# DOS Project Report Part 2

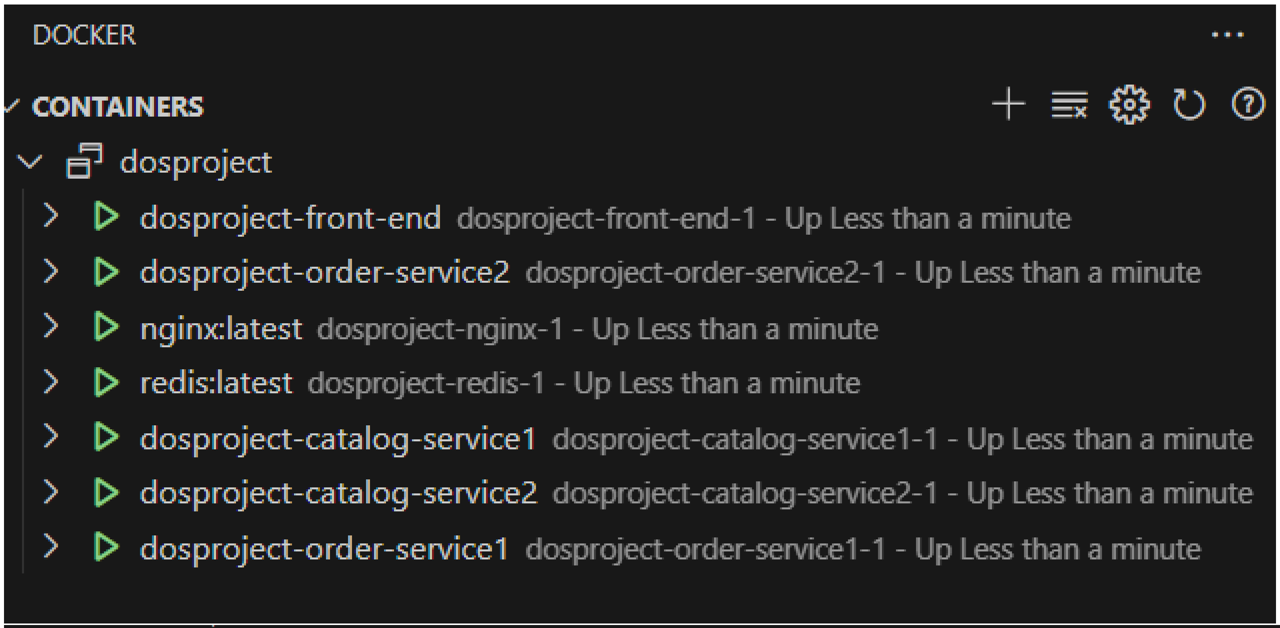
**Student Name: Osamah Abdullah  #12111983**

**Student Name: Shahd Hennwai     #12115159**

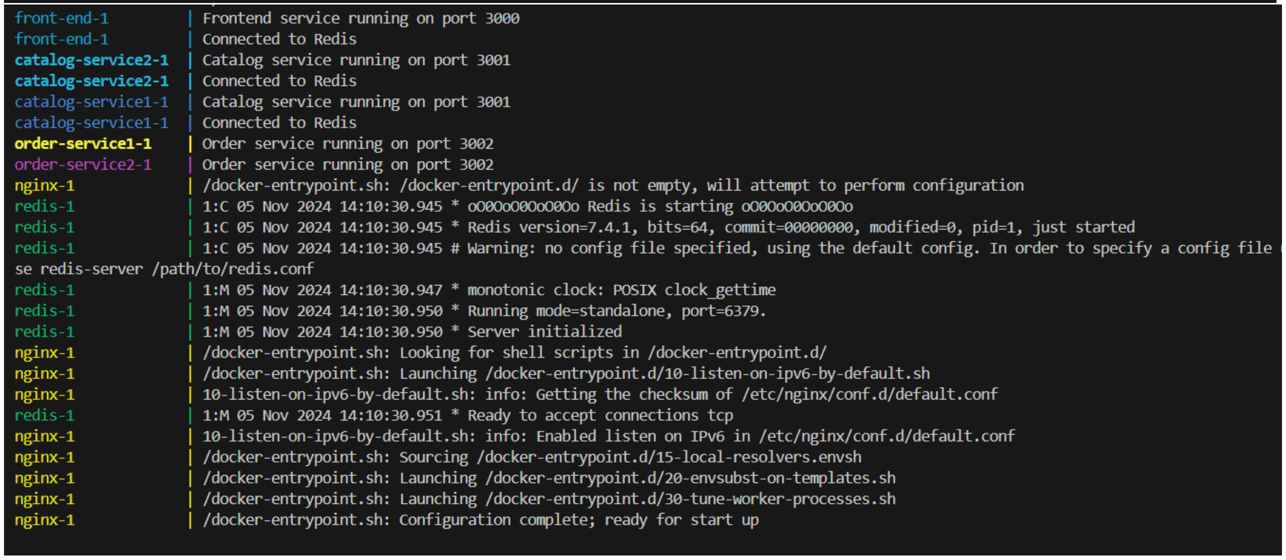
# Part1: Cache Consistency

**Redis Cache Implementation**

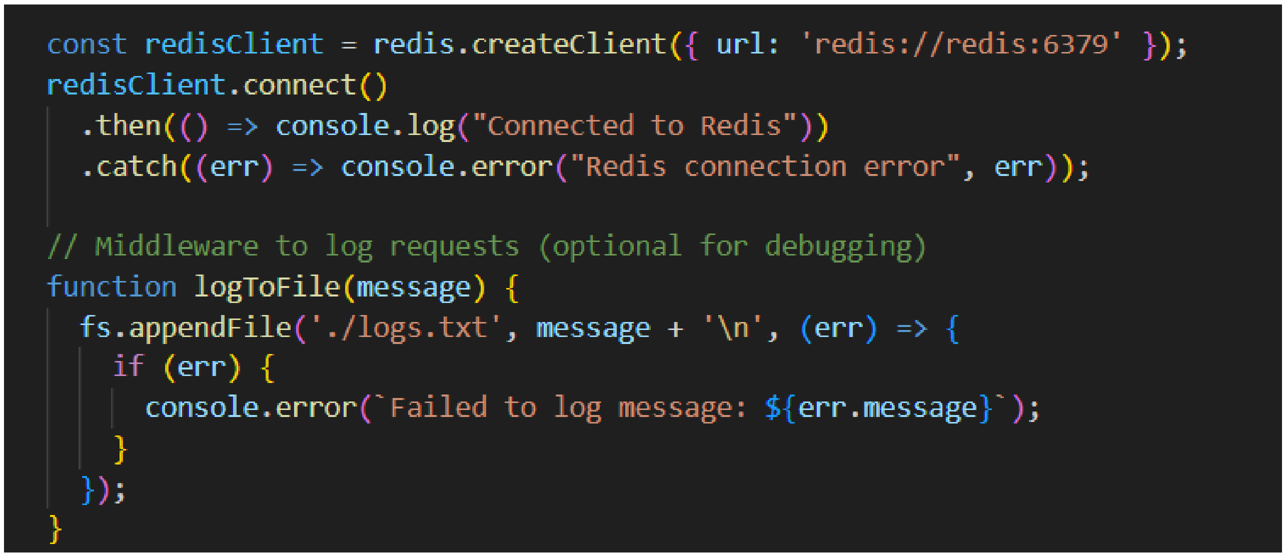
- **Description**: Integrated Redis to cache frequently accessed data, reducing load on the catalog service and improving response times.
- **Implementation**: Cached responses for specific routes (like /search/:topic and /info/:item_number) using Redis. Implemented caching with a TTL (time-to-live) of 1 hour for each entry.



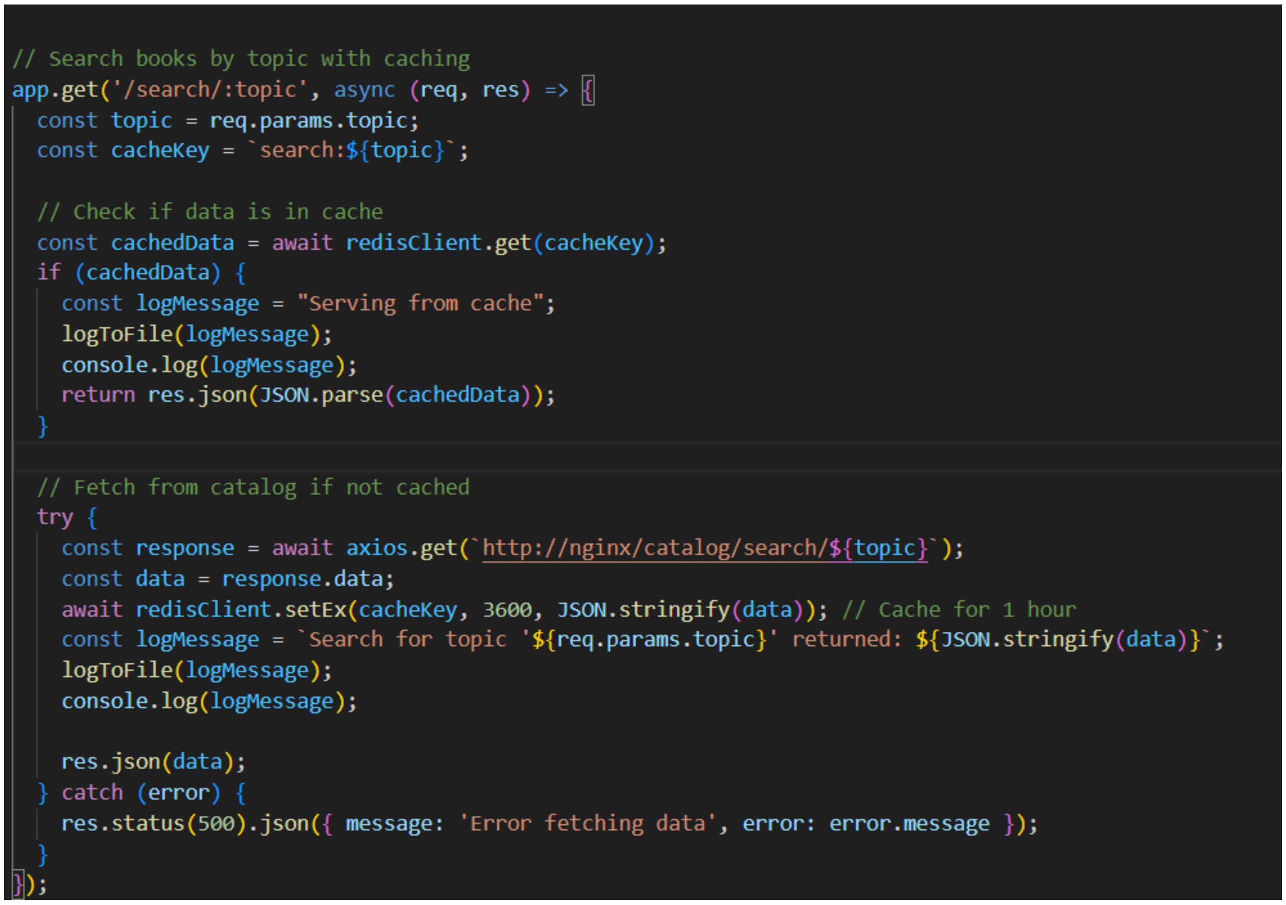
```
DOCKER
CONTAINERS
dosproject
  dosproject-front-end  dosproject-front-end-1 - Up Less than a minute
  dosproject-order-service2  dosproject-order-service2-1 - Up Less than a minute
  nginx:latest  dosproject-nginx-1 - Up Less than a minute
  redis:latest  dosproject-redis-1 - Up Less than a minute
  dosproject-catalog-service1  dosproject-catalog-service1-1 - Up Less than a minute
  dosproject-catalog-service2  dosproject-catalog-service2-1 - Up Less than a minute
  dosproject-order-service1  dosproject-order-service1-1 - Up Less than a minute
```



```
front-end-1           | Frontend service running on port 3000
front-end-1           | Connected to Redis
catalog-service2-1    | Catalog service running on port 3001
catalog-service2-1    | Connected to Redis
catalog-service1-1    | Catalog service running on port 3001
catalog-service1-1    | Connected to Redis
order-service1-1      | Order service running on port 3002
order-service2-1      | Order service running on port 3002
nginx-1               | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
redis-1               | 1:C 05 Nov 2024 14:10:30.945 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis-1               | 1:C 05 Nov 2024 14:10:30.945 * Redis version=7.4.1, bits=64, commit=00000000, modified=0, pid=1, just started
redis-1               | 1:C 05 Nov 2024 14:10:30.945 # Warning: no config file specified, using the default config. In order to specify a config file
se redis-server /path/to/redis.conf
redis-1               | 1:M 05 Nov 2024 14:10:30.947 * monotonic clock: POSIX clock_gettime
redis-1               | 1:M 05 Nov 2024 14:10:30.950 * Running mode=standalone, port=6379.
redis-1               | 1:M 05 Nov 2024 14:10:30.950 * Server initialized
nginx-1               | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx-1               | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx-1               | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
redis-1               | 1:M 05 Nov 2024 14:10:30.951 * Ready to accept connections tcp
nginx-1               | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
nginx-1               | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
nginx-1               | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx-1               | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
nginx-1               | /docker-entrypoint.sh: Configuration complete; ready for start up
```



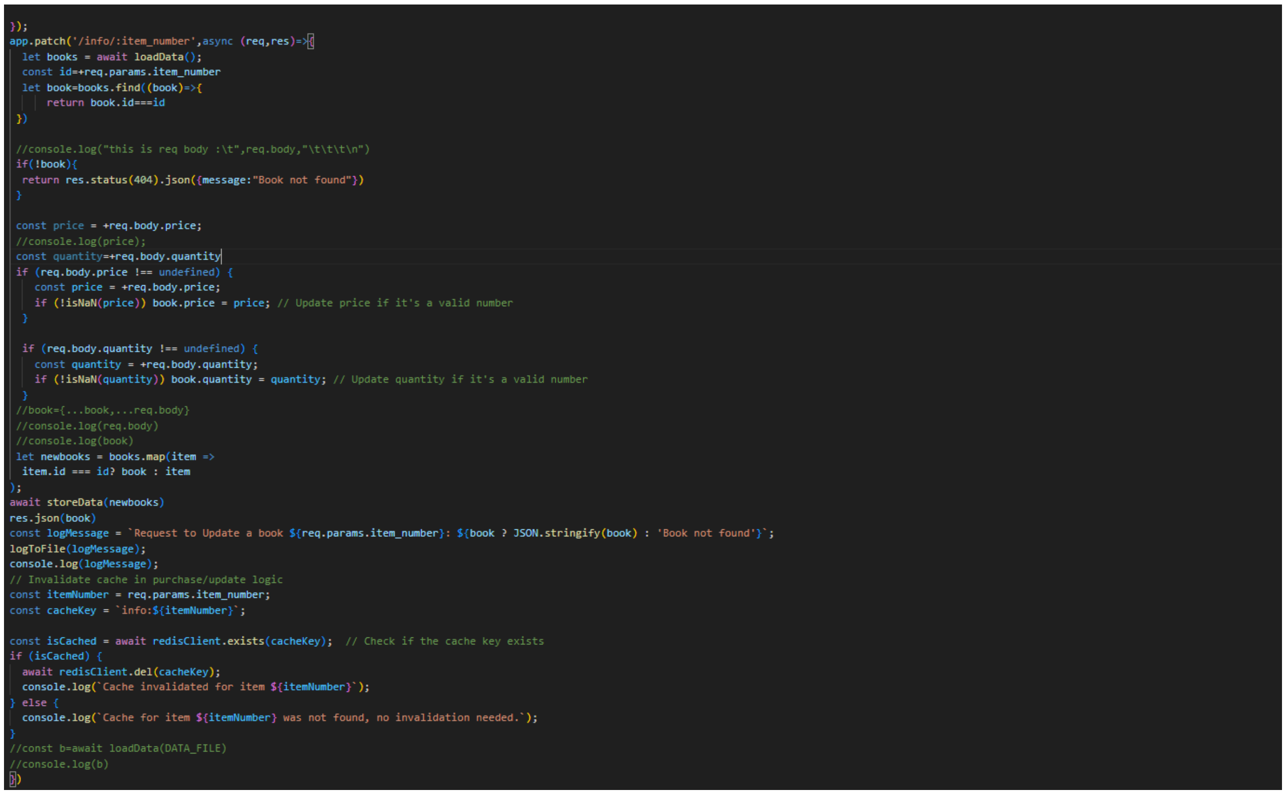
```
const redisClient = redis.createClient({ url: 'redis://redis:6379' });
redisClient.connect()
  .then(() => console.log("Connected to Redis"))
  .catch((err) => console.error("Redis connection error", err));

// Middleware to log requests (optional for debugging)
function logToFile(message) {
  fs.appendFile('./logs.txt', message + '\n', (err) => {
    if (err) {
      console.error(`Failed to log message: ${err.message}`);
    }
  });
}
```



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

  // Check if data is in cache
  const cachedData = await redisClient.get(cacheKey);
  if (cachedData) {
    const logMessage = "Serving from cache";
    logToFile(logMessage);
    console.log(logMessage);
    return res.json(JSON.parse(cachedData));
  }

  // Fetch from catalog if not cached
  try {
    const response = await axios.get(`http://nginx/catalog/search/${topic}`);
    const data = response.data;
    await redisClient.setEx(cacheKey, 3600, JSON.stringify(data)); // Cache for 1 hour
    const logMessage = `Search for topic '${req.params.topic}' returned: ${JSON.stringify(data)}`;
    logToFile(logMessage);
    console.log(logMessage);

    res.json(data);
  } catch (error) {
    res.status(500).json({ message: 'Error fetching data', error: error.message });
  }
});
```

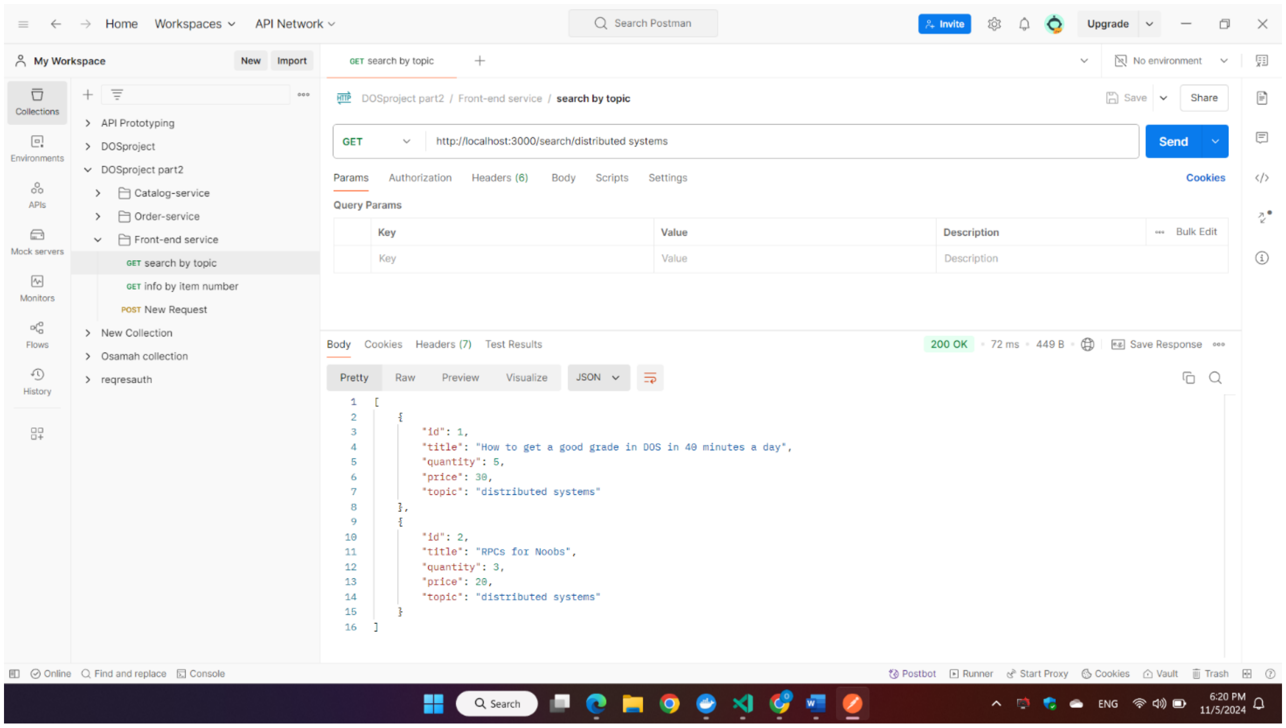
## Cache Invalidation Mechanism

- **Description**: Added a cache invalidation mechanism to keep cached data up-to-date when data changes occur.
- **Implementation**: Invalidated cache entries after a purchase or catalog update by deleting the relevant Redis cache keys.



```
});
app.patch('/info/:item_number',async (req,res)=>{
  let books = await loadData();
  const id=+req.params.item_number
  let book=books.find((book)=>{
      return book.id===id
 })

 //console.log("this is req body :\t",req.body,"\t\t\t\n")
 if(!book){
  return res.status(404).json({message:"Book not found"})
 }

 const price = +req.body.price;
 //console.log(price);
 const quantity=+req.body.quantity
 if (req.body.price !== undefined) {
    const price = +req.body.price;
    if (!isNaN(price)) book.price = price; // Update price if it's a valid number
  }

  if (req.body.quantity !== undefined) {
    const quantity = +req.body.quantity;
    if (!isNaN(quantity)) book.quantity = quantity; // Update quantity if it's a valid number
  }
 //book={...book,...req.body}
 //console.log(req.body)
 //console.log(book)
 let newbooks = books.map(item =>
  item.id === id? book : item
);
await storeData(newbooks)
res.json(book)
const logMessage = `Request to Update a book ${req.params.item_number}: ${book ? JSON.stringify(book) : 'Book not found'}`;
logToFile(logMessage);
console.log(logMessage);
// Invalidate cache in purchase/update logic
const itemNumber = req.params.item_number;
const cacheKey = `info:${itemNumber}`;

const isCached = await redisClient.exists(cacheKey);  // Check if the cache key exists
if (isCached) {
  await redisClient.del(cacheKey);
  console.log(`Cache invalidated for item ${itemNumber}`);
} else {
  console.log(`Cache for item ${itemNumber} was not found, no invalidation needed.`);
}
//const b=await loadData(DATA_FILE)
//console.log(b)
})
```

- When i send `GET` request the **first time**, **<u>before Caching the data</u>**:

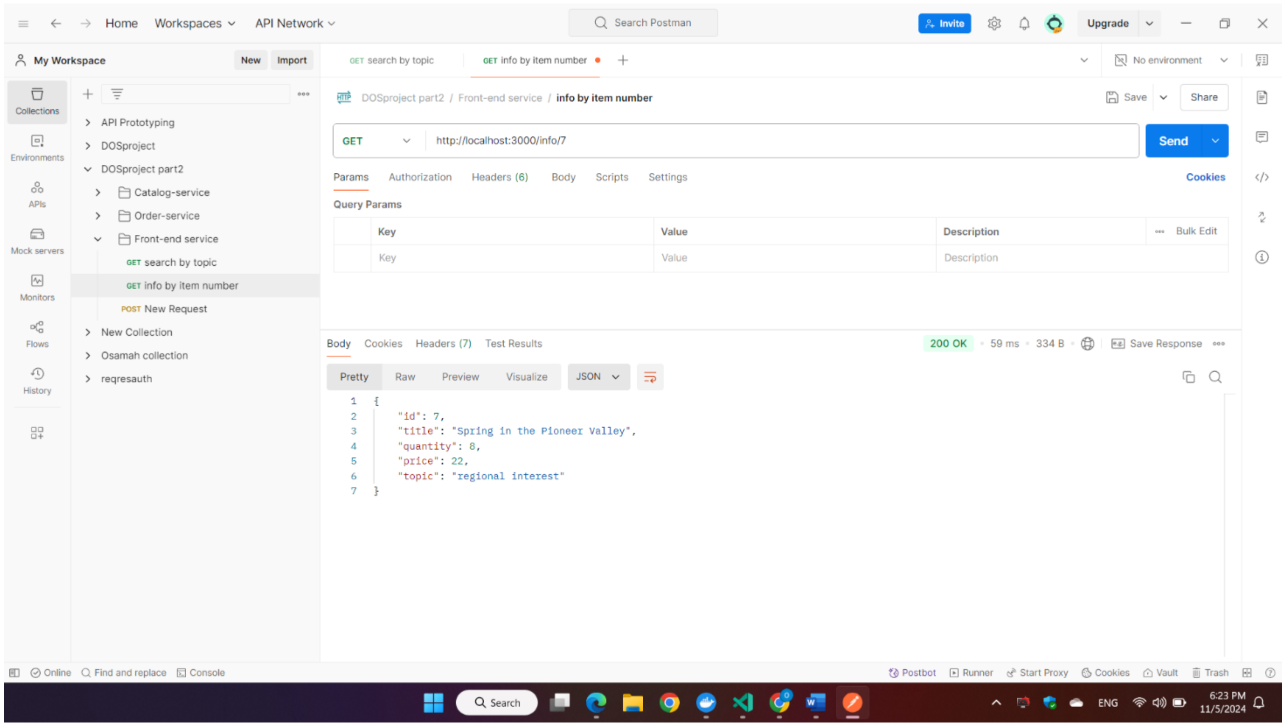


DOSproject part2 / Front-end service / search by topic

GET http://localhost:3000/search/distributed systems

200 OK · 72 ms · 449 B

```
[
    {
        "id": 1,
        "title": "How to get a good grade in DOS in 40 minutes a day",
        "quantity": 5,
        "price": 30,
        "topic": "distributed systems"
    },
    {
        "id": 2,
        "title": "RPCs for Noobs",
        "quantity": 3,
        "price": 20,
        "topic": "distributed systems"
    }
]
```



DOSproject part2 / Front-end service / info by item number

GET http://localhost:3000/info/7

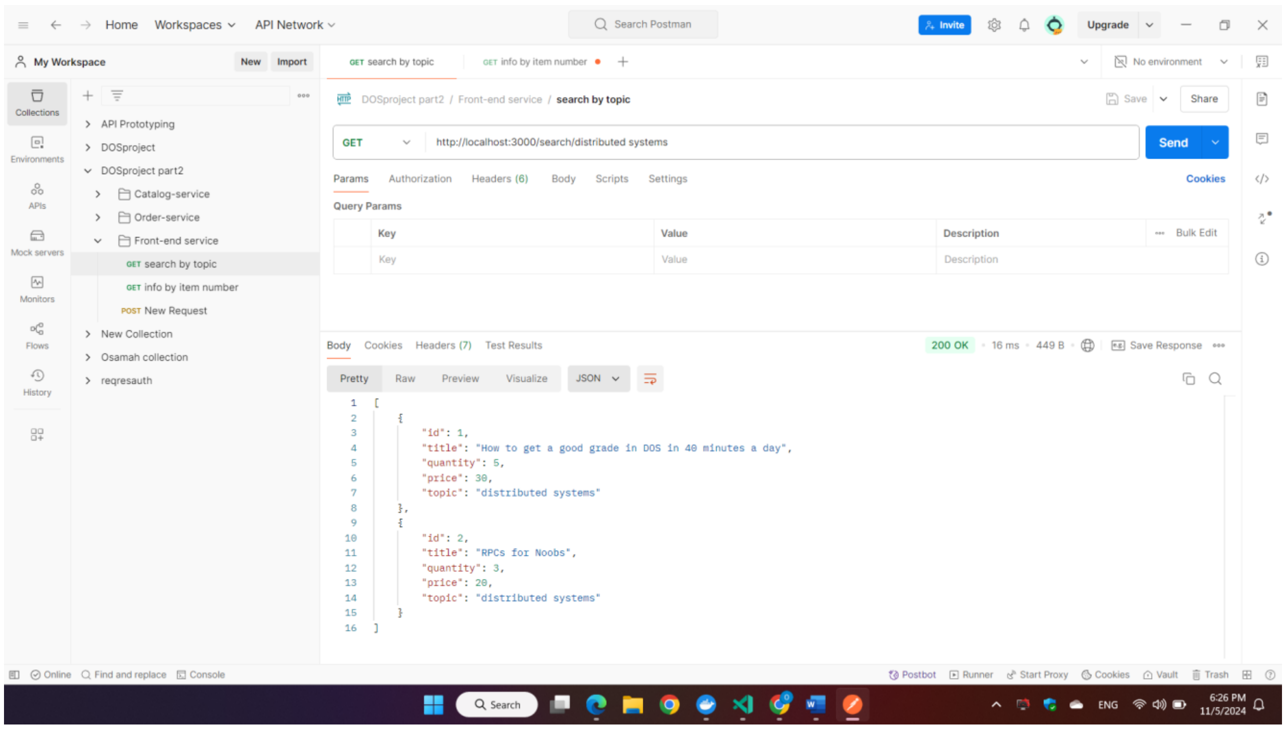
200 OK · 59 ms · 334 B

```
{
    "id": 7,
    "title": "Spring in the Pioneer Valley",
    "quantity": 8,
    "price": 22,
    "topic": "regional interest"
}
```

Q1) Compute the average response time (query/buy) of your new systems. What is the response time with and without caching?

- Answers
  - for info: **59ms**
  - for search: 72**ms**

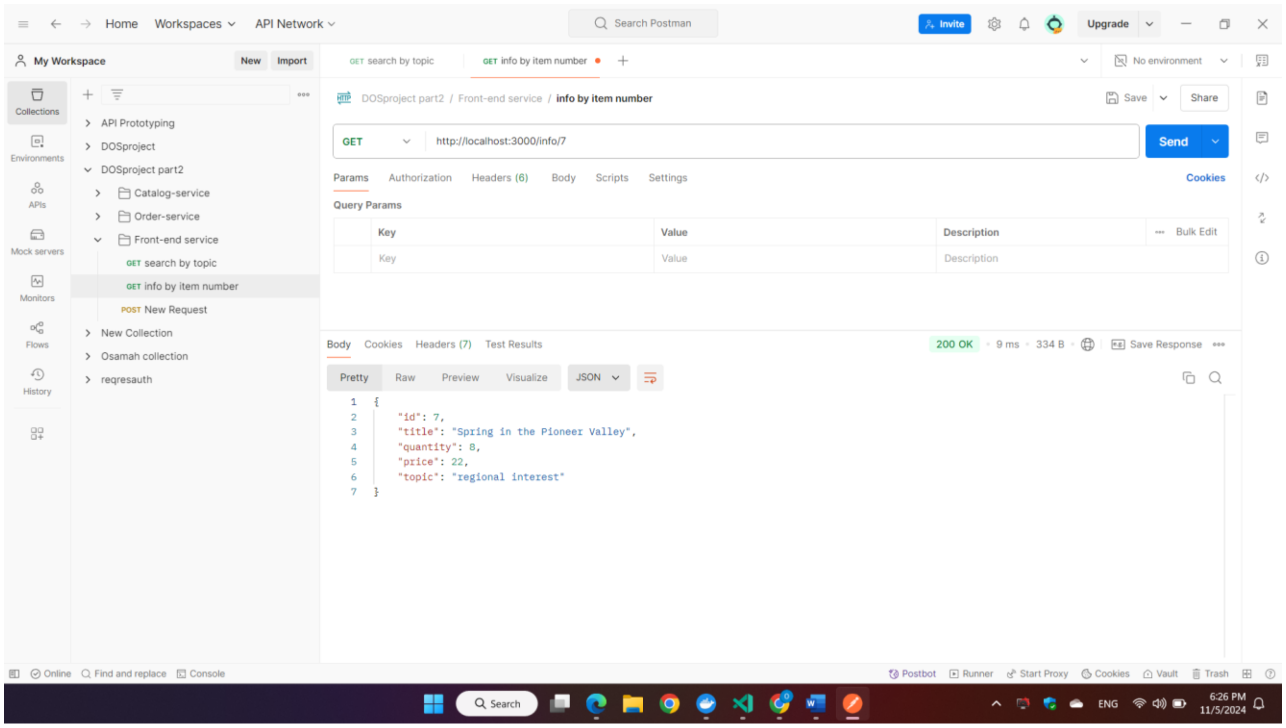
- With Cache:


GET http://localhost:3000/search/distributed systems

DOSproject part2 / Front-end service / search by topic

200 OK · 16 ms · 449 B

```
[
    {
        "id": 1,
        "title": "How to get a good grade in DOS in 40 minutes a day",
        "quantity": 5,
        "price": 30,
        "topic": "distributed systems"
    },
    {
        "id": 2,
        "title": "RPCs for Noobs",
        "quantity": 3,
        "price": 20,
        "topic": "distributed systems"
    }
]
```



```
GET http://localhost:3000/info/7
200 OK  9 ms  334 B
```

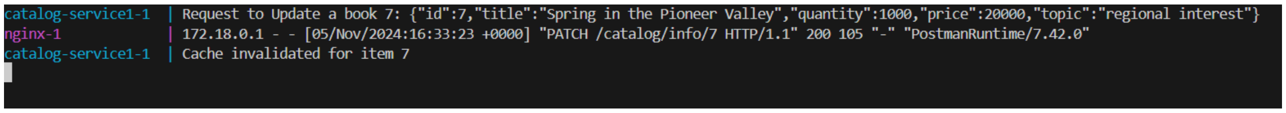
```
{
    "id": 7,
    "title": "Spring in the Pioneer Valley",
    "quantity": 8,
    "price": 22,
    "topic": "regional interest"
}
```

Q2) How much does caching help?

- Answers
  - for info: **9ms ,   59/9 —> 6.55 Faster than without cache**
  - for search: **16ms,   72/16 —> 4.5 Faaster than without using cache**

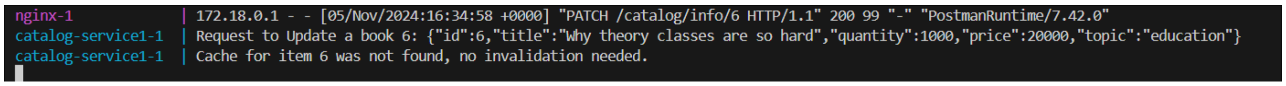
## <u>Invalidate Message</u>

### When the key is in cache really:



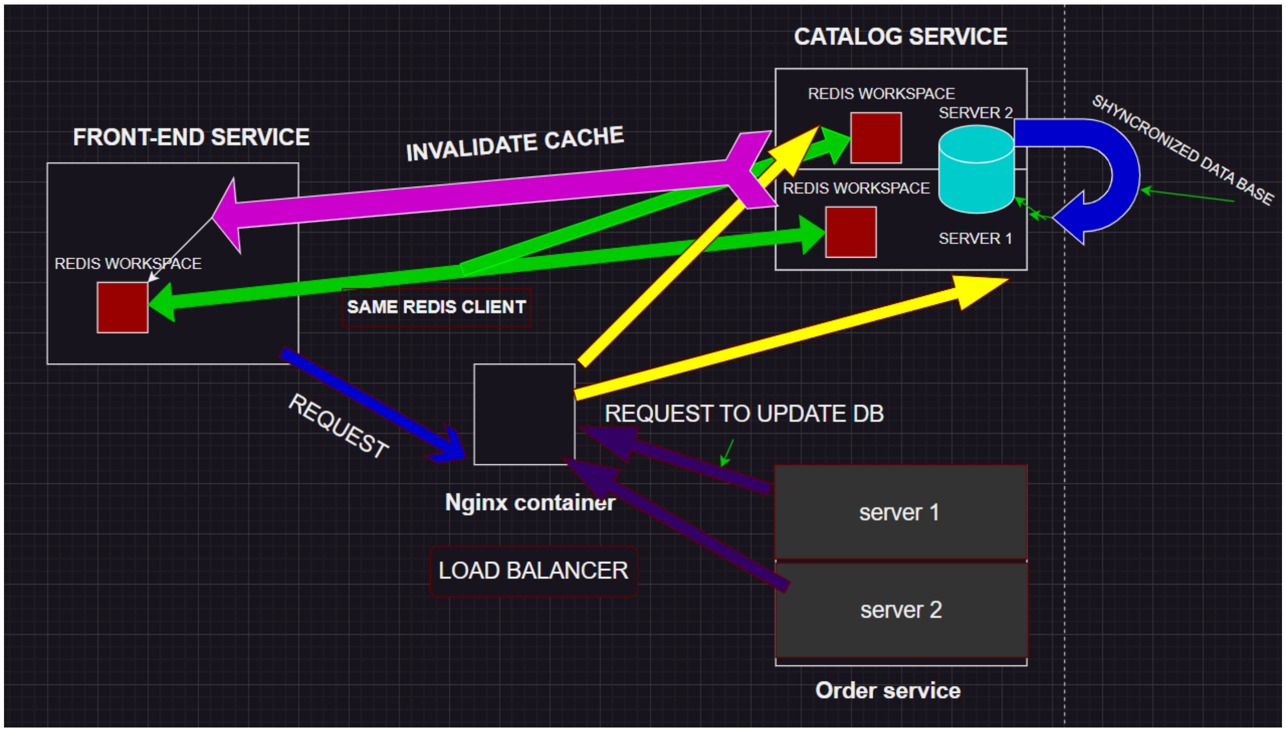
```
catalog-service1-1  | Request to Update a book 7: {"id":7,"title":"Spring in the Pioneer Valley","quantity":1000,"price":20000,"topic":"regional interest"}
nginx-1             | 172.18.0.1 - - [05/Nov/2024:16:33:23 +0000] "PATCH /catalog/info/7 HTTP/1.1" 200 105 "-" "PostmanRuntime/7.42.0"
catalog-service1-1  | Cache invalidated for item 7
```

### When it  aren't in the cache :



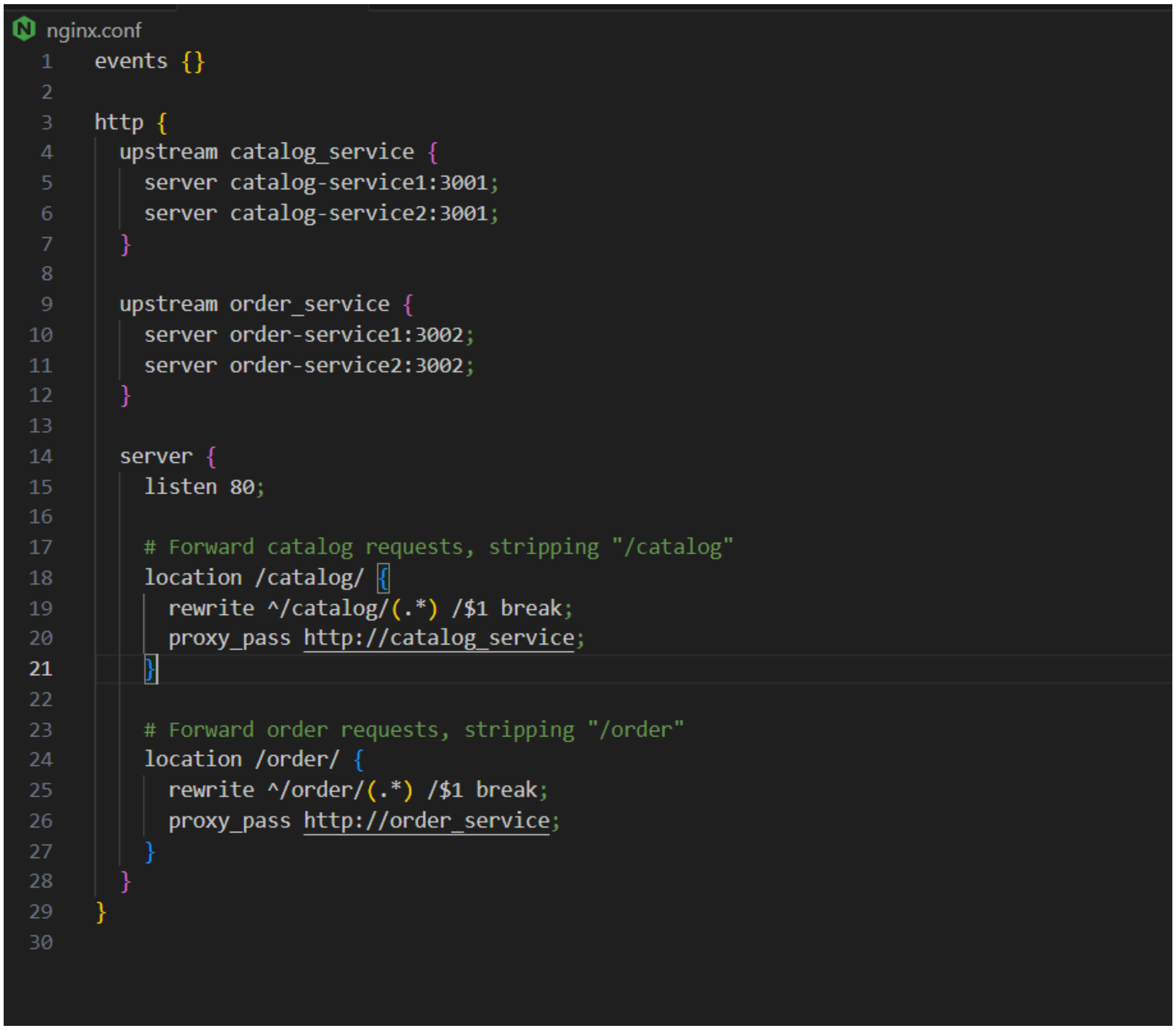
```
nginx-1             | 172.18.0.1 - - [05/Nov/2024:16:34:58 +0000] "PATCH /catalog/info/6 HTTP/1.1" 200 99 "-" "PostmanRuntime/7.42.0"
catalog-service1-1  | Request to Update a book 6: {"id":6,"title":"Why theory classes are so hard","quantity":1000,"price":20000,"topic":"education"}
catalog-service1-1  | Cache for item 6 was not found, no invalidation needed.
```

## System Hierarchy:



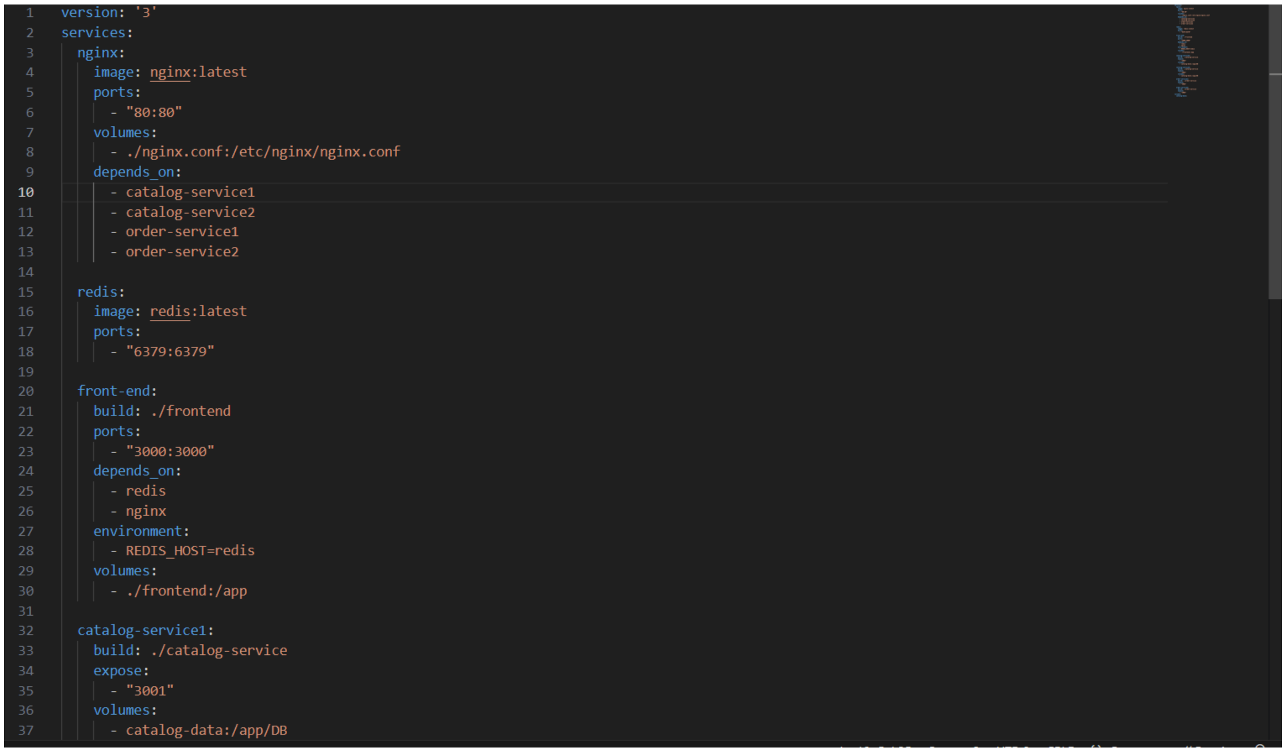
## Part2: Loadbalance with NGINX

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

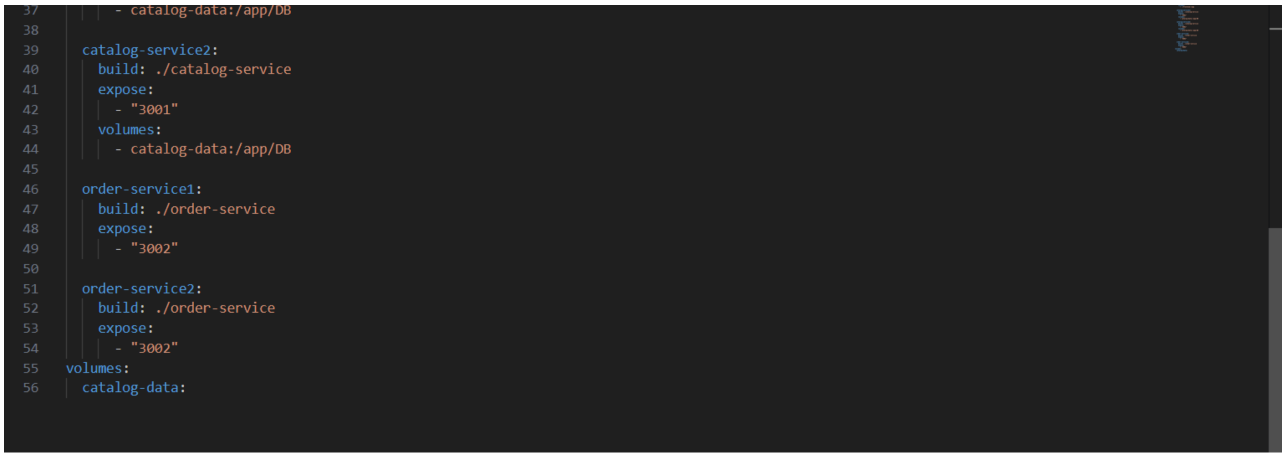

nginx.conf

```
events {}

http {
  upstream catalog_service {
    server catalog-service1:3001;
    server catalog-service2:3001;
  }

  upstream order_service {
    server order-service1:3002;
    server order-service2:3002;
  }

  server {
    listen 80;

    # Forward catalog requests, stripping "/catalog"
    location /catalog/ {
      rewrite ^/catalog/(.*) /$1 break;
      proxy_pass http://catalog_service;
    }

    # Forward order requests, stripping "/order"
    location /order/ {
      rewrite ^/order/(.*) /$1 break;
      proxy_pass http://order_service;
    }
  }
}
```

## **Part3: Dockerize your Application (Optional part)**

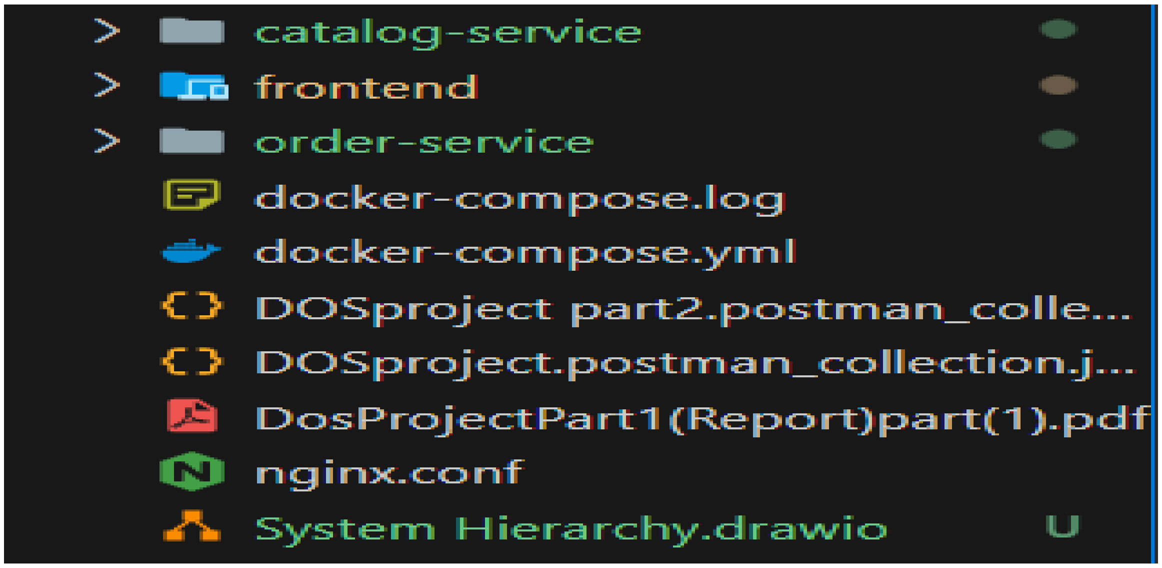


```yaml
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - catalog-service1
      - catalog-service2
      - order-service1
      - order-service2

  redis:
    image: redis:latest
    ports:
      - "6379:6379"

  front-end:
    build: ./frontend
    ports:
      - "3000:3000"
    depends_on:
      - redis
      - nginx
    environment:
      - REDIS_HOST=redis
    volumes:
      - ./frontend:/app

  catalog-service1:
    build: ./catalog-service
    expose:
      - "3001"
    volumes:
      - catalog-data:/app/DB
```



- catalog-data:/app/DB

```yaml

  catalog-service2:
    build: ./catalog-service
    expose:
      - "3001"
    volumes:
      - catalog-data:/app/DB

  order-service1:
    build: ./order-service
    expose:
      - "3002"

  order-service2:
    build: ./order-service
    expose:
      - "3002"
volumes:
  catalog-data:
```



```
> catalog-service
> frontend
> order-service
  docker-compose.log
  docker-compose.yml
  DOSproject part2.postman_colle...
  DOSproject.postman_collection.j...
  DosProjectPart1(Report)part(1).pdf
  nginx.conf
  System Hierarchy.drawio          U
```