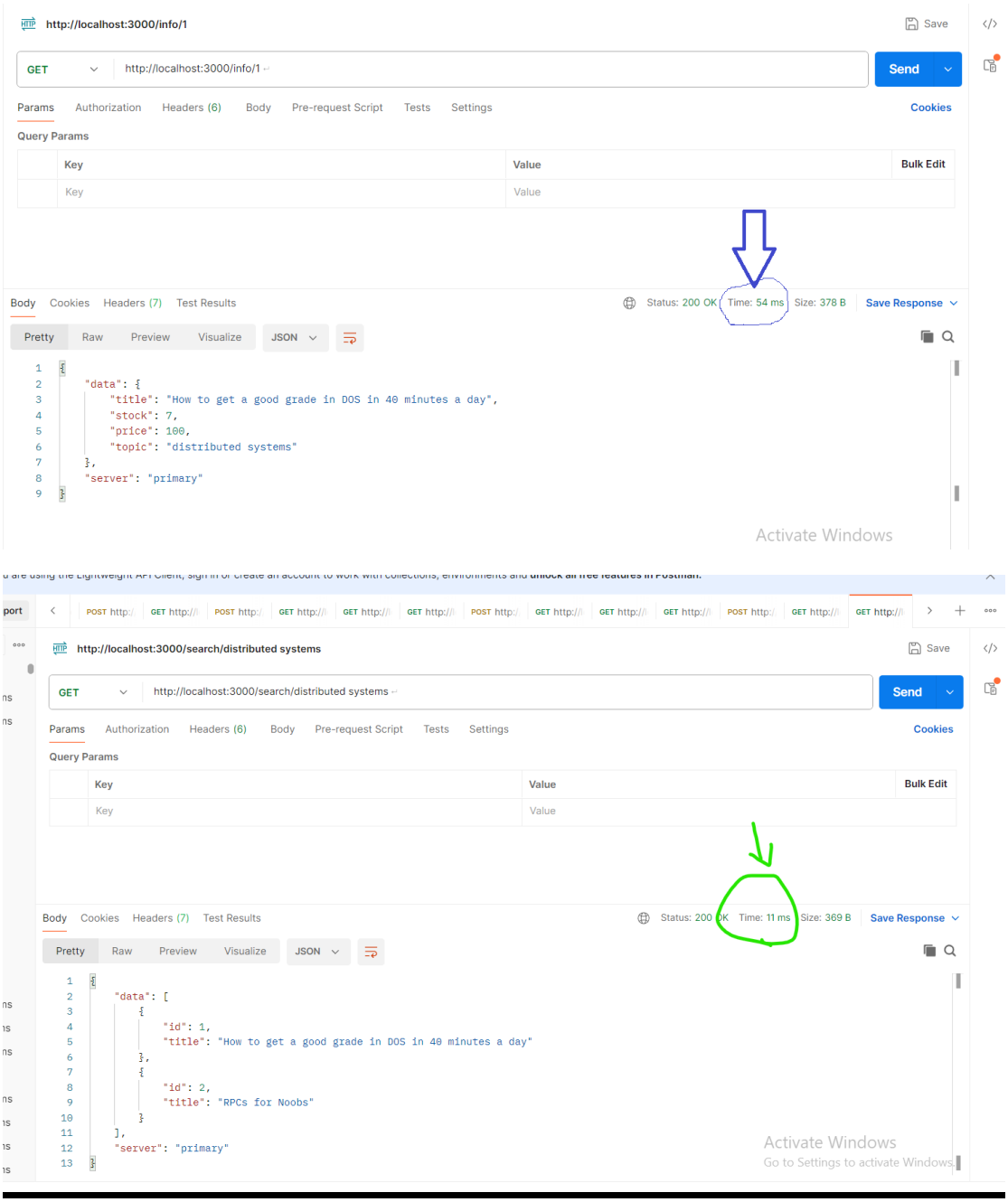
Student Name: Lama Qawareeq

Student Name: Anwar Aqraa

Part1: Cache Consistency


```
// Search books by topic
app.get('/search/:topic', async (req, res) => {
    const { topic } = req.params;

    // Check cache
    if (cache[`search-${topic}`]) {
        console.log(`Cache hit for search topic "${topic}"`);
        return res.json({ data: cache[`search-${topic}`], server: 'cache' });
    }

    const server = chooseServer(catalogServers);

    try {
        const response = await axios.get(`${server}/search/${topic}`);
        cache[`search-${topic}`] = response.data; // save respo
        console.log(`Cache updated for search topic "${topic}"`);

        res.json({ data: response.data, server: server.includes('replica') ? 'replica' : 'primary' });
    } catch (error) {
        console.error(`Error fetching search results for topic "${topic}":`, error.message);
        res.status(500).send('Error fetching search results');
    }
});
```

When I send GET request the first time, before Caching the data:



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

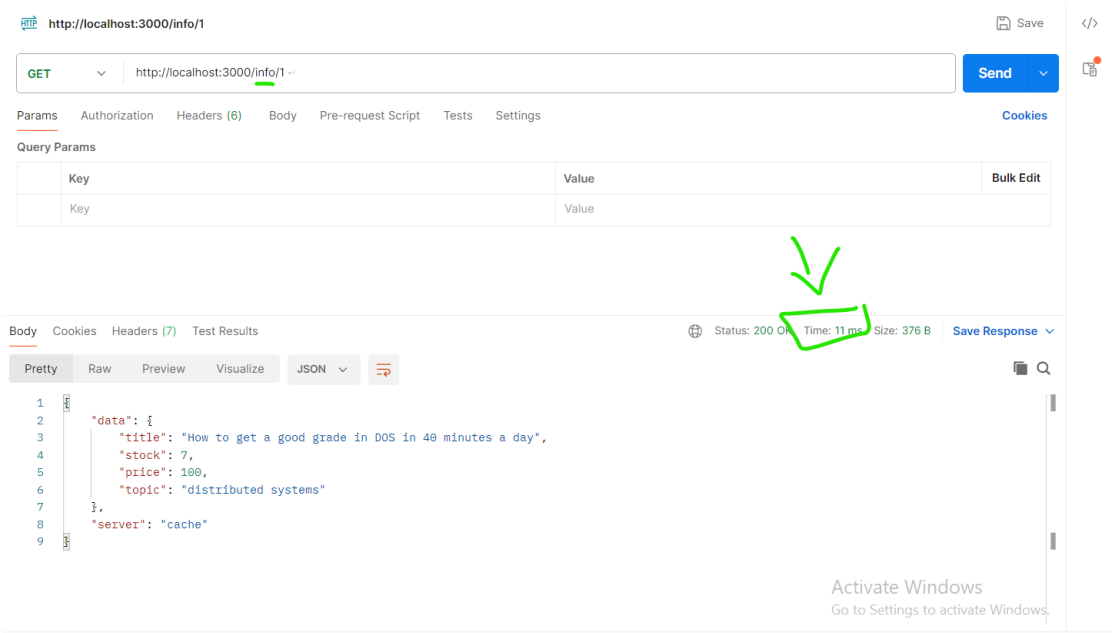
What is the response time with and without caching?

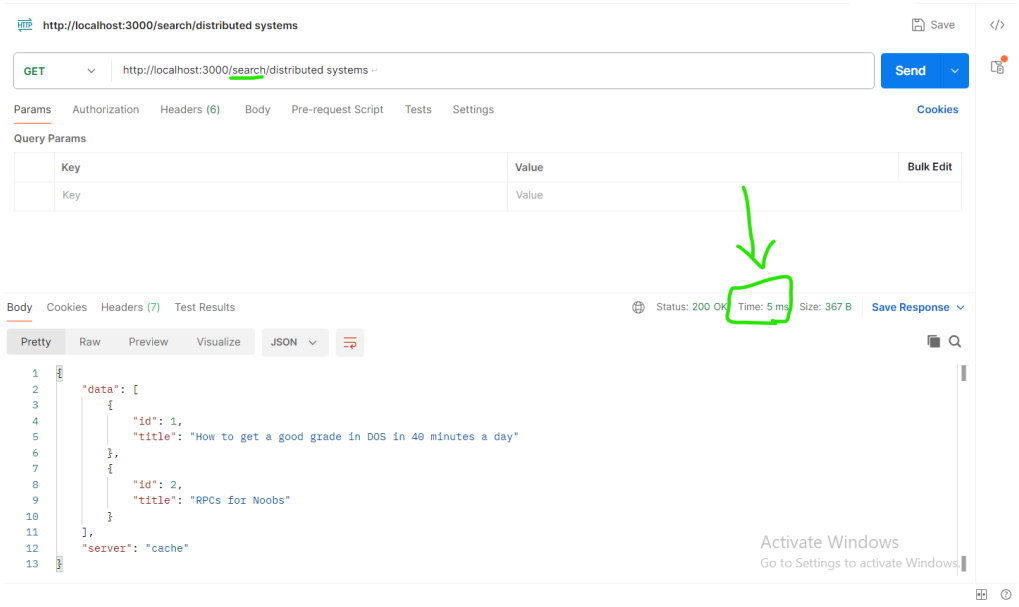
Answers：

for info: 54 ms

for search: 11 ms

With Cache:





Q2) How much does caching help?

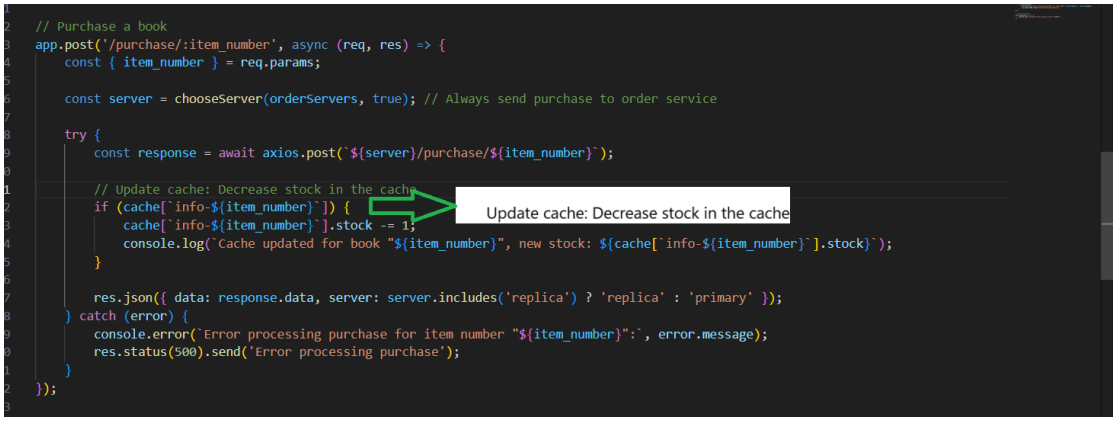
Answers:

for info: 11ms, 54/11-> 4.9 Faster than without cache.

for search: 5ms, 11/5-> 2.2 Faaster than without using cache.

Invalidate

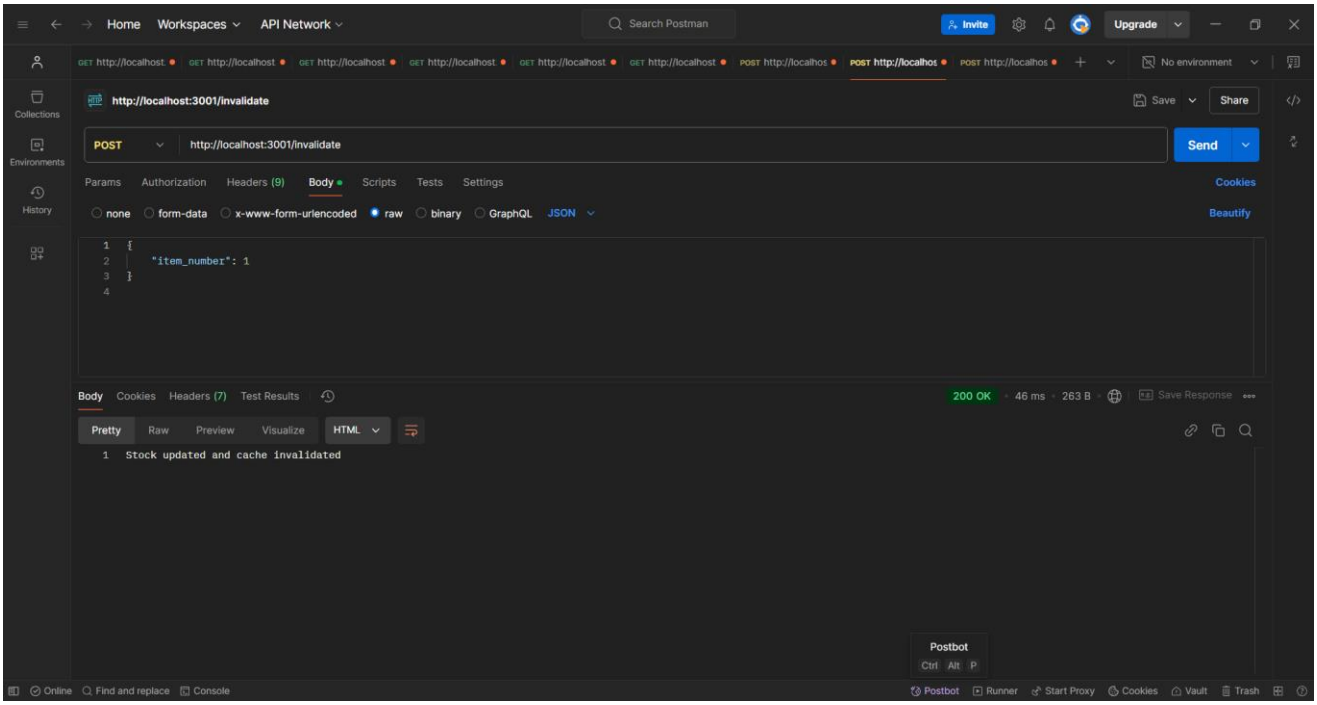
When purchasing, the data is disabled in cash until the new data values are taken after updating the inventory.



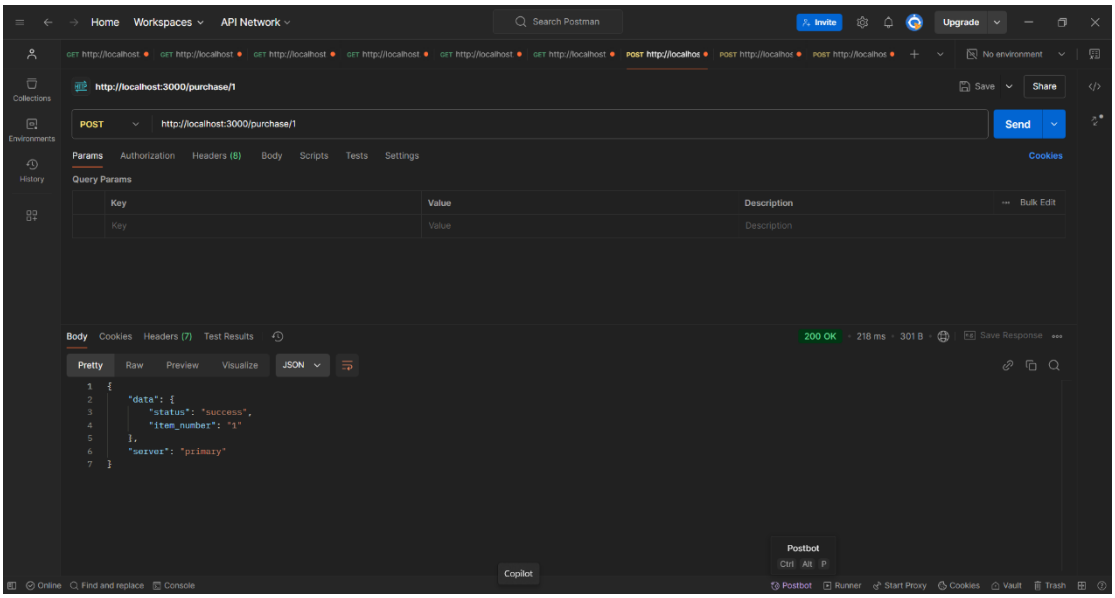
The code in (catalog & catalog replica) updates the stock of a specific item in the database by decreasing it by 1, then removes the cached data for that item if it exists. The goal is to ensure that the cached data stays consistent with the database after any changes.

```
app.post('/invalidate', (req, res) => {
    if (!item_number) {
        return res.status(400).json({ error: 'item_number is required' });
    }

    // First, update stock in the database
    db.get("SELECT stock FROM books WHERE id = ?", [item_number], (err, row) => {
        if (err) {
            console.error('Error fetching book from database:', err);
            return res.status(500).send('Error updating stock');
        }

        if (row) {
            const updatedStock = row.stock - 1; // Decrease stock by 1

            // Update stock in the database
            db.run("UPDATE books SET stock = ? WHERE id = ?", [updatedStock, item_number], (updateErr) => {
                if (updateErr) {
                    console.error('Error updating stock:', updateErr);
                    return res.status(500).send('Error updating stock');
                }

                // Cache invalidation
                if (cache[`info-${item_number}`]) {
                    delete cache[`info-${item_number}`];  // Invalidate cache
                    console.log(`Cache invalidated for item ${item_number}`);
                } else {
                    console.log(`Cache item ${item_number} not found`);
                }

                res.send('Stock updated and cache invalidated');
            });
        } else {
            console.log(`Item ${item_number} not found in database`);
            res.status(404).send('Item not found');
        }
```

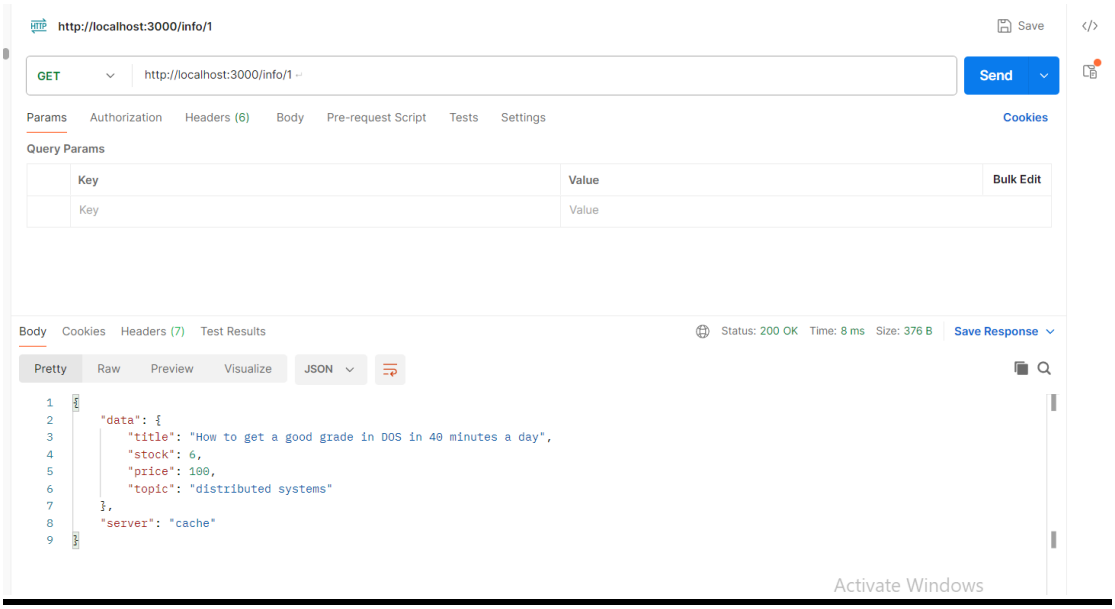
Test invalidate in postman:



We made the purchase:



The cashe inventory has been updated from 7 to 6 as shown in the following image:



Exoirement run	Without cashe	With cache	#of tims when using cache speed
1	17	***6***	***17/5=3.4 times***
2	***20***	***5***	***4***
3	***15***	***5***	***3***

Part2: Loadbalance with NGINX

I Used Nginx to acheive loadbalance, each service exist in it's seperate Docker Container and it has own Inerface & Port to communicate with other services, Below my File Configuration for NGINX.

We did the following function to distribute the load between the original server and the replica:

```
// Server URLs
const catalogServers = [
    'http://catalog-server:3001',
    'http://catalog-replica:3003',
];

const orderServers = [
    'http://order-server:3002',
    'http://order-replica:3004',
];

let catalogIndex = 0;
let orderIndex = 0;

const chooseServer = (servers, isOrder = false) => {
    const index = isOrder ? orderIndex++ : catalogIndex++;
    if (isOrder) orderIndex = orderIndex % orderServers.length;  //// Update the pointer after reaching the end of the list
    else catalogIndex = catalogIndex % catalogServers.length;
    return servers[index % servers.length];
};
```

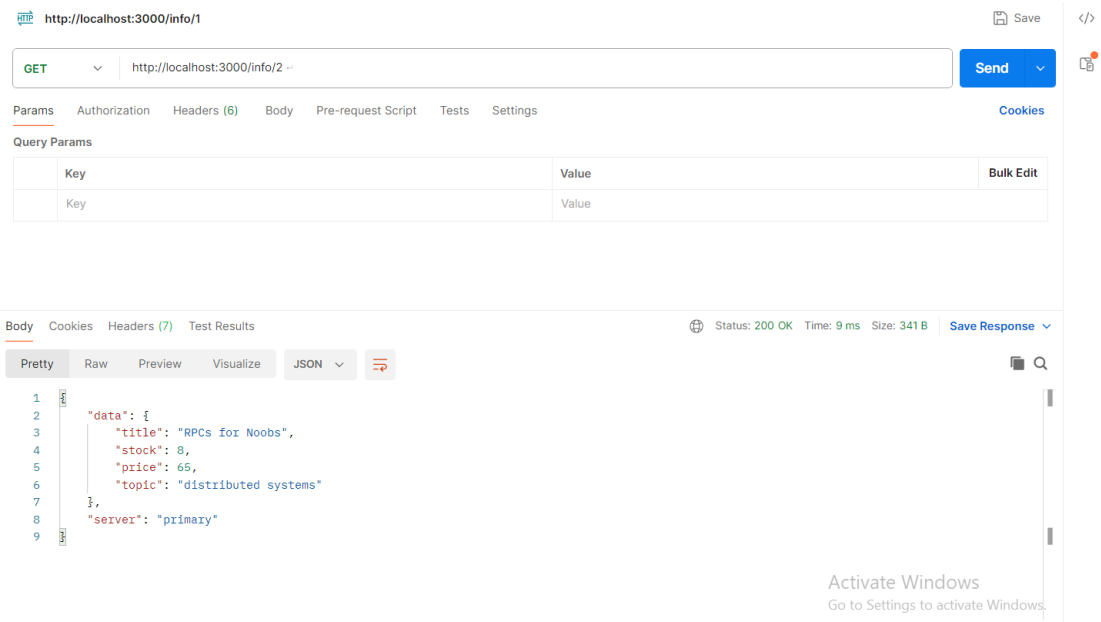
```
// Get book info
app.get('/info/:item_number', async (req, res) => {
    const { item_number } = req.params;

    // Check cache
    if (cache[`info-${item_number}`]) {
        console.log(`Cache hit for book info "${item_number}"`);
        return res.json({ data: cache[`info-${item_number}`], server: 'cache' });
    }

    const server = chooseServer(catalogServers);

    try {
        const response = await axios.get(`${server}/info/${item_number}`);
        cache[`info-${item_number}`] = response.data; // Cache the response

        console.log(`Cache updated for book info "${item_number}"`);

        res.json({ data: response.data, server: server.includes('replica') ? 'replica' : 'primary' });
    } catch (error) {
        console.error(`Error fetching book info for item number "${item_number}":`, error.message);
        res.status(500).send('Error fetching book info');
    }
});
```

here the function is called to distribute the load and based on the server that is chosen we request it through the API

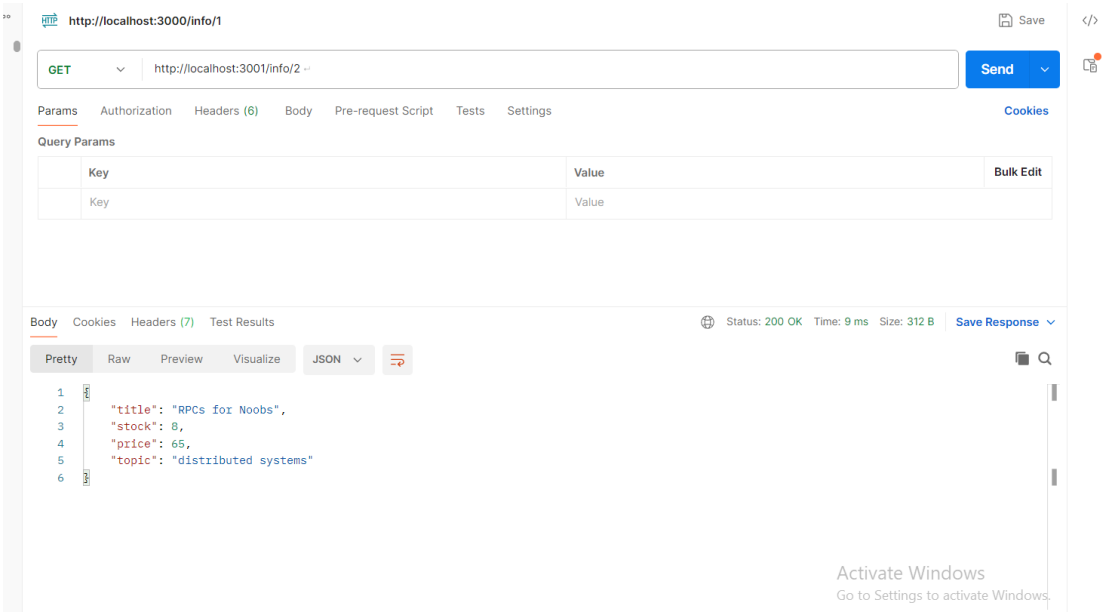
The same thing was done for info according to the id and for search according to the topic.

- The purchase, info or search process can be distributed to more than one server as shown in the following images:

- From the original server "front end server" (Port 3000):



From the catalog server (Port 3001):



From the replica server (Port 3003):