

Bazar.com: A Multi-tier Online Book Store

Name:mustafa Ramahi

Id:11923890

Overall Program Design:

The program consists of three components: a Catalog Server, an Order Server, and a Front-End Server. The Catalog Server manages a collection of books, allowing users to search, retrieve information, and update stock levels. The Order Server handles the purchase process, coordinating with the Catalog Server to update stock and generating order IDs. The Front-End Server acts as a proxy, forwarding requests from clients to either the Catalog or Order Server.

How It Works:

Catalog Server:

Exposes endpoints for searching (/search/:topic), retrieving book information (/info/:item_number), and updating book details (/update/:item_number).

Handles these requests by interacting with a books module, which likely encapsulates book-related logic and data.

That it work on localhost:3000

Postman interface showing a GET request to `http://localhost:3000/info/1`. The response status is 200 OK, Time: 6 ms, Size: 325 B. The response body is displayed in the Pretty view:

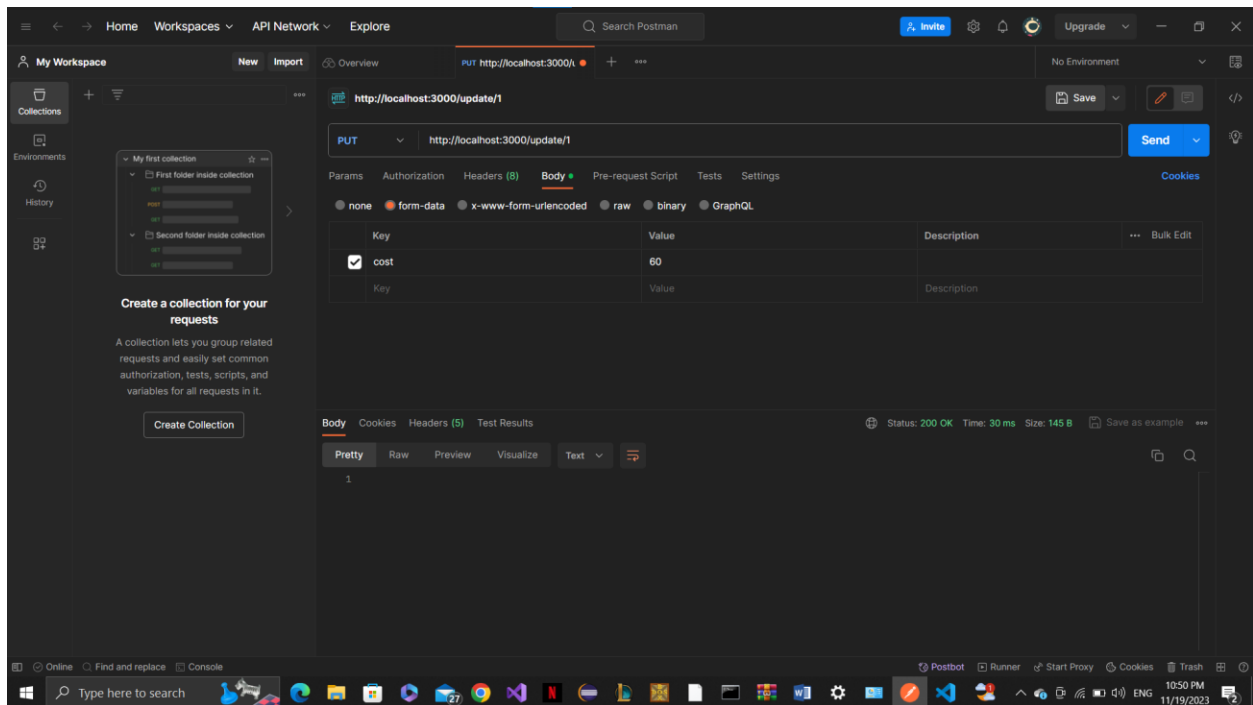
```
1 {
2   "title": "How to get a good grade in OOS in 40 minutes a day",
3   "itemsInStock": 38,
4   "cost": 49
5 }
```

The interface includes a sidebar with Collections, Environments, and History. The main workspace shows the request details, including Params, Headers (8), Body, Pre-request Script, Tests, and Settings. The response is shown in the Body tab with a status of 200 OK.

Postman interface showing a GET request to `http://localhost:3000/search/Undergraduate Book`. The response status is 200 OK, Time: 5 ms, Size: 379 B. The response body is displayed in the Pretty view:

```
1 {
2   "item_number": 3,
3   "title": "Xen and the Art of Surviving Undergraduate School"
4 },
5 {
6   "item_number": 4,
7   "title": "Cooking for the Impatient Undergrad"
8 }
9
10 }
```

The interface includes a sidebar with Collections, Environments, and History. The main workspace shows the request details, including Params, Headers (8), Body, Pre-request Script, Tests, and Settings. The response is shown in the Body tab with a status of 200 OK.



Order Server:

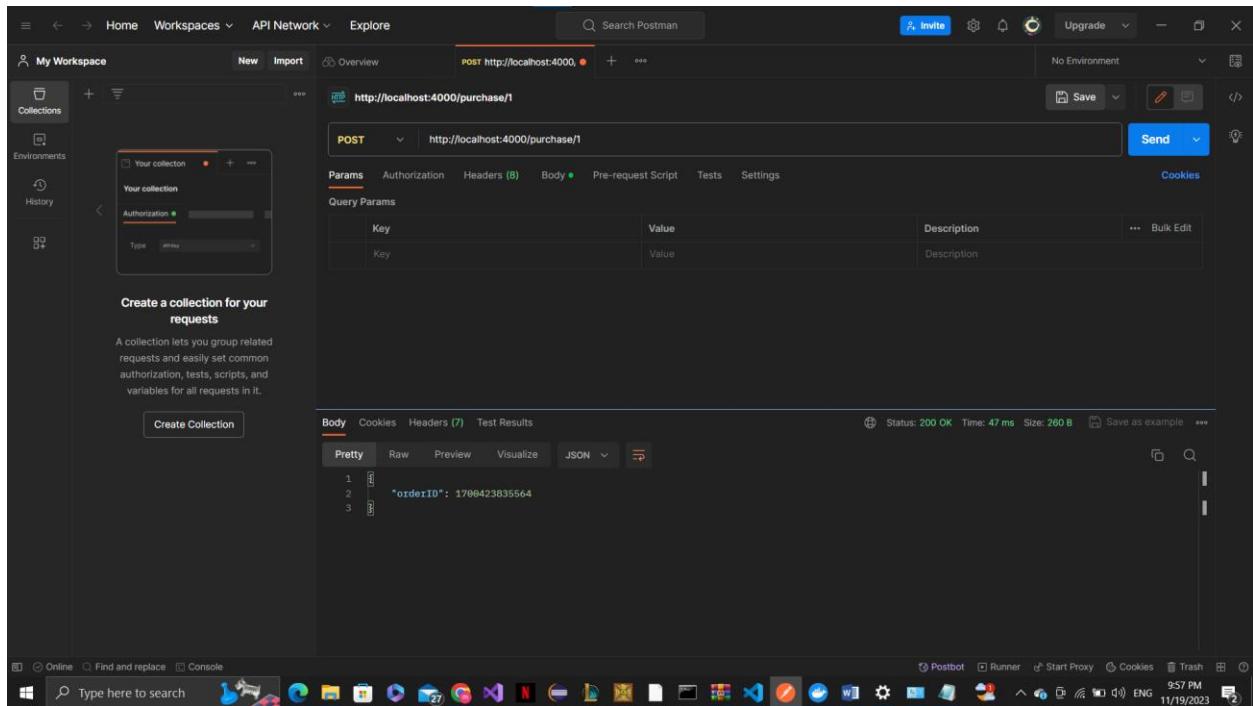
Listens for purchase requests (/purchase/:item_number).

Sends a GET request to the Catalog Server to check if the item is available.

If available, sends a PUT request to the Catalog Server to update stock and adds an order using the orders module.

Returns the order ID to the client.

That it work on localhost:4000



Front-End Server:

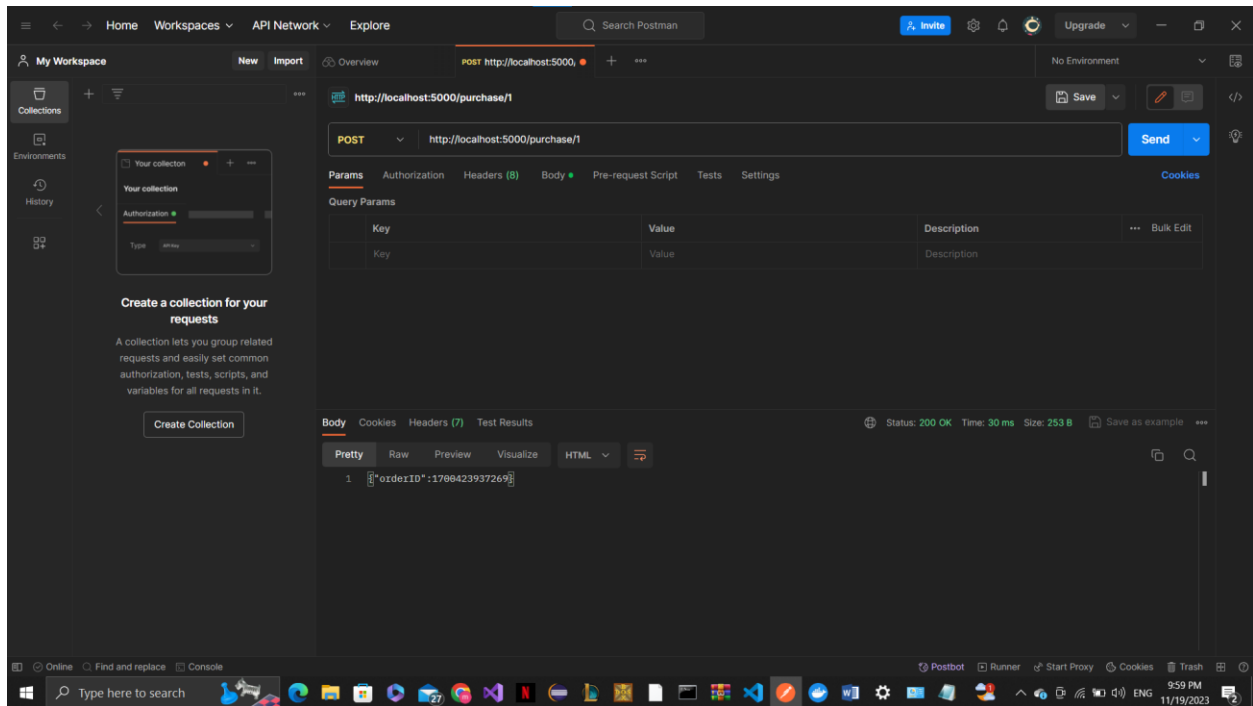
Acts as a middleware between the client and other servers.

Forwards requests to either the Catalog or Order Server based on the endpoint.

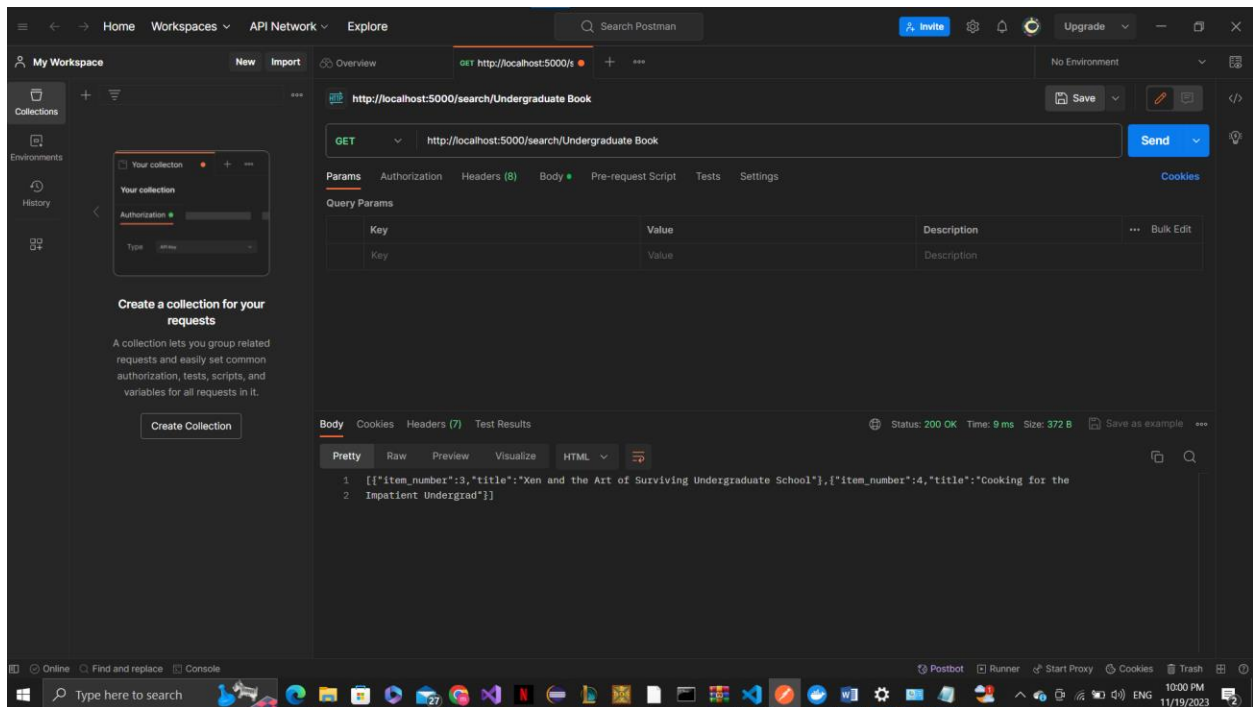
Uses the request module to make HTTP requests.

That it work on localhost:5000

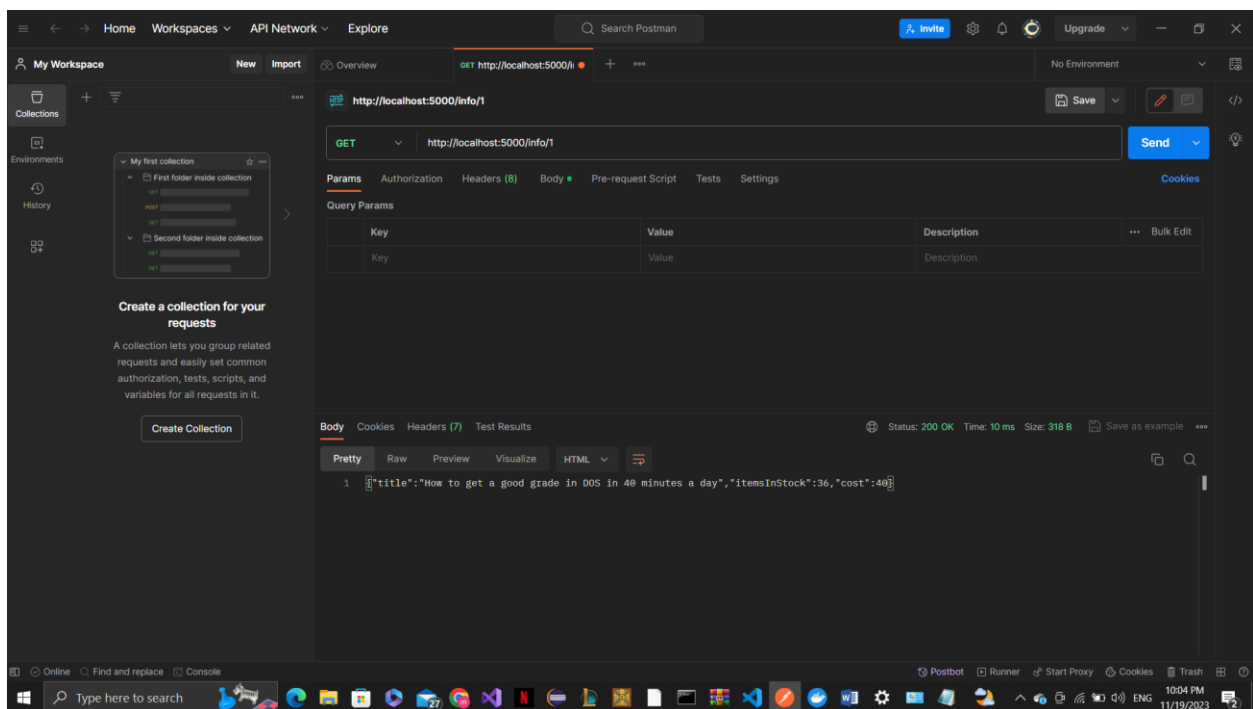
In this I make purchase from front end server.



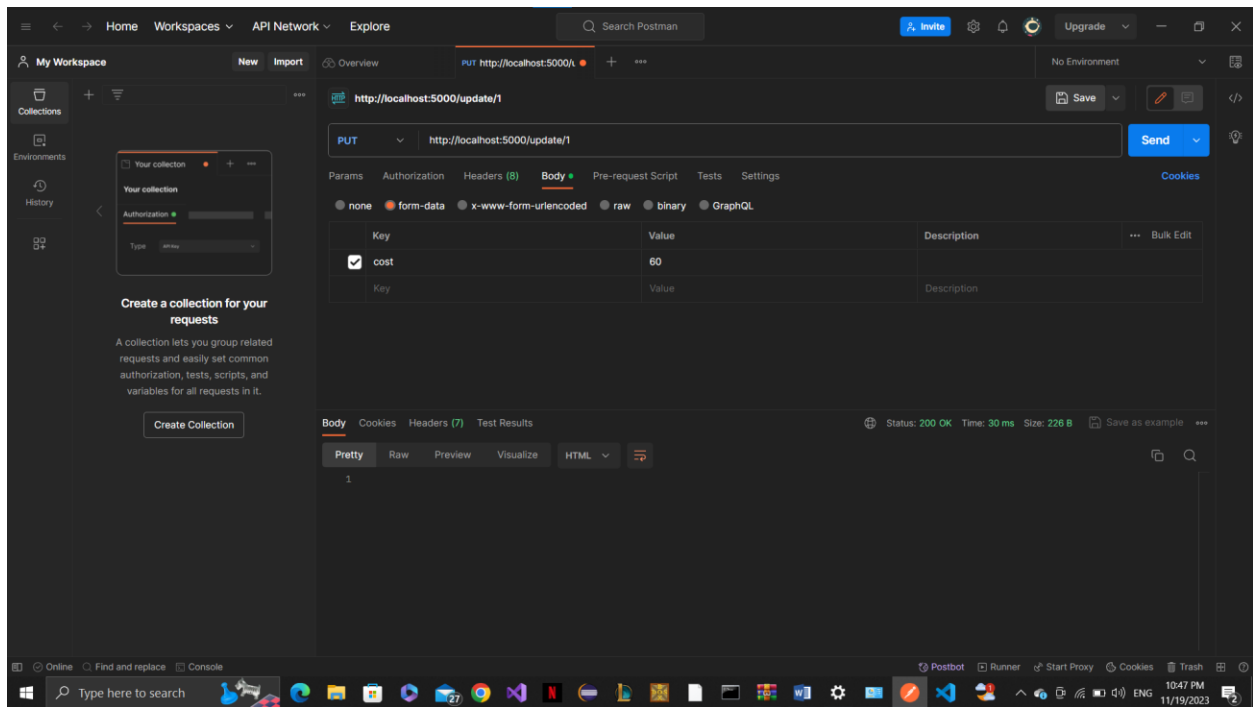
In this I make search from front end server.



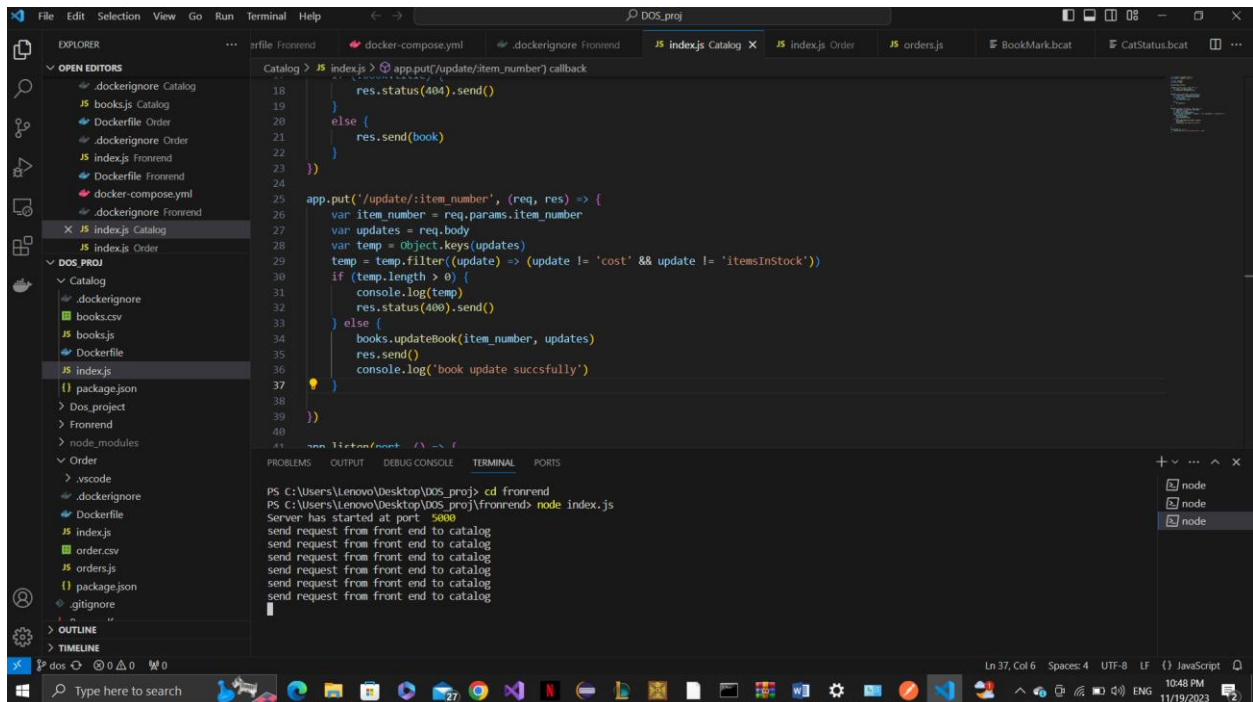
In this I make info from front end server.



In this I make update from front end server.



In this photo I print send request from front end to catalog.



Built from the front-end server's Dockerfile.

Docker Compose:

A Docker Compose file (docker-compose.yml) is provided to orchestrate the deployment of the three containers. It defines the services, their configurations, and the network they share. This simplifies the deployment process and allows running the entire application stack with a single command.

Running the Program with Docker:

Build Docker Images:

Navigate to each server's directory containing the Dockerfile.

Build the images using the command: `docker build -t <image-name>:<tag> .`

Docker Compose:

In the project root directory, run `docker-compose up`.

This command starts the defined services, creating containers for the Catalog, Order, and Front-End Servers.

Accessing Services:

The Catalog Server is accessible at `http://localhost:3000`.

The Order Server is accessible at `http://localhost:4000`.

The Front-End Server is accessible at `http://localhost:5000`.

Docker Containers:

The program has been containerized using Docker to simplify deployment and ensure consistent environments across different systems. There are three main containers:

Catalog Server Container:

Hosts the Catalog Server.

Exposes the necessary ports to communicate with the Front-End Server.

Built from the catalog server's Dockerfile.

Order Server Container:

Hosts the Order Server.

Exposes the necessary ports to communicate with the Front-End Server.

Built from the order server's Dockerfile.

Front-End Server Container:

Hosts the Front-End Server.

Exposes the necessary ports to accept client requests.

Built from the front-end server's Dockerfile.

Docker Desktop

Update to latest

Search for images, containers, volumes, extensions and more...

Ctrl+K

musta...

Containers

Images

Volumes

Dev Environments BETA

Docker Scout

Learning center

Extensions

Add Extensions

dos_proj

C:\Users\Lenovo\Desktop\DOS_proj

Open

dos_proj-fronrend-1

dos_proj-fronrend

Running

8081:3000

dos_proj-order-1

dos_proj-order

Running

8083:3000

dos_proj-catalog-1

dos_proj-catalog

Running

8085:3000

```
2023-11-19 21:23:28 dos_proj-catalog-1 |
2023-11-19 21:23:28 dos_proj-order-1 |
2023-11-19 21:23:28 dos_proj-fronrend-1 |
2023-11-19 21:23:28 dos_proj-catalog-1 | > Catalog@1.0.0 start /usr/src/app
2023-11-19 21:23:28 dos_proj-catalog-1 | > node index.js
2023-11-19 21:23:28 dos_proj-catalog-1 |
2023-11-19 21:23:28 dos_proj-order-1 | > order@1.0.0 start /usr/src/app
2023-11-19 21:23:28 dos_proj-order-1 | > node index.js
2023-11-19 21:23:28 dos_proj-fronrend-1 | > fronrend@1.0.0 start /usr/src/app
2023-11-19 21:23:29 dos_proj-catalog-1 | Server has started at port 3000
2023-11-19 21:23:28 dos_proj-order-1 |
2023-11-19 21:23:28 dos_proj-fronrend-1 | > node index.js
2023-11-19 21:23:28 dos_proj-fronrend-1 |
2023-11-19 21:23:29 dos_proj-order-1 | Server has started at port 4000
2023-11-19 21:23:29 dos_proj-fronrend-1 | Server has started at port 5000
```

Engine running

RAM 2.63 GB CPU 0.13% Signed in

v4.25.0

Docker Desktop

Update to latest

Search for images, containers, volumes, extensions and more...

Ctrl+K

musta...

Containers

Images

Volumes

Dev Environments BETA

Docker Scout

Learning center

Extensions

Add Extensions

Containers Give feedback

Container CPU usage ⓘ

0.00% / 800% (8 cores available)

Container memory usage ⓘ

85.48MB / 7.51GB

Show charts ▾

Search

Only show running containers

Delete

▶

⏸

■

<input checked="" type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input checked="" type="checkbox"/>	> dos_proj		Running (3/3)	0%		8 minutes ago	<div>■ ⋮ 🗑</div>

Selected 1 of 1

I make three containers and images