Bazar.com

A Multi-tier Online Book Store

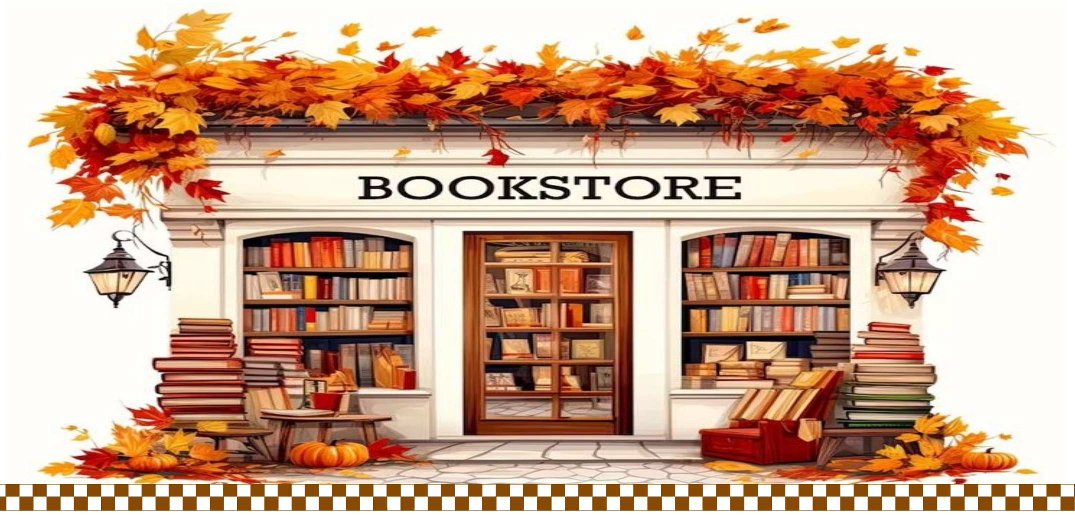
SUBMITTED TO: Dr.Samer Arandi

Reem AbdAl Raheem Hasan << 12112527>>

Farah Faisal Ahmad << 12112820>>

Github link:

https://github.com/ReemHasan31/my-Book-app



✓ Introduction

The main objective of this project is to design and implement a simple distributed e-commerce system called **Bazar.com** using the **microservices architecture pattern**.

It consists of several backend services written in **Java (Ninja Framework)** and a **frontend CLIENT interface** developed using **Node.js**.

All services are containerized and orchestrated using **Docker Compose** for easy deployment and testing.

✓ System Architecture

The Bazar.com platform is designed using three main microservices, each responsible for a specific domain.

1. Front-end Service (Client Application)

- Acts as the user interface of the system.
- Developed using Node.js and executed inside a terminal-based interface (CLI).
- Communicates directly with the backend services (Catalog and Order) through HTTP requests.

Main Operations:

- search(topic) → Sends a request to the Catalog Service to retrieve all books related to the specified topic.
- info(item_number) → Requests detailed information about a specific book (title, price, topic, and stock).
- purchase(item_number) → Initiates the purchase process by sending a request to the Order Service, which validates and updates the stock.

2. Catalog Service

- Responsible for managing and providing access to the book inventory.
- Developed using Node.js with the Express.js framework.
- Maintains data such as the book title, item number, topic, price, and available stock.
- Supports two primary operations that can be accessed via REST endpoints.

Supported Operations:

- query (by topic or item number) → Returns all available books based on a given topic, or the details of a specific item when queried by its ID.
- update → Modifies stock levels or adjusts the price of a book after a successful purchase.

  Additional Features:
- Data is persisted in a local file (catalog.csv) .

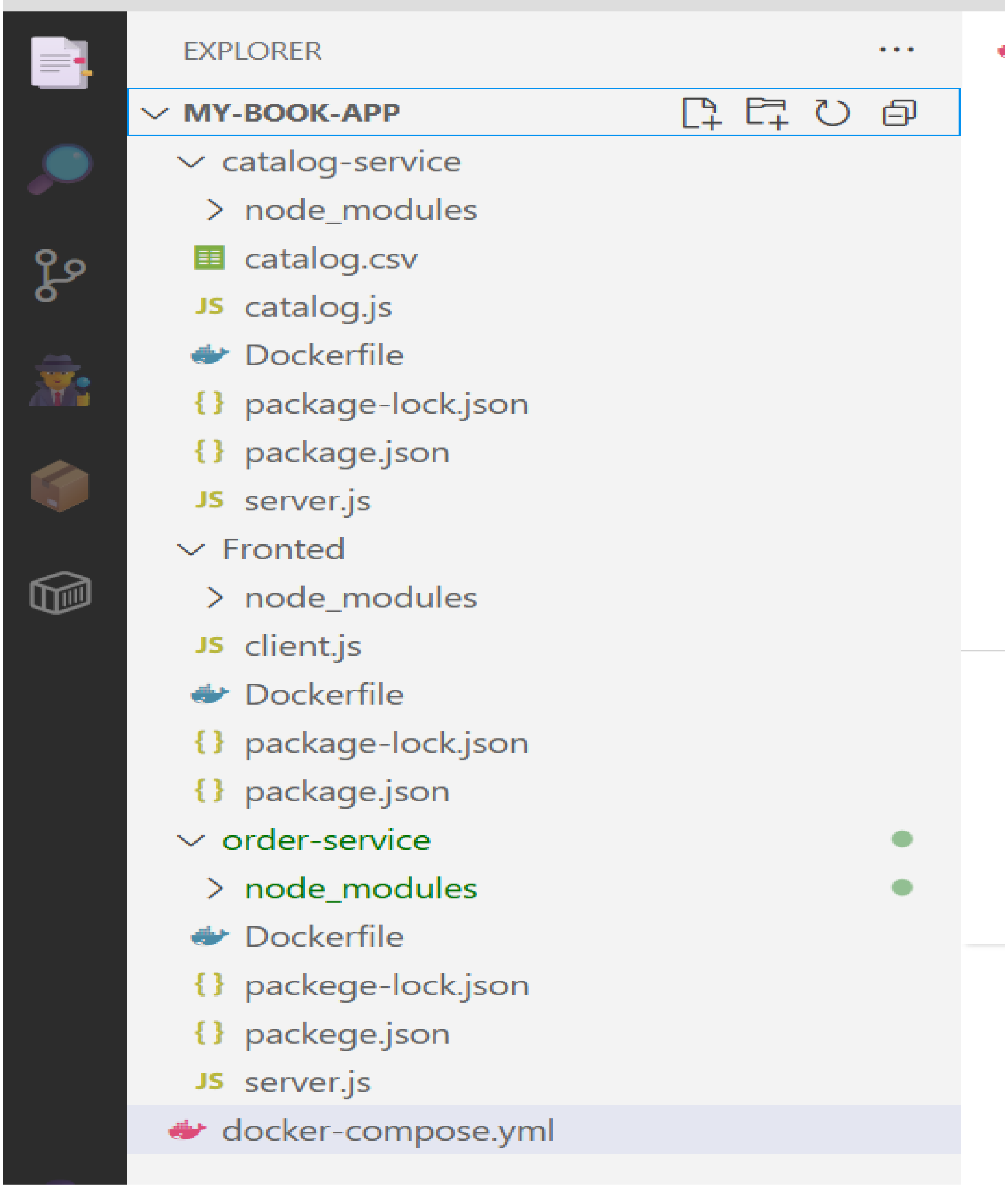
3. Order Service

- Handles all purchase transactions within the system.
- Developed using Node.js with the Express.js framework and exposes REST endpoints for the frontend to communicate with.
- Interacts with the Catalog Service to validate stock availability before finalizing any purchase.
- Upon a successful transaction, it updates the catalog to reflect the reduced inventory.

  Supported Operation:
- purchase(item_number) →
  - Verifies whether the requested book is still available in stock.
  - Communicates with the Catalog Service to confirm and update the stock quantity.
  - Returns a success message if the purchase is completed, or an error if the item is out of stock.

✓System Design and Folder Structure

The system follows a modular structure, where each service is completely independent and containerized.



EXPLORER

```
MY-BOOK-APP
  catalog-service
    node_modules
    catalog.csv
    catalog.js
    Dockerfile
    package-lock.json
    package.json
    server.js
  Fronted
    node_modules
    client.js
    Dockerfile
    package-lock.json
    package.json
  order-service
    node_modules
    Dockerfile
    packege-lock.json
    packege.json
    server.js
  docker-compose.yml
```

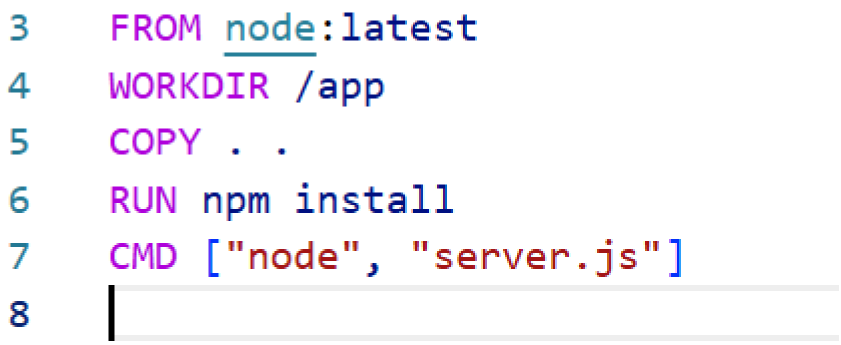
✓ How the project works

- Git clone https: https://github.com/ReemHasan31/my-Book-app
- docker-compose up --build
- Inside the Desktop docker for the frontend service container, open the exec tab and run the command (node client.js).
- Search and book info results are stored locally in memory to reduce repeated API calls.
- When a purchase is made, the order service verifies stock from the catalog, reduces the quantity, and confirms the transaction.

✓ Work and Results

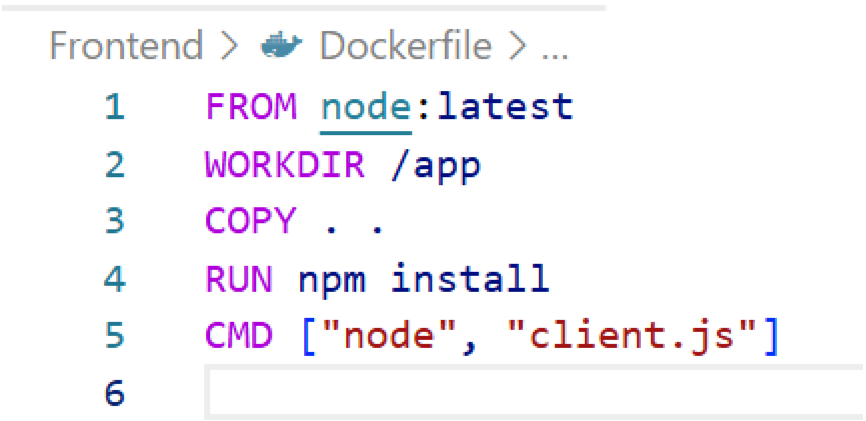
We started by writing individual Dockerfiles for every service

- ➢ catalog service



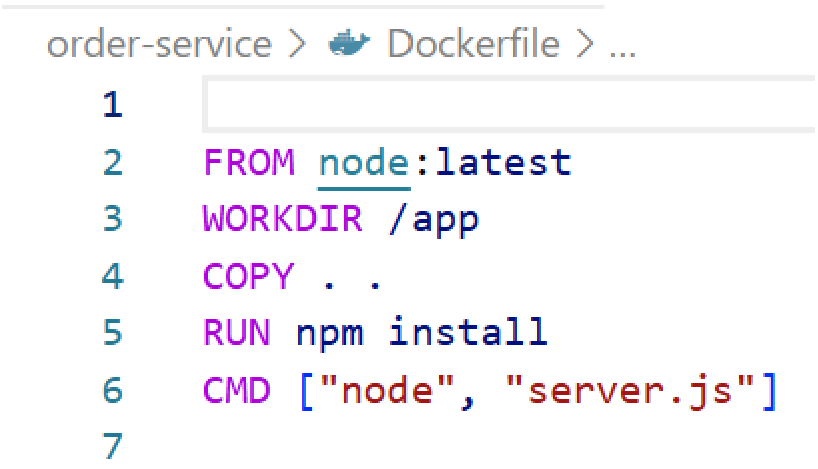
```
FROM node:latest
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "server.js"]
```

- **Frontend service**



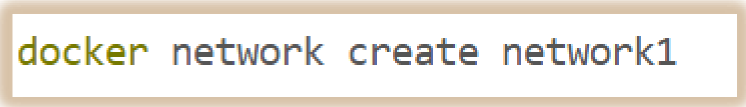
```
FROM node:latest
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "client.js"]

```

- **Order service**



```

FROM node:latest
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "server.js"]

```

Next, we had to build the Docker images for each service
And It is necessary to set up a network to allow communication between the containers.



```
docker network create network1
```

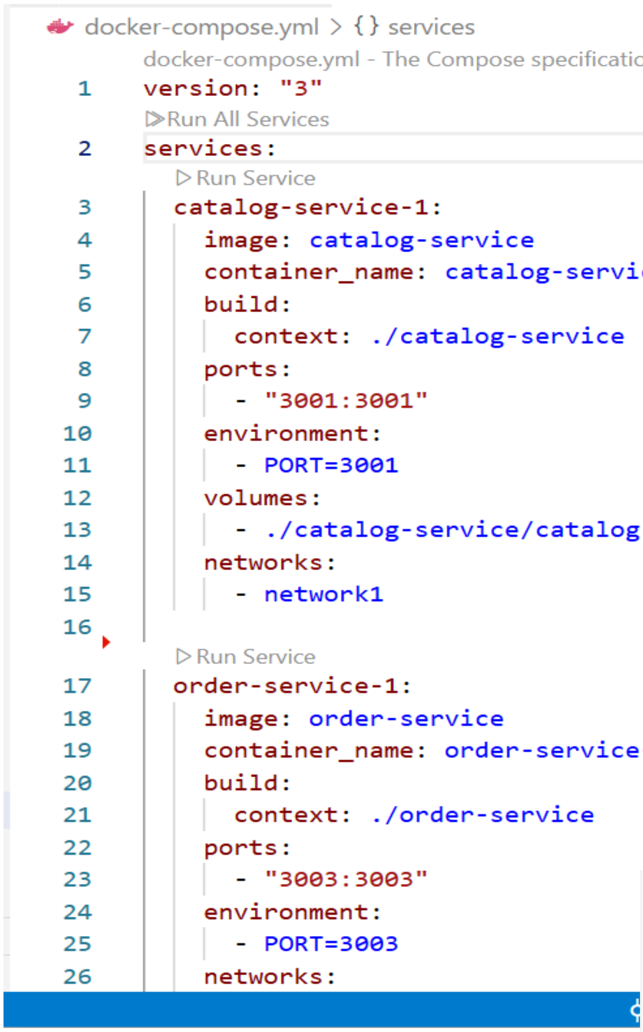
Now we will run the Docker containers for our services. First, we created a Docker network named network1 to allow the containers to communicate.

The catalog-service container runs on port 3001, with the catalog.csv file from the host mounted inside the container. This service provides the book catalog.

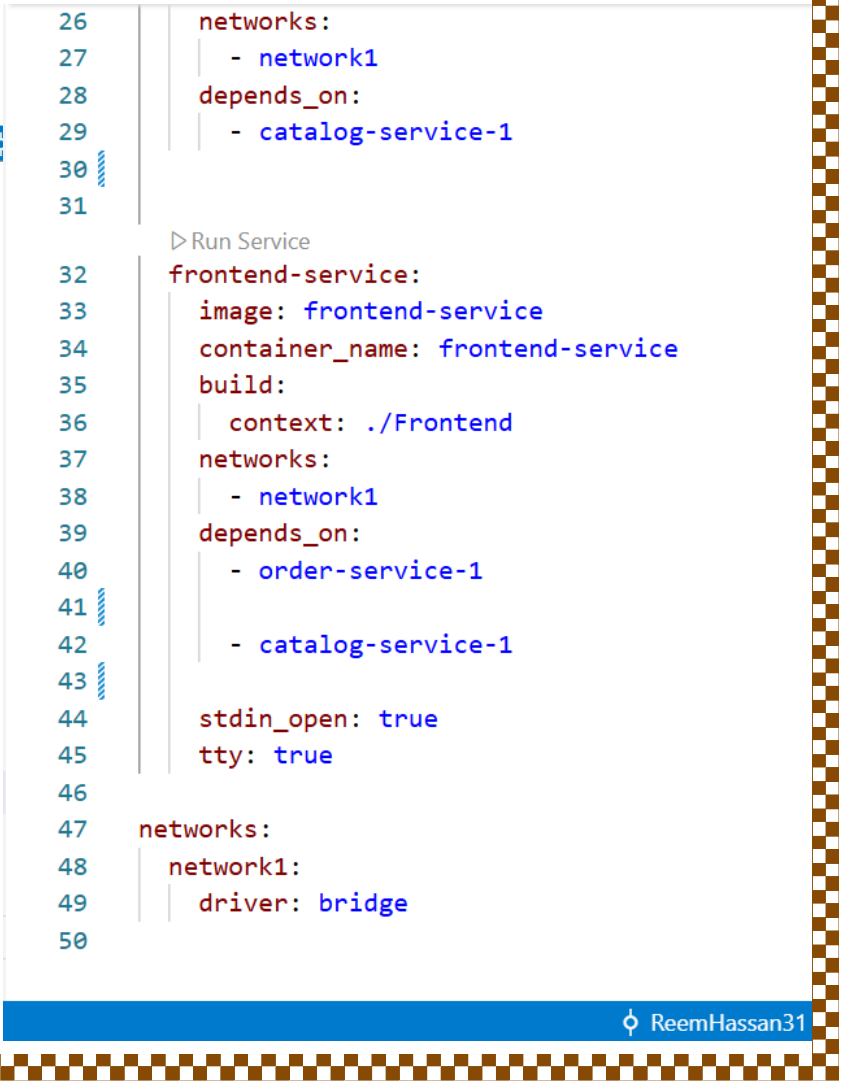
The order-service container runs on port 3003 and depends on the catalog service to process book purchases.

The frontend-service container is run with interactive terminal support, allowing the user to interact with the CLI application. All containers are connected to network1, enabling communication between the services and creating a functional microservices setup.

However, instead of running each container manually with separate docker run commands, we can use Docker Compose. Docker Compose allows us to define all services, networks, and volumes in a **single docker-compose.yml file** and then launch them together with a single command. This simplifies deployment, ensures the services start in the correct order, and makes managing the multi-container microservices architecture much easier.

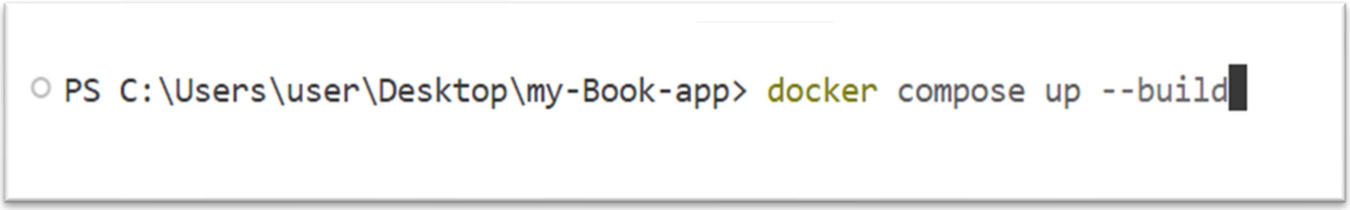

docker-compose.yml > { } services
docker-compose.yml - The Compose specification establishes a standard for the
▷Run All Services
▷ Run Service
▷ Run Service

```yaml
version: "3"
services:
  catalog-service-1:
    image: catalog-service
    container_name: catalog-service-1
    build:
      context: ./catalog-service
    ports:
      - "3001:3001"
    environment:
      - PORT=3001
    volumes:
      - ./catalog-service/catalog.csv:/app/catalog.csv
    networks:
      - network1

  order-service-1:
    image: order-service
    container_name: order-service-1
    build:
      context: ./order-service
    ports:
      - "3003:3003"
    environment:
      - PORT=3003
    networks:
```

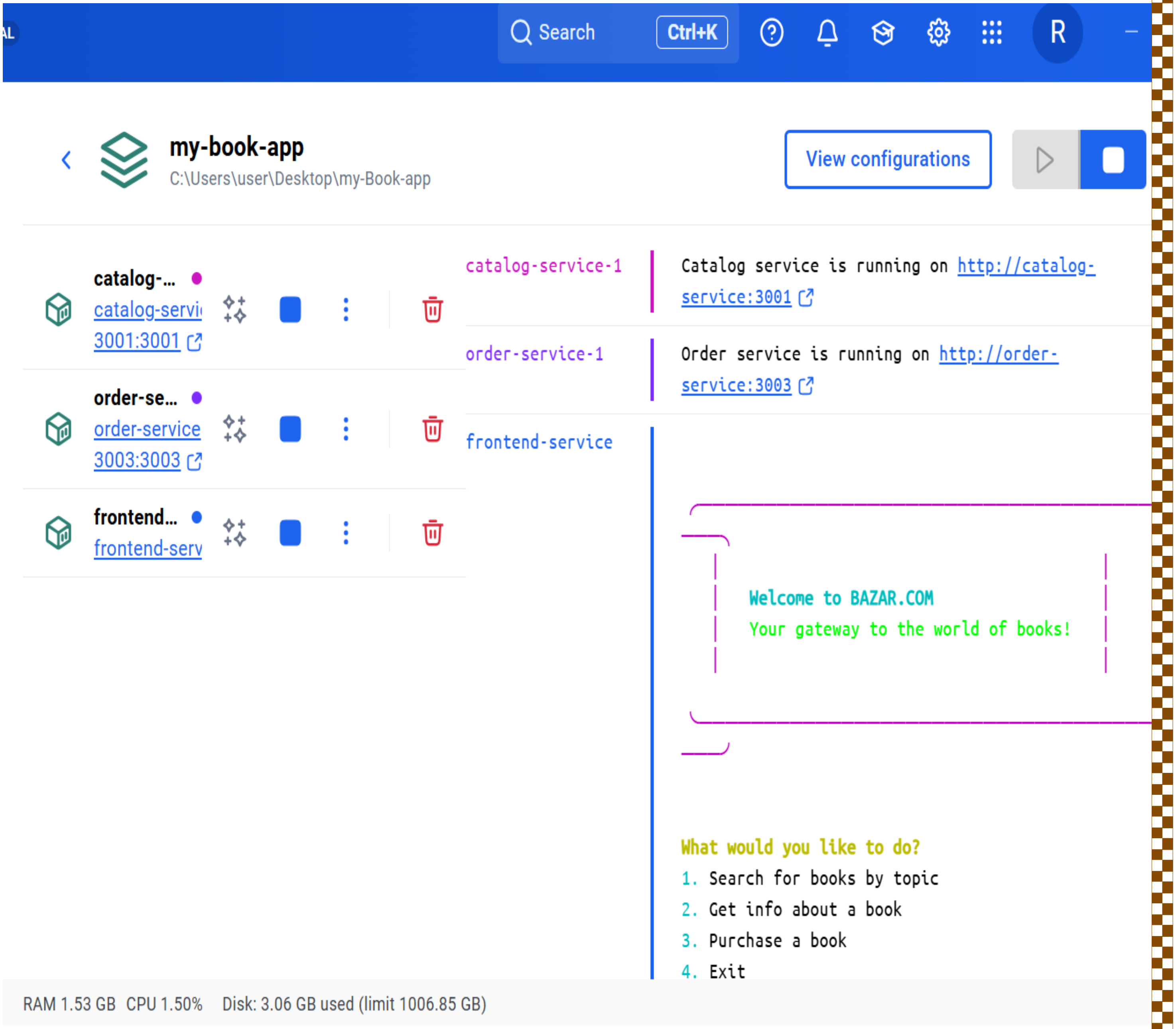


```yaml
    networks:
      - network1
    depends_on:
      - catalog-service-1


  frontend-service:
    image: frontend-service
    container_name: frontend-service
    build:
      context: ./Frontend
    networks:
      - network1
    depends_on:
      - order-service-1

      - catalog-service-1

    stdin_open: true
    tty: true

networks:
  network1:
    driver: bridge

```

Now we will run the docker compose



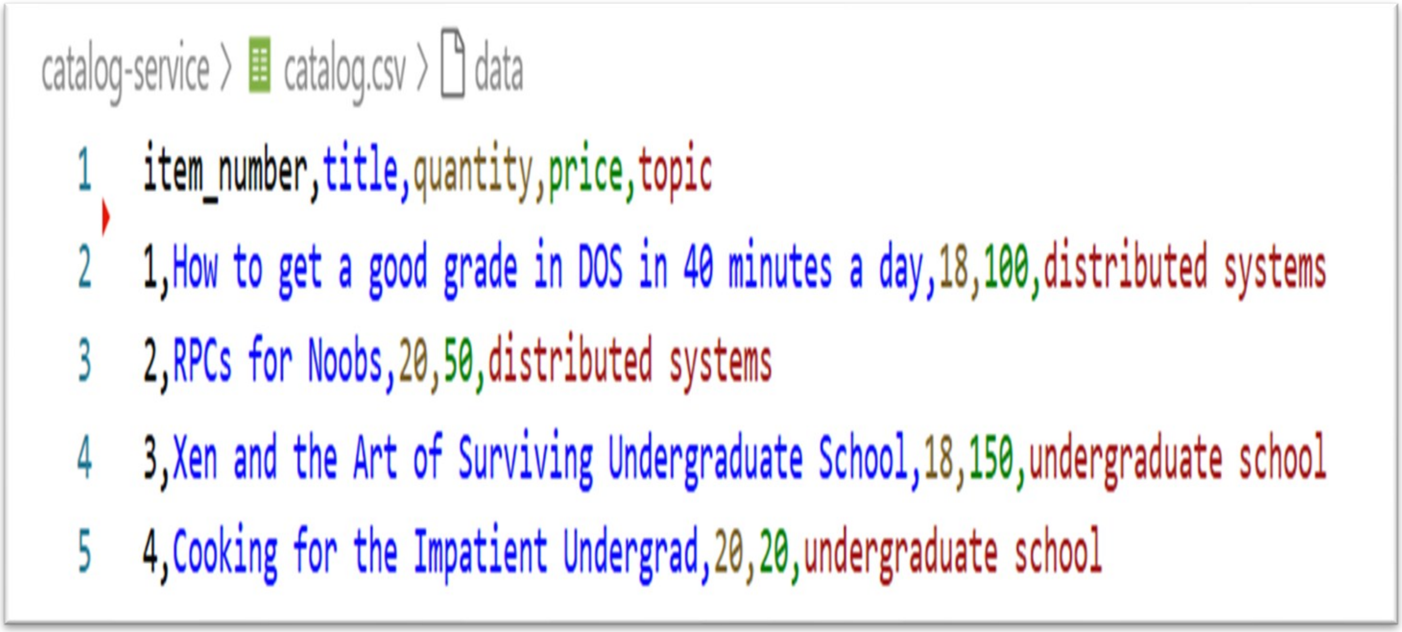
```
PS C:\Users\user\Desktop\my-Book-app> docker compose up --build
```



```
catalog-service-1  | Catalog service is running on http://catalog-service:3001
order-service-1    | Order service is running on http://order-service:3003
frontend-service   |
                   |  ┌──────────────────────────────────
                   | ─┐
                   |   |                                    |
                   |   |    Welcome to BAZAR.COM            |
                   |   |    Your gateway to the world of books!  |
                   |   |                                    |
                   |  └──────────────────────────────────
                   | ─┘
                   |
                   | What would you like to do?
                   | 1. Search for books by topic
                   | 2. Get info about a book
                   | 3. Purchase a book
                   | 4. Exit
```

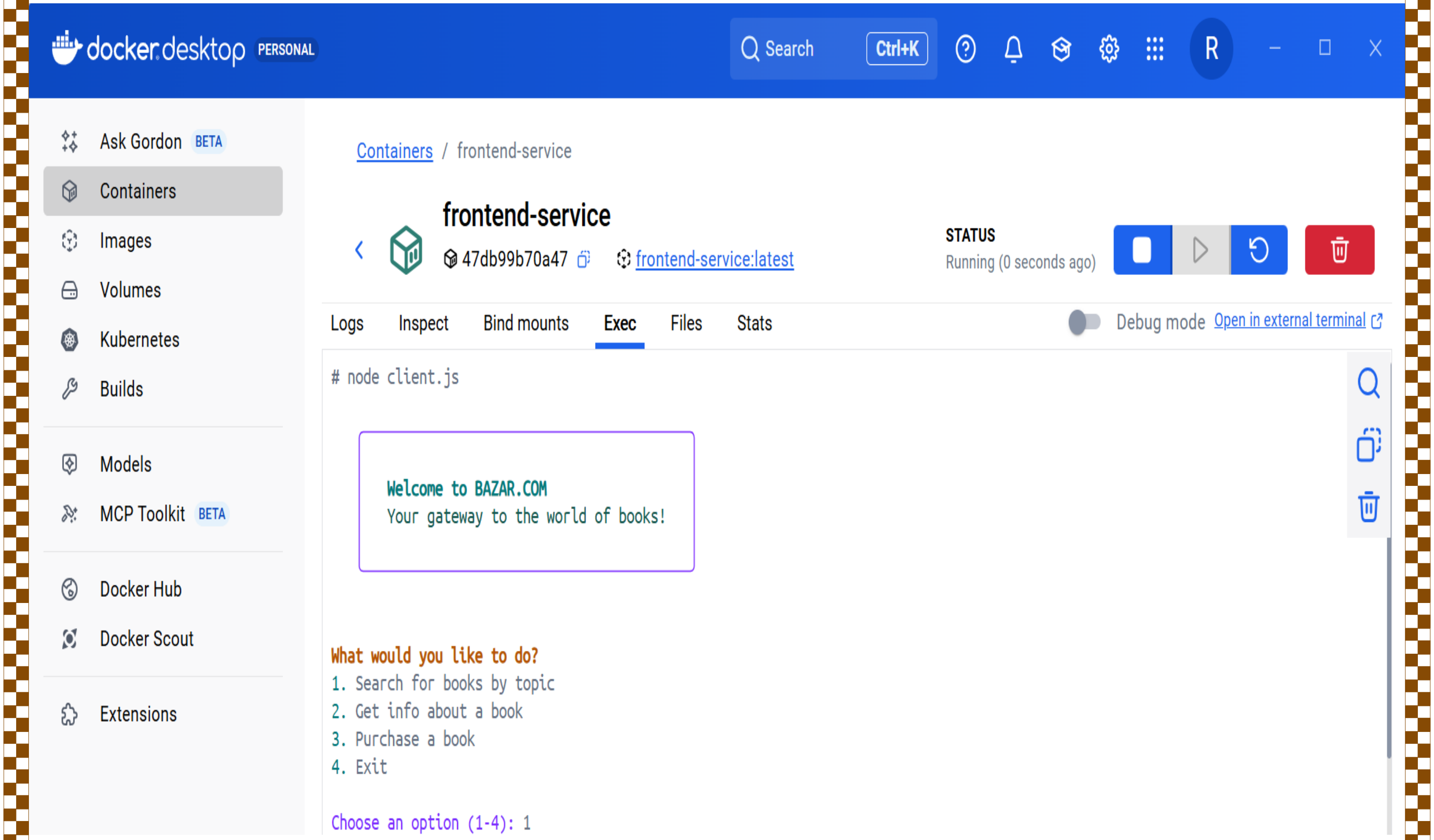
RAM 1.53 GB CPU 1.50% Disk: 3.06 GB used (limit 1006.85 GB)

Now, the frontend acts as a CLI, sending user requests to the catalog and order services. And the catalog service data in project as shown below:

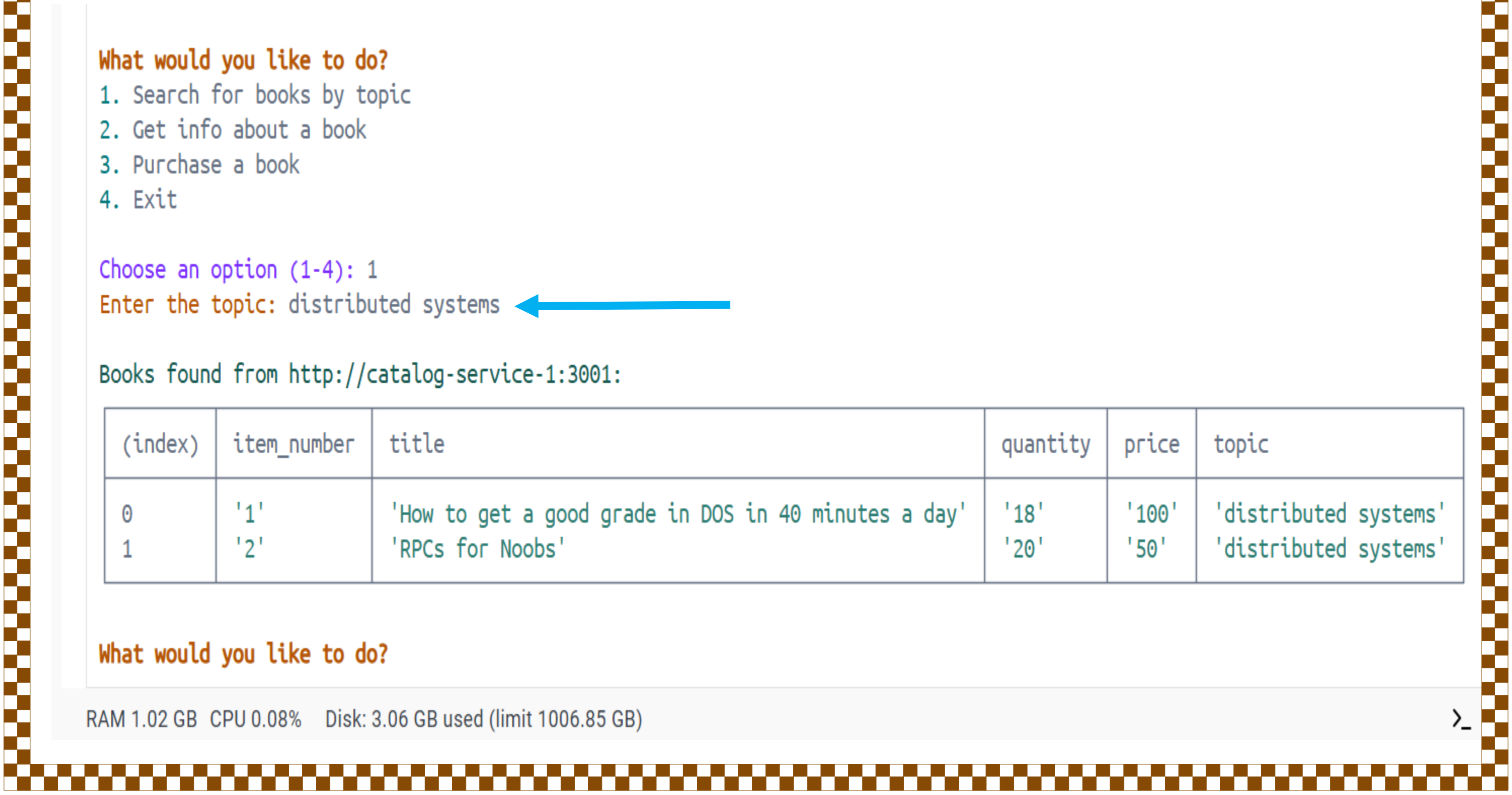


```
item_number,title,quantity,price,topic
1,How to get a good grade in DOS in 40 minutes a day,18,100,distributed systems
2,RPCs for Noobs,20,50,distributed systems
3,Xen and the Art of Surviving Undergraduate School,18,150,undergraduate school
4,Cooking for the Impatient Undergrad,20,20,undergraduate school
```

Now, search for details of the "Distributed Systems" book.

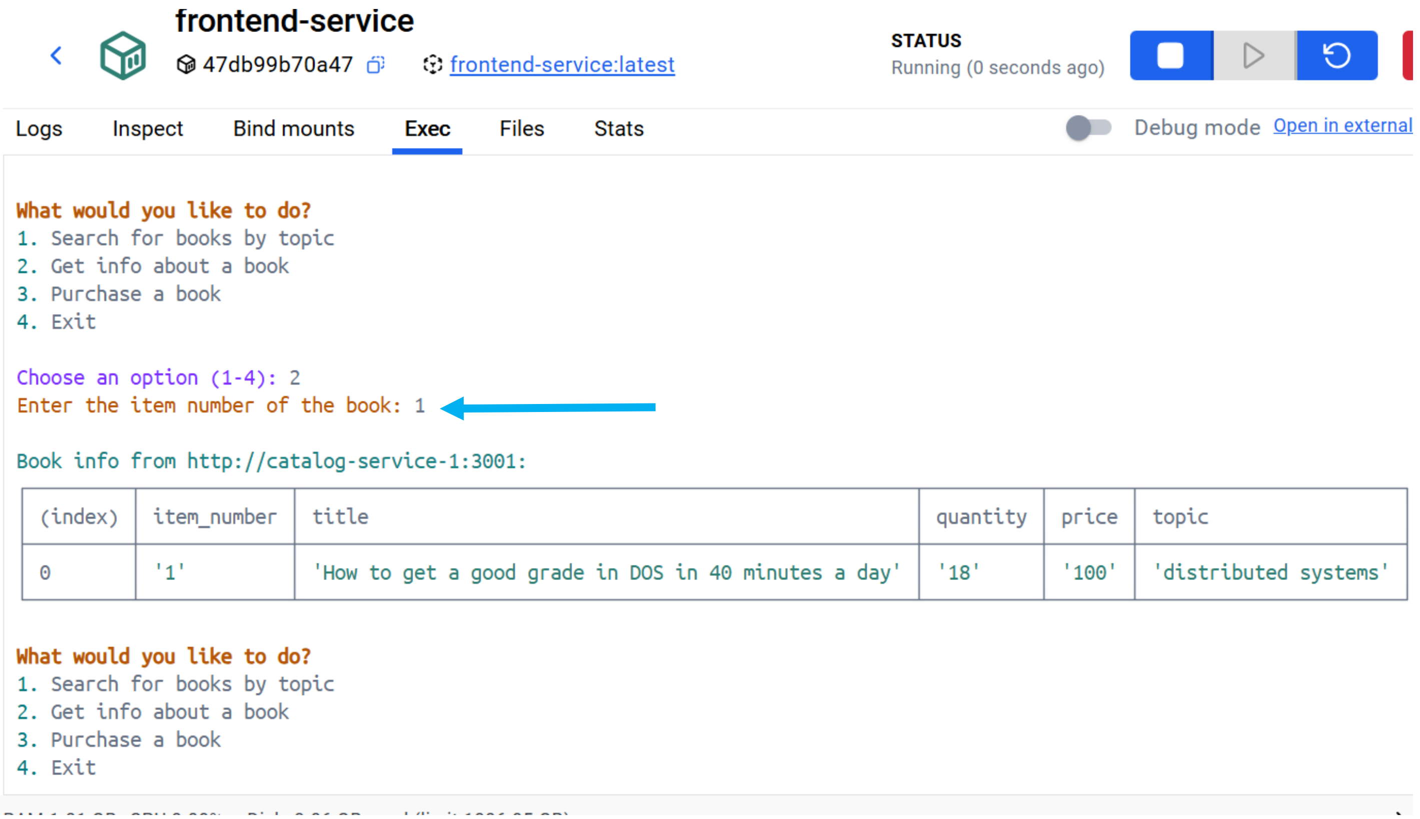


```
# node client.js

 Welcome to BAZAR.COM
 Your gateway to the world of books!

What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit

Choose an option (1-4): 1
```



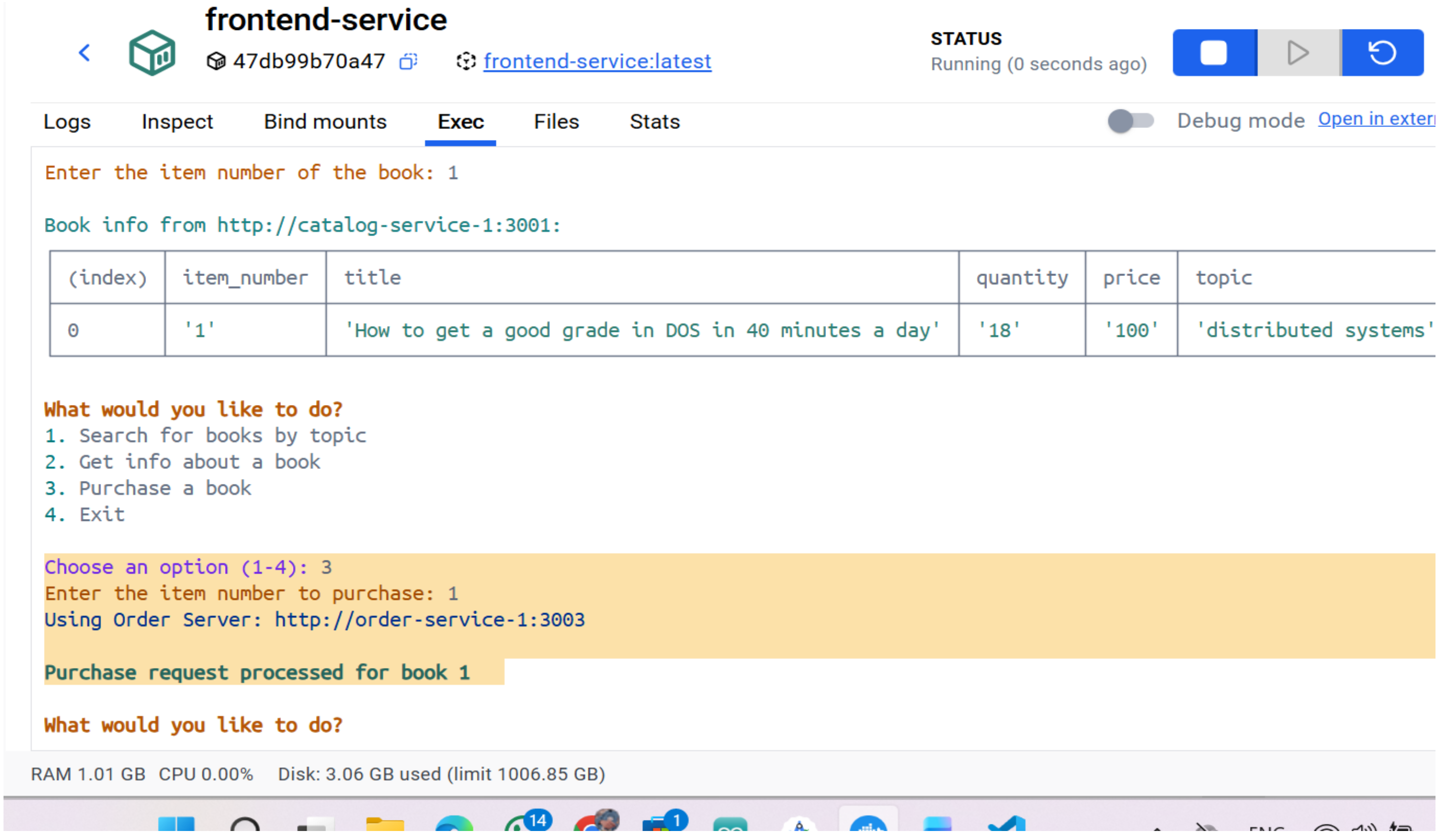
```
What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit

Choose an option (1-4): 1
Enter the topic: distributed systems

Books found from http://catalog-service-1:3001:
┌─────────┬─────────────┬───────────────────────────────────────────────────┬──────────┬───────┬───────────────────────┐
│ (index) │ item_number │ title                                             │ quantity │ price │ topic                 │
├─────────┼─────────────┼───────────────────────────────────────────────────┼──────────┼───────┼───────────────────────┤
│ 0       │ '1'         │ 'How to get a good grade in DOS in 40 minutes a day' │ '18'  │ '100' │ 'distributed systems' │
│ 1       │ '2'         │ 'RPCs for Noobs'                                  │ '20'     │ '50'  │ 'distributed systems' │
└─────────┴─────────────┴───────────────────────────────────────────────────┴──────────┴───────┴───────────────────────┘

What would you like to do?
```

Get information of a book from it's id



```
What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit

Choose an option (1-4): 2
Enter the item number of the book: 1

Book info from http://catalog-service-1:3001:
┌─────────┬─────────────┬──────────────────────────────────────────────┬──────────┬───────┬───────────────────────┐
│ (index) │ item_number │ title                                        │ quantity │ price │ topic                 │
├─────────┼─────────────┼──────────────────────────────────────────────┼──────────┼───────┼───────────────────────┤
│ 0       │ '1'         │ 'How to get a good grade in DOS in 40 minutes a day' │ '18'     │ '100' │ 'distributed systems' │
└─────────┴─────────────┴──────────────────────────────────────────────┴──────────┴───────┴───────────────────────┘

What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit
```

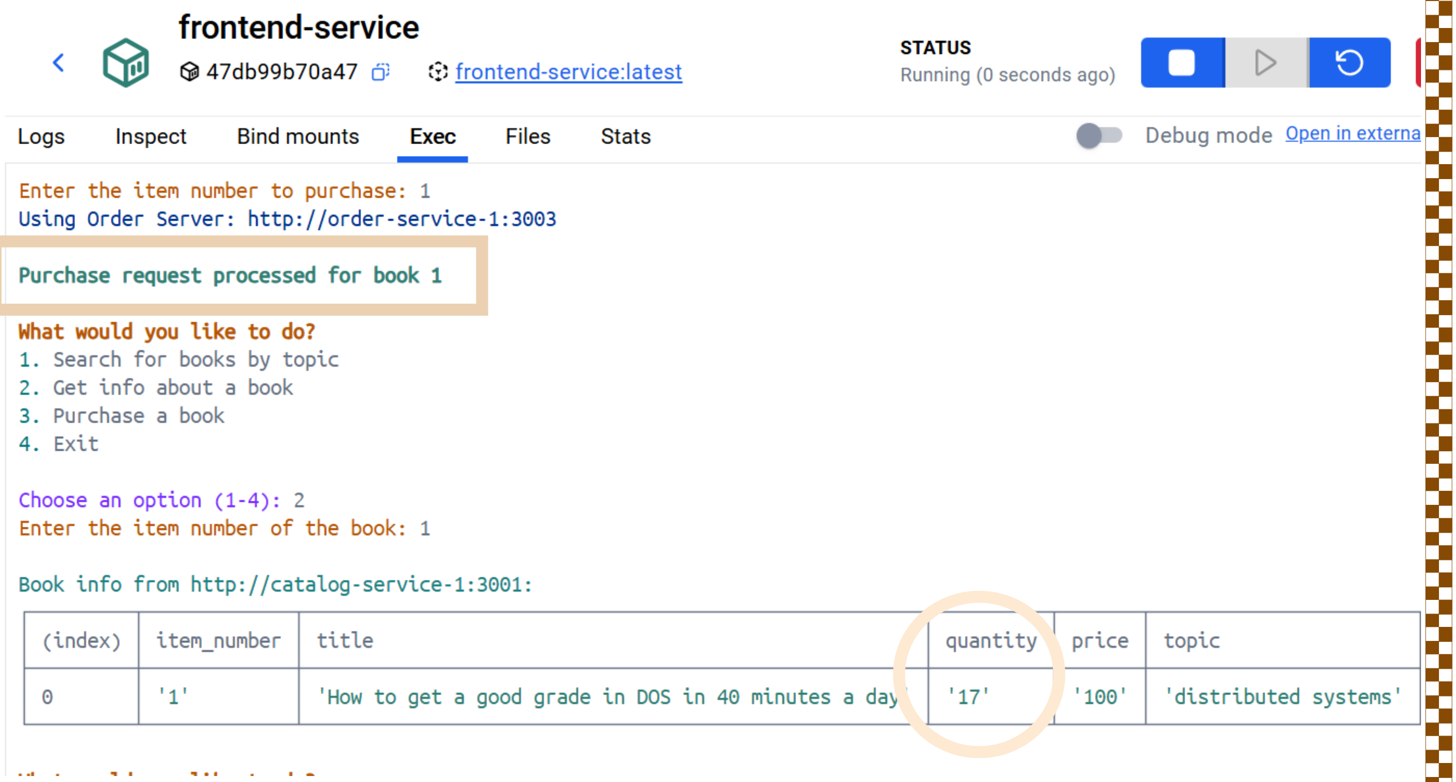
Purchase a book and the stock



```
Enter the item number of the book: 1

Book info from http://catalog-service-1:3001:
┌─────────┬─────────────┬──────────────────────────────────────────────┬──────────┬───────┬───────────────────────
│ (index) │ item_number │ title                                        │ quantity │ price │ topic
├─────────┼─────────────┼──────────────────────────────────────────────┼──────────┼───────┼───────────────────────
│ 0       │ '1'         │ 'How to get a good grade in DOS in 40 minutes a day' │ '18'     │ '100' │ 'distributed systems'
└─────────┴─────────────┴──────────────────────────────────────────────┴──────────┴───────┴───────────────────────

What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit

Choose an option (1-4): 3
Enter the item number to purchase: 1
Using Order Server: http://order-service-1:3003

Purchase request processed for book 1

What would you like to do?
```

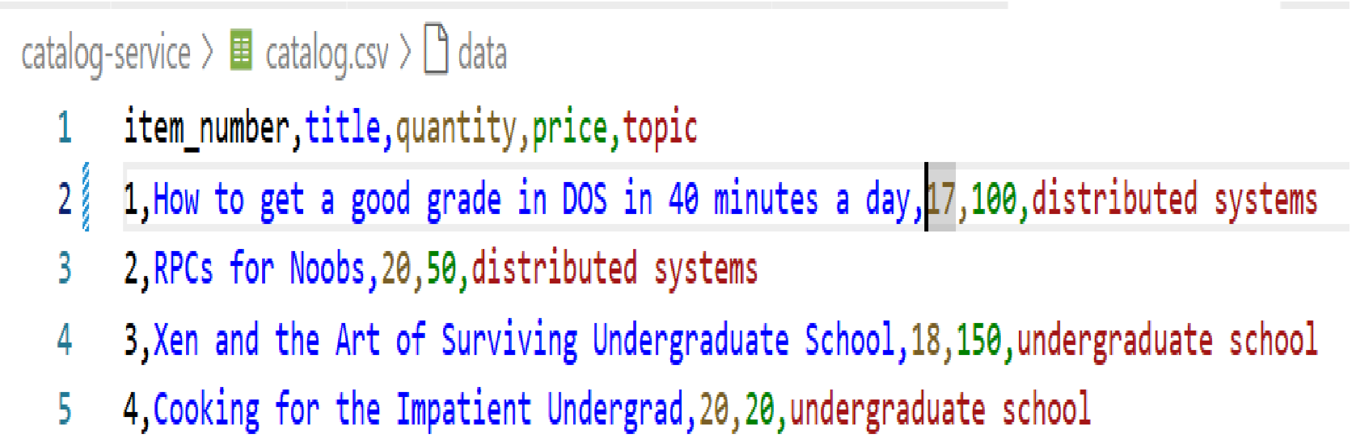
RAM 1.01 GB CPU 0.00%   Disk: 3.06 GB used (limit 1006.85 GB)

After purchase



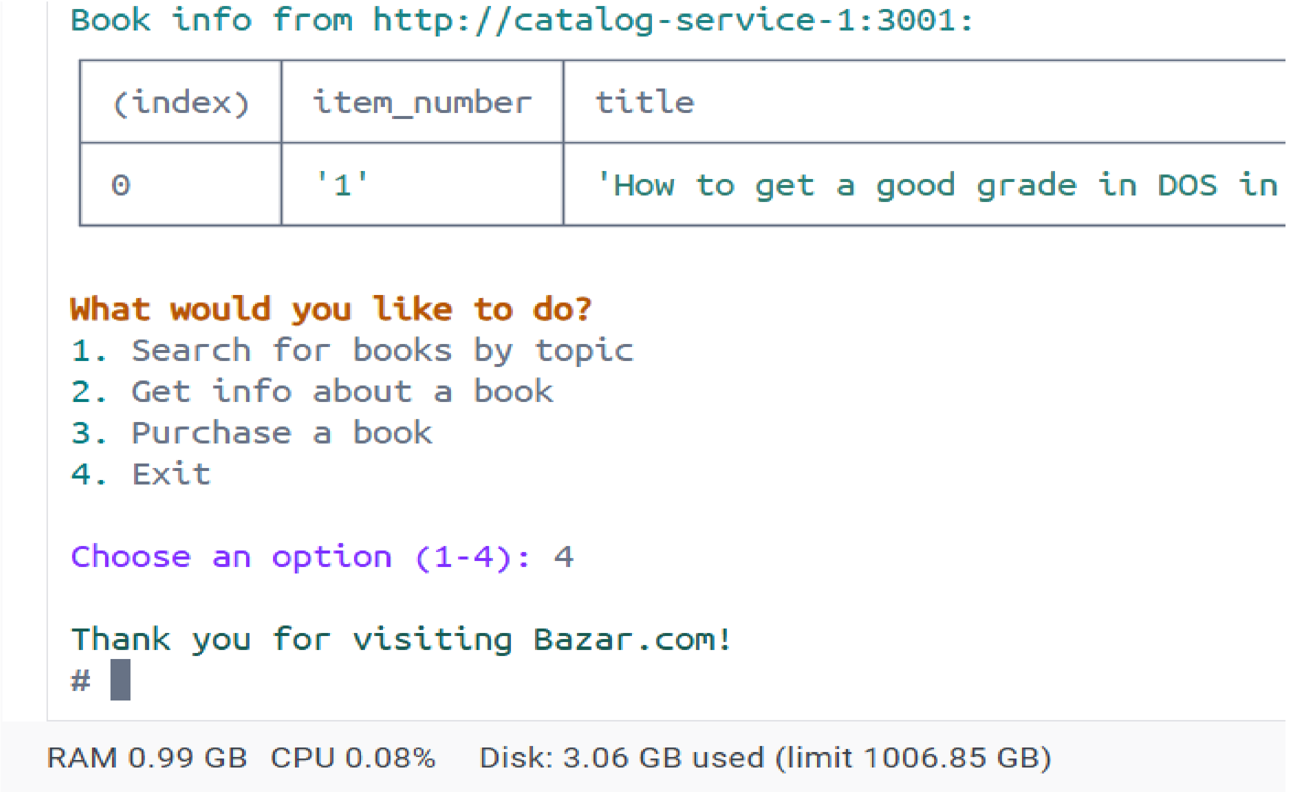
```
Enter the item number to purchase: 1
Using Order Server: http://order-service-1:3003

Purchase request processed for book 1

What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit

Choose an option (1-4): 2
Enter the item number of the book: 1

Book info from http://catalog-service-1:3001:
┌─────────┬─────────────┬──────────────────────────────────────────────┬──────────┬───────┬───────────────────────┐
│ (index) │ item_number │ title                                        │ quantity │ price │ topic                 │
├─────────┼─────────────┼──────────────────────────────────────────────┼──────────┼───────┼───────────────────────┤
│ 0       │ '1'         │ 'How to get a good grade in DOS in 40 minutes a day' │ '17'     │ '100' │ 'distributed systems' │
└─────────┴─────────────┴──────────────────────────────────────────────┴──────────┴───────┴───────────────────────┘
```

And it decrements by one in catalog.csv



```
item_number,title,quantity,price,topic
1,How to get a good grade in DOS in 40 minutes a day,17,100,distributed systems
2,RPCs for Noobs,20,50,distributed systems
3,Xen and the Art of Surviving Undergraduate School,18,150,undergraduate school
4,Cooking for the Impatient Undergrad,20,20,undergraduate school
```

Finally, we exit the program



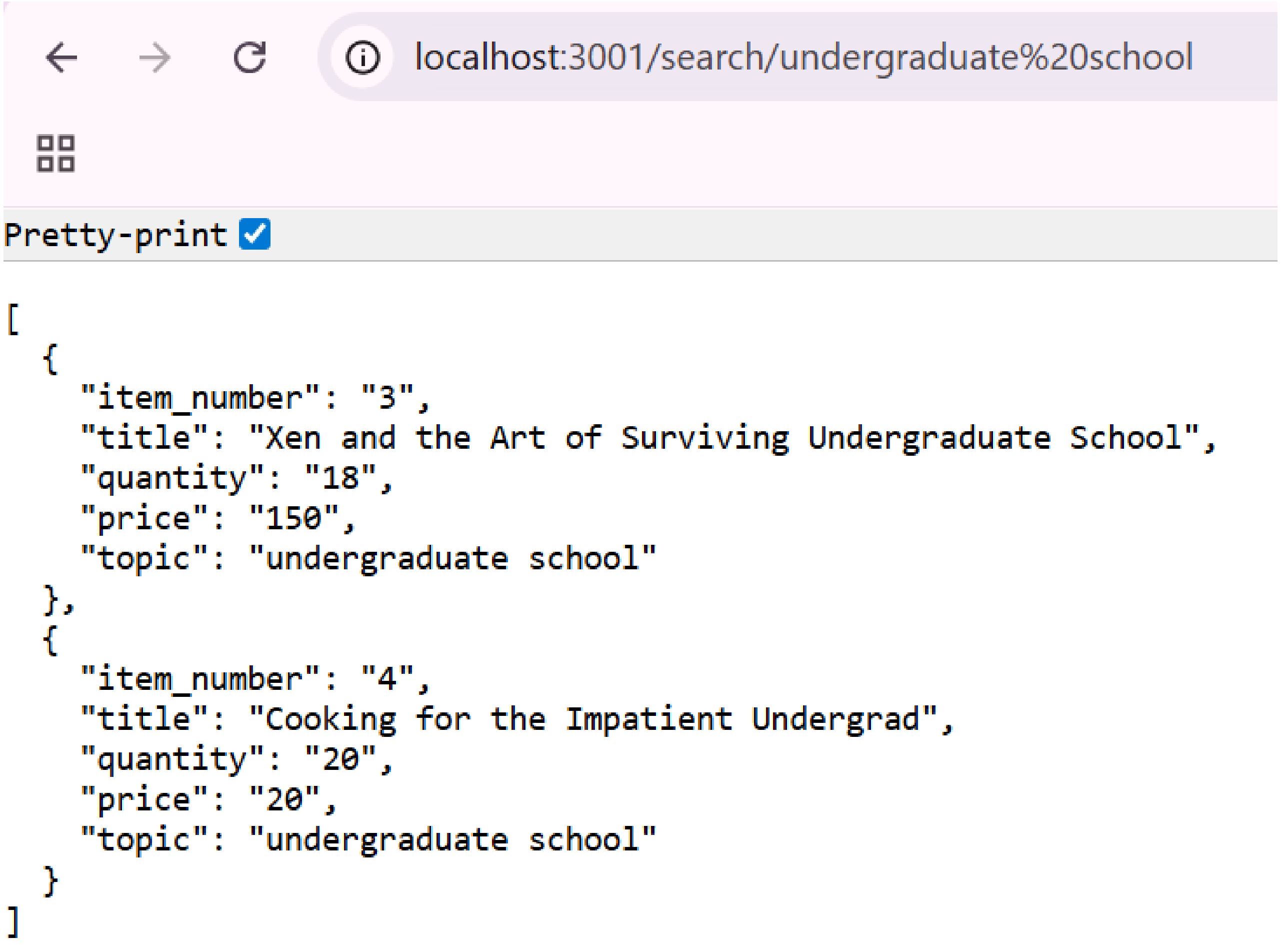
```
Book info from http://catalog-service-1:3001:
┌─────────┬─────────────┬──────────────────────────────────
│ (index) │ item_number │ title
├─────────┼─────────────┼──────────────────────────────────
│ 0       │ '1'         │ 'How to get a good grade in DOS in
└─────────┴─────────────┴──────────────────────────────────

What would you like to do?
1. Search for books by topic
2. Get info about a book
3. Purchase a book
4. Exit

Choose an option (1-4): 4

Thank you for visiting Bazar.com!
# 
```

RAM 0.99 GB   CPU 0.08%   Disk: 3.06 GB used (limit 1006.85 GB)

The **Catalog Service** can be accessed and tested both through the **terminal (CLI frontend)** and directly from a **web browser** using its RESTful endpoints.

When the system is running (via docker-compose up --build), the Catalog microservice listens on port **3001**, allowing users to view and test book information through simple HTTP requests.

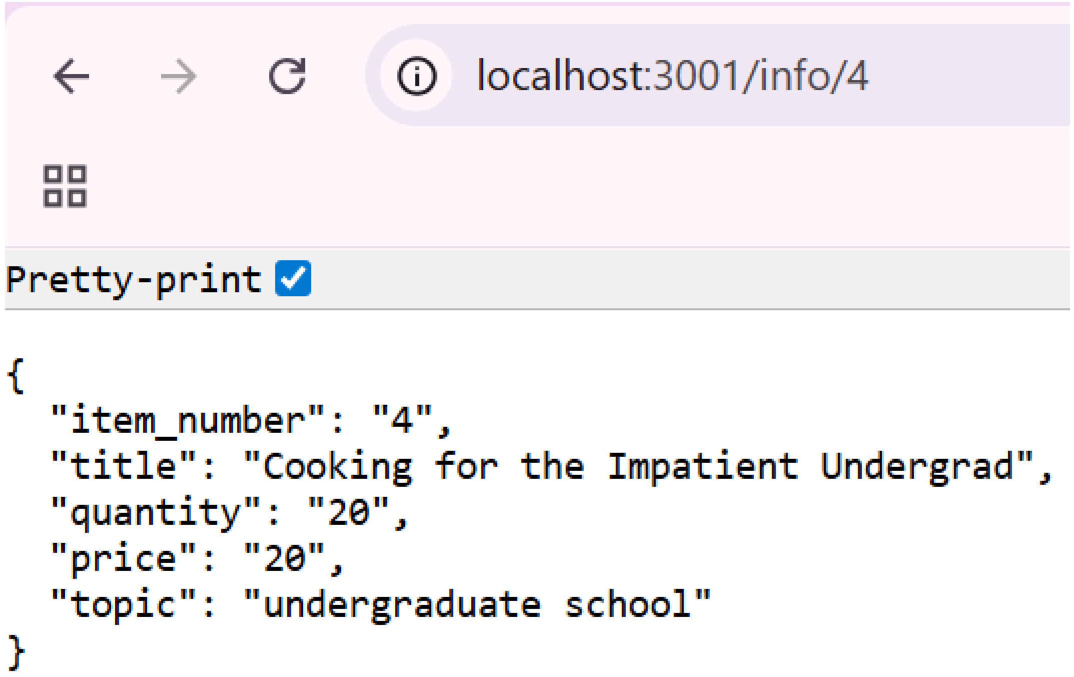
Search by topic:



Pretty-print ☑

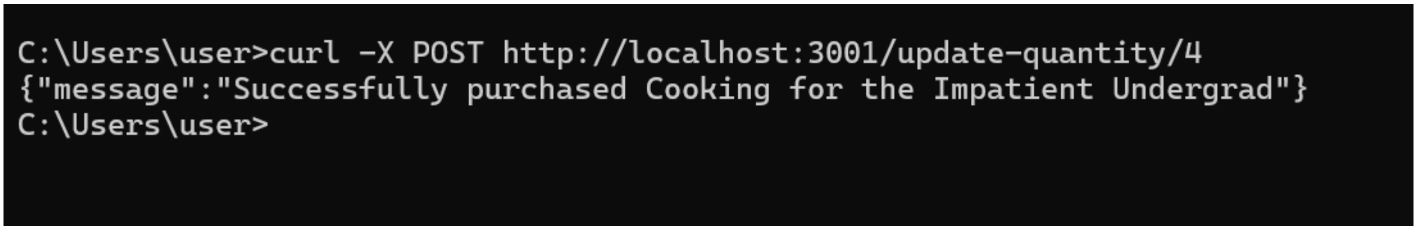
```
[
  {
    "item_number": "3",
    "title": "Xen and the Art of Surviving Undergraduate School",
    "quantity": "18",
    "price": "150",
    "topic": "undergraduate school"
  },
  {
    "item_number": "4",
    "title": "Cooking for the Impatient Undergrad",
    "quantity": "20",
    "price": "20",
    "topic": "undergraduate school"
  }
]
```

Get book details by item number:

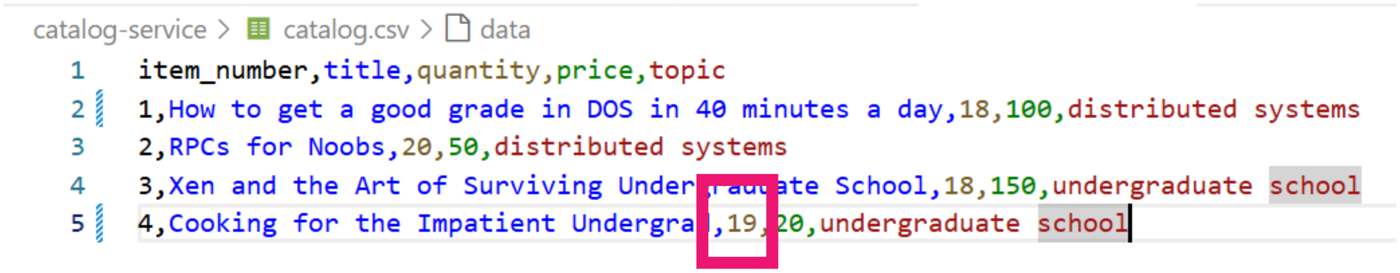


```
{
  "item_number": "4",
  "title": "Cooking for the Impatient Undergrad",
  "quantity": "20",
  "price": "20",
  "topic": "undergraduate school"
}
```

purchase and automatically decreases the stock quantity of the selected book in the catalog.csv file.



```
C:\Users\user>curl -X POST http://localhost:3001/update-quantity/4
{"message":"Successfully purchased Cooking for the Impatient Undergrad"}
C:\Users\user>
```



```
item_number,title,quantity,price,topic
1,How to get a good grade in DOS in 40 minutes a day,18,100,distributed systems
2,RPCs for Noobs,20,50,distributed systems
3,Xen and the Art of Surviving Undergraduate School,18,150,undergraduate school
4,Cooking for the Impatient Undergrad,19,20,undergraduate school
```

Design Trade-offs:

The system uses CSV files to store catalog and order data for simplicity and ease of setup. This approach is efficient for small-scale projects and reduces setup complexity. However, as the system grows, using a lightweight database such as SQLite would provide better performance, faster queries, and improved data consistency.

Known Limitations

- Concurrent Purchase Conflicts:
  - Multiple users purchasing the same book simultaneously may lead to overselling due to race conditions in stock verification.
- Data Persistence with CSV:
  - Concurrent writes to the CSV file can cause data corruption, leading to inconsistent inventory data.

Improvements and Extensions

- **Database Integration**:
  - Switching from a CSV file to a lightweight database such as SQLite would provide better scalability, faster queries, and improved data consistency.
- **Implement Docker Swarm or Kubernetes**:
  - By adopting Docker Swarm or Kubernetes for orchestration, the services can be scaled horizontally (i.e., adding more instances of each service) automatically. This is especially useful if you expect the catalog or order service to have high traffic at times.

Conclusion

This project successfully demonstrates a microservices-based architecture using Docker containers to manage catalog, order, and frontend services. By separating each service, the system achieves modularity, scalability, and easier maintenance. The use of Docker Compose simplifies deployment and networking between services. Although a CSV file was used for simplicity, the architecture can be easily extended to integrate databases or additional services in the future. Overall, the project provides a practical example of how microservices communicate and collaborate within a distributed environment.