“An-Najah National University”

“Faculty of Engineering”

“Computer Engineering Department”

DOS Project Part-2

Preparing By

Aisha Ishtayeh “12028269“

- **Part1: Cache Consistency:**

we'll use an in-memory caching library like 'node-cache'

we have three state for the cache:

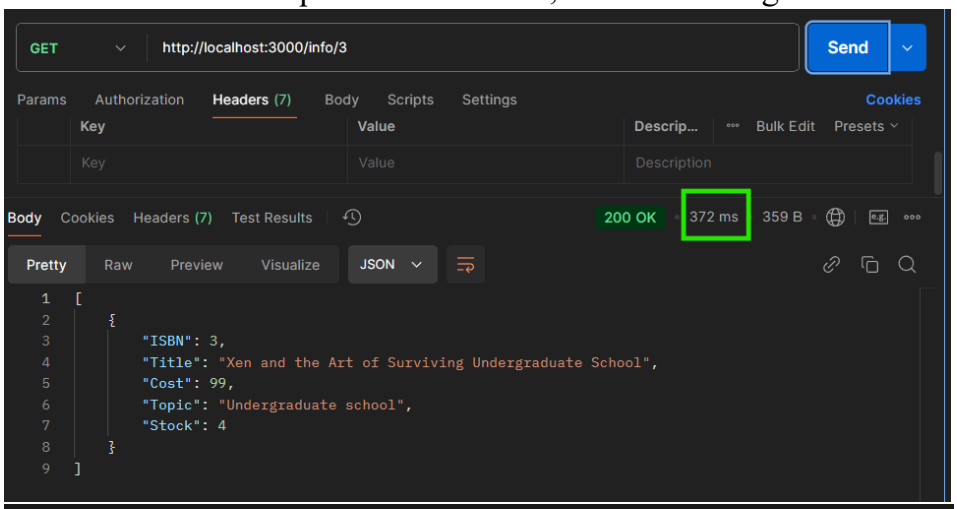
1. **Cache Hit**: use the same URL twice to confirm the second request is served from the cache.
2. **Cache Miss**: use a URL not in the cache, confirm it fetches data from the catalog/order server, and then stores it in the cache.
3. **Cache Invalidation**: Update a book's stock and confirm the cache entry is invalidated.

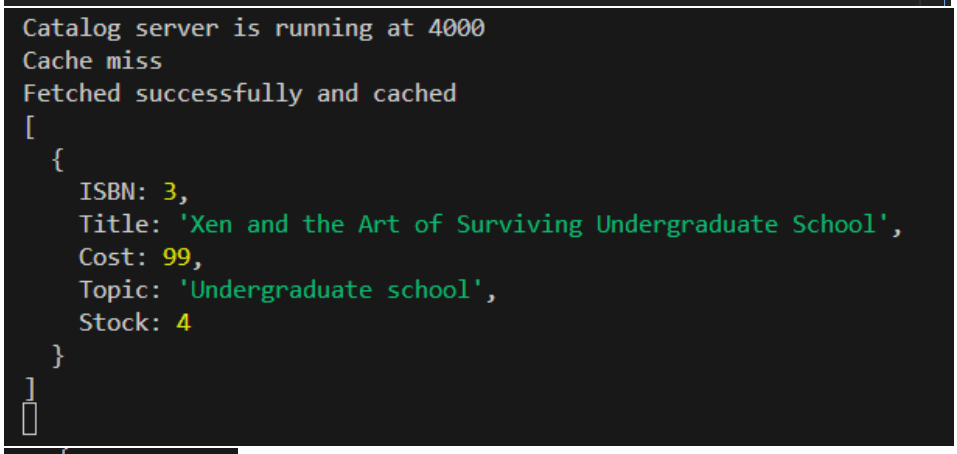
Applying the cache in catalog server:

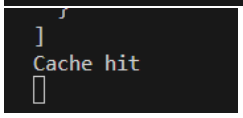
```
const app = express(); // Create express app
const cache = new NodeCache({ stdTTL: 3600, checkperiod: 600, maxKeys: 100 }); // Cache with a 1-hour TTL
const port = 4000; // The port for the front-end server is 4000

app.use(express.json()); // Middleware to parse incoming JSON data

// Search by topic (uses cache)
app.get('/search/:topic', (req, res) => {
  const topic = req.params.topic;

  // Check cache
  const cachedData = cache.get(topic);
  if (cachedData) {
    console.log('Cache hit');
    return res.json(cachedData);
  }

  console.log('Cache miss');
  // Fetch from database if not in cache
  DatabaseConfig.searchTopic(topic, (err, data) => {
    if (err) {
      res.status(500).send('Error fetching data from database'); // Error handling
    } else {
      cache.set(topic, data); // Cache the result
      console.log('Fetched successfully and cached');
      console.log(data);
      res.json(data);
    }
  });
});
```

And do the same in order server.

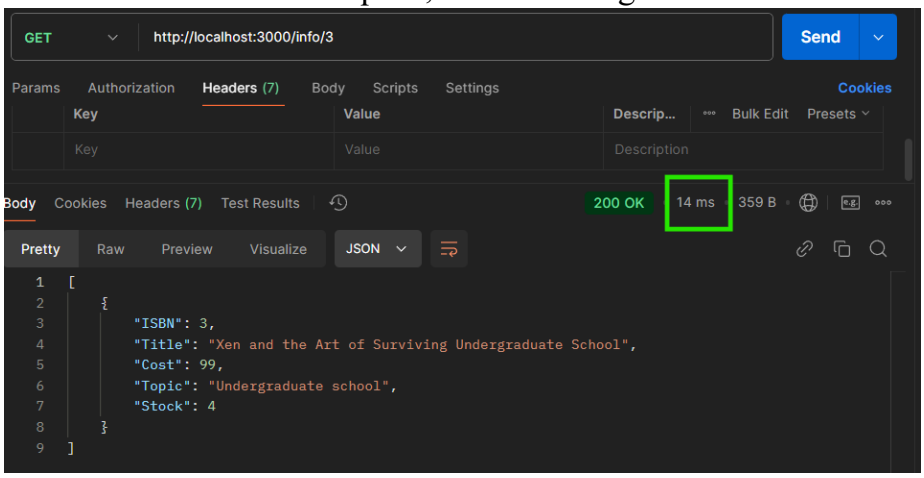
- When I send GET request the first time, before Caching the data:





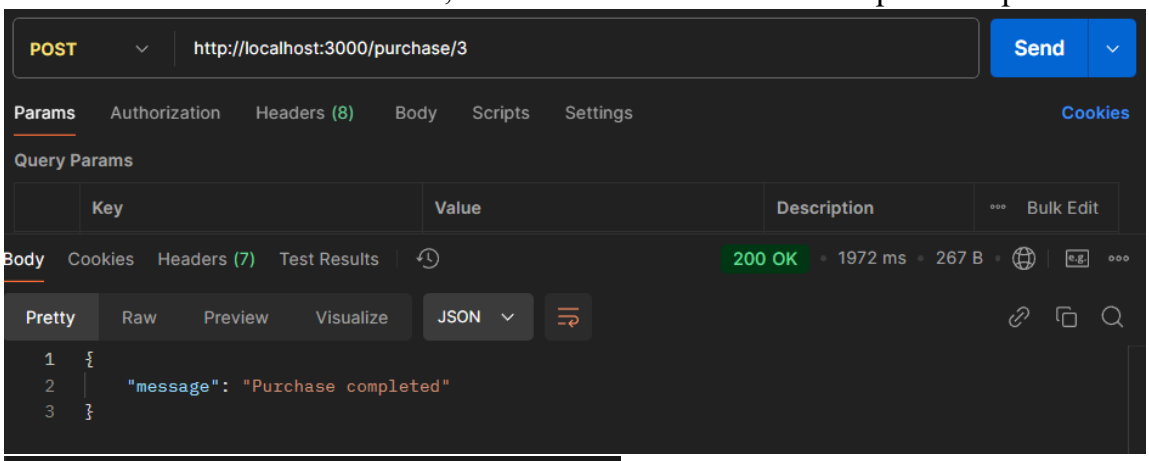


- When I send the Same request, “with caching”:



- How much does caching help?
  Answers
  - For info: 372/14 —> 26.57 Faster with cache

- And for the Cache Invalidation , when the stock of an item is updated “ purchase”:




```
Updating stock: 3
Stock updated successfully
Cache invalidated for item: 3
```

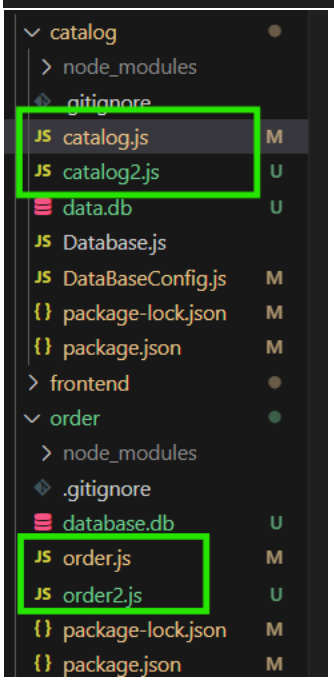
-so when I get the info for this item again , it will be without the cache:

```
Cache miss
Fetched successfully and cached
[
  {
    ISBN: 3,
    Title: 'Xen and the Art of Surviving Undergraduate School',
    Cost: 99,
    Topic: 'Undergraduate school',
    Stock: 2
  }
]
```

- **Part2: Replication:**

  To handle the replication setup, we'll implement load balancing for the replicated catalog and order servers in front end server , and Iam using for the **Load Balancing Algorithm**: 'Round-robin'

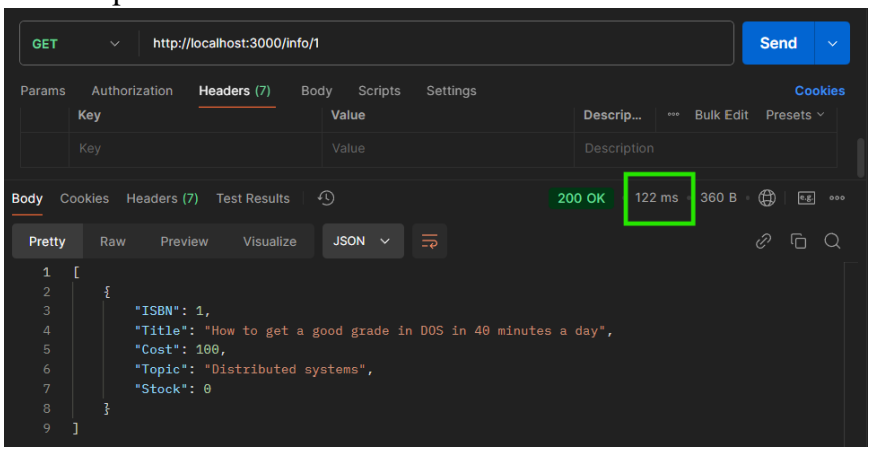
```
// Replica servers for catalog and order
const catalogReplicas = ["http://catalog:4000", "http://catalog2:4001"];
const orderReplicas = ["http://order:5000", "http://order2:5001"];

// Indexes for round-robin load balancing
let catalogIndex = 0;
let orderIndex = 0;

// Round-robin function
Codeium: Refactor | Explain | ×
function getNextReplica(replicas, currentIndex) {
    const replica = replicas[currentIndex];
    const nextIndex = (currentIndex + 1) % replicas.length; // Move to the next replica
    return { replica, nextIndex };
}
```

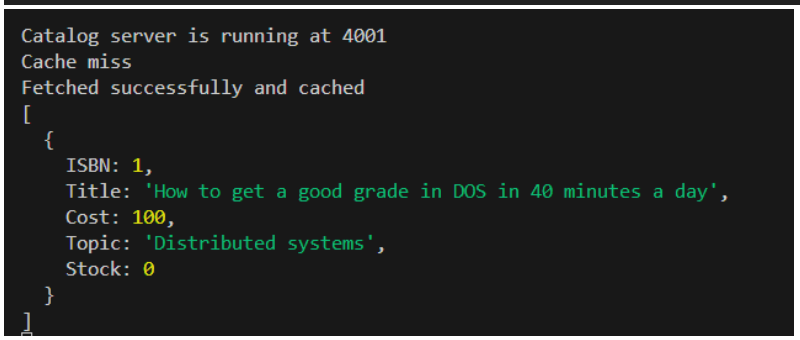


We have 2 replicas for the catalog/order serves.

- Who its work?
  So I will hit the same URL , and the frontend server must balancing the two request for the 2 catalog replicas “1 for each every time”:

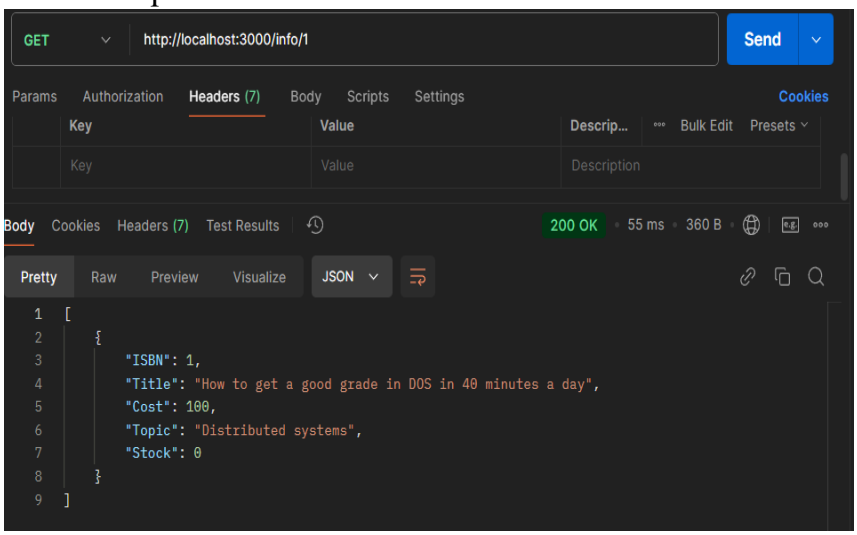
First request:





We notice that its in catalog2 server and without cache (it’s the first time)

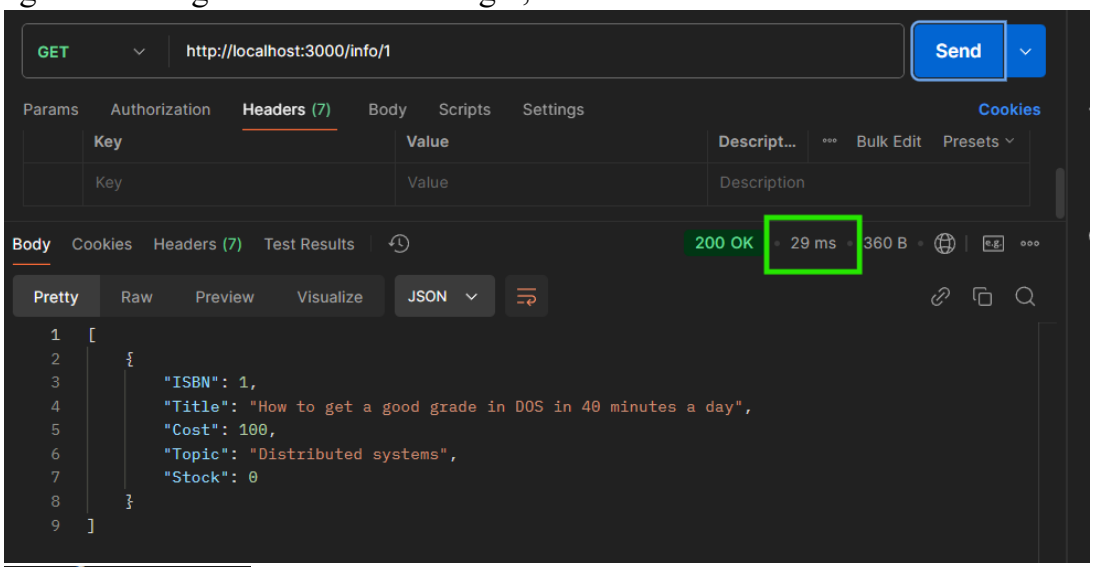
Second request:



```
Catalog server is running at 4000
Cache miss
Fetched successfully and cached (catalog1)
[
  {
    ISBN: 1,
    Title: 'How to get a good grade in DOS in 40 minutes a day',
    Cost: 100,
    Topic: 'Distributed systems',
    Stock: 0
  }
]
```

We notice that its in catalog server and without cache (it's the first time in this server)

- The frontend balance the request between these replicas , and if we hit this URI again its will go back to the catalog2 , but cached:




```
]
Cache hit
```

-With cache: 122/29 => 4.2 faster