



An-Najah National University

Faculty of Engineering

Computer Engineering Department

Distributed Operation Systems

DOC Project part-1

Student's name:

Nancy Sawalmeh 12029645

Masa Abu Aisheh 12027844

Our project's services include:

1. Frontend service.
2. Order service.
3. Catalog service.

### **Frontend Service:**

The frontend service provides three main operations that interact with the catalog and order servers.

1. **Search:** Sends a search request to the catalog server, which returns the matching items.

**GET** <http://catalog:4401/search/:topic>

**Example:** **GET** [http://catalog:4401/search/Distributed systems](http://catalog:4401/search/Distributed%20systems)

2. **Info:** Requests specific information about an item from the catalog server.

**GET** <http://catalog:4401/info/:id>

**Example:** **GET** <http://catalog:4401/info/1>

3. **Purchase:** Sends a purchase order to the order server.

**POST** <http://order:4402/purchase/:id>

**Example:** **POST** <http://order:4402/purchase/1>

## Order Service:

The Order service acts as an intermediary between the front-end and the Catalog service. When a purchase request is received from the front-end, the Order service checks the stock level of the requested item in the Catalog database and updates the quantity if available.

1. **Receive Purchase Request:** The Order service listens for POST requests on the /purchase/:id endpoint, where id represents the item to be purchased.
2. **Check Stock:** The service queries the local catalog database for the stock level of the requested item. If the quantity is greater than zero, it proceeds with the purchase.
3. **Update Stock and Log Order:** If the item is in stock, the service reduces the quantity in the catalog table and logs the order in the orders table. This operation is performed in two steps:
  - 1- First, an UPDATE query decreases the stock quantity by 1.
  - 2- Then, an INSERT query logs the transaction in the orders table.
4. **Respond to Front-End:**
  - ✓ If the purchase is successful, it responds with a confirmation message, "Purchase successful".
  - ✓ If the item is out of stock, it returns an error message, "Item out of stock".

The Order service listens on port 4402, awaiting purchase requests.

## Example Usage:

**POST <http://localhost:4402/purchase/1>**

## **Catalog Service:**

The Catalog service provides a REST API to manage and retrieve information about books in the catalog. It offers endpoints for searching books by topic, retrieving detailed information about a specific book by its ID, and updating book stock or pricing.

### **1. Search Books by Topic:**

- ✓ Endpoint: GET /search/:topic
- ✓ Description: Retrieves a list of books that match the specified topic.
- ✓ Database Query: Selects "id" and "title" of books where the topic matches the parameter.
- ✓ Response: Returns the list of matching books or an error if the retrieval fails.

### **2. Get Book Info by Item ID:**

- ✓ Endpoint: GET /info/:id
- ✓ Description: Fetches detailed information about a specific book based on its unique "id".
- ✓ Database Query: Selects all columns from the "catalog" table where the "id" matches the parameter.
- ✓ Response: If a book is found, returns its details; if not, returns a 404 error with a message indicating no book with the specified ID exists.

### **3. Update Book Stock or Price:**

- ✓ Endpoint: PUT /update/:id
- ✓ Description: Updates the stock quantity and price of a book by id.
- ✓ Database Query: Updates the "quantity" and "price" fields in the catalog table.
- ✓ Request Body: Expects "quantity" and "price" as fields in the request body.

- ✓ Response: Returns a success message if the book is updated successfully, or an error if the update fails.

The Catalog service listens on port 4401 and is ready to handle requests from other services, such as the Order service or a front-end client.

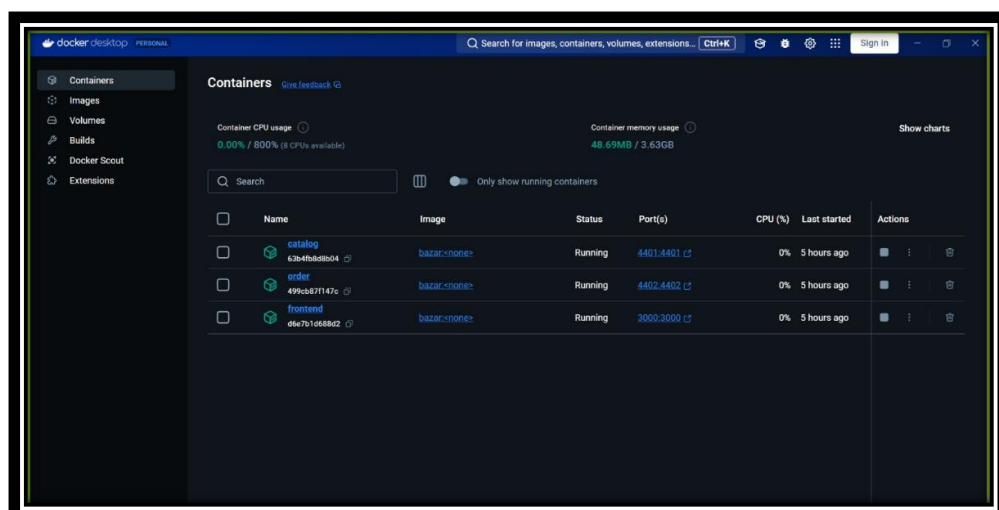
### Example Usage:

- ✓ Search by topic: **GET** <http://localhost:4401/search/fiction>
- ✓ Get book info: **GET** <http://localhost:4401/info/1>
- ✓ Update book details: **PUT** <http://localhost:4401/update/1> with JSON body:

```
{  
  "quantity": 5,  
  "price": 19.99  
}
```

### Running the project:

- 1- Building docker image:
  - `docker build -t bazar .`

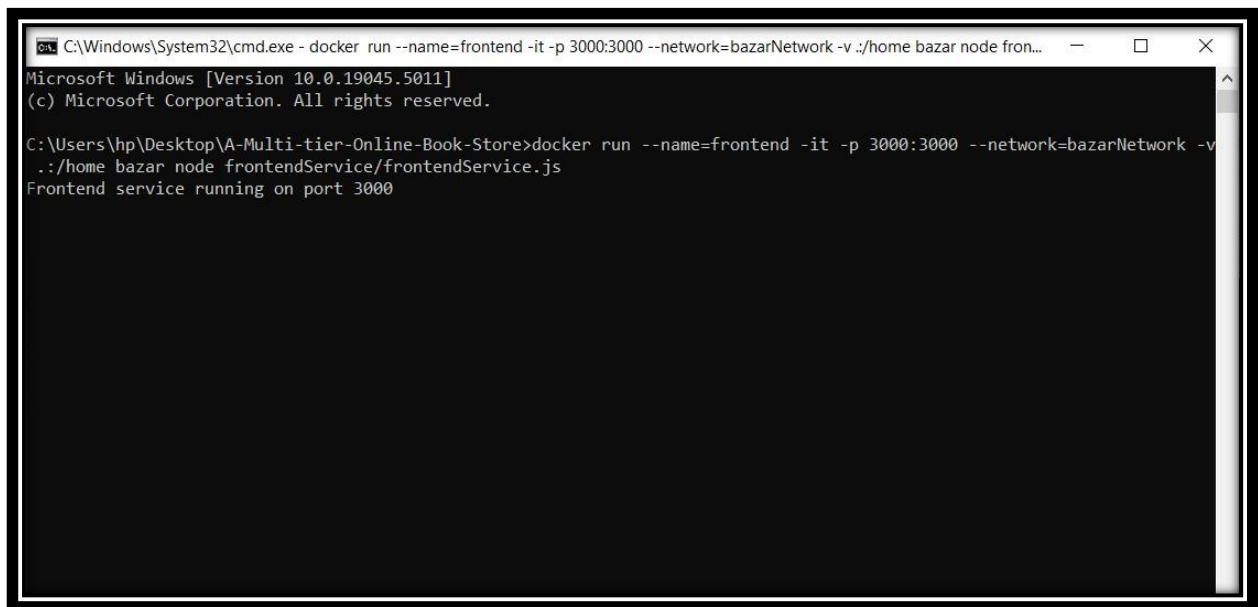


2- Creating common network to allow services to communicate:

- `docker network create bazarNetwork`

3- Running docker container for frontend server at port 3000 and hostname=frontend:

- `docker run --name=frontend -it -p 3000:3000 --network=bazarNetwork -v ./home bazar node frontendService/frontendService.js`

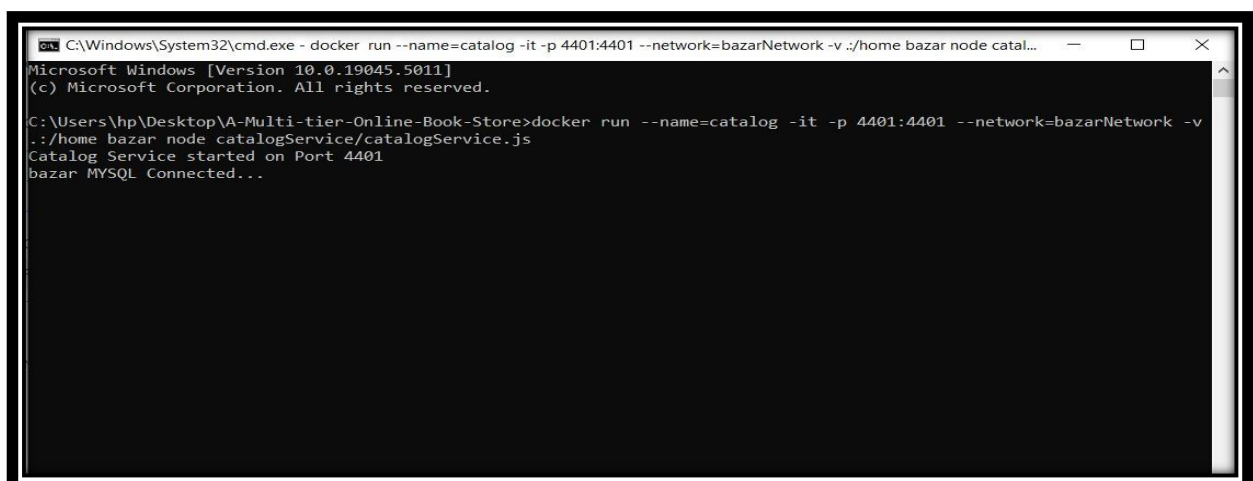


```
C:\Windows\System32\cmd.exe - docker run --name=frontend -it -p 3000:3000 --network=bazarNetwork -v ./home bazar node fron...
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp\Desktop\A-Multi-tier-Online-Book-Store>docker run --name=frontend -it -p 3000:3000 --network=bazarNetwork -v
./home bazar node frontendService/frontendService.js
Frontend service running on port 3000
```

4- Running docker container for catalog server at port 4401 and hostname=catalog:

- `docker run --name=catalog -it -p 4401:4401 --network=bazarNetwork -v ./home bazar node catalogService/catalogService.js`

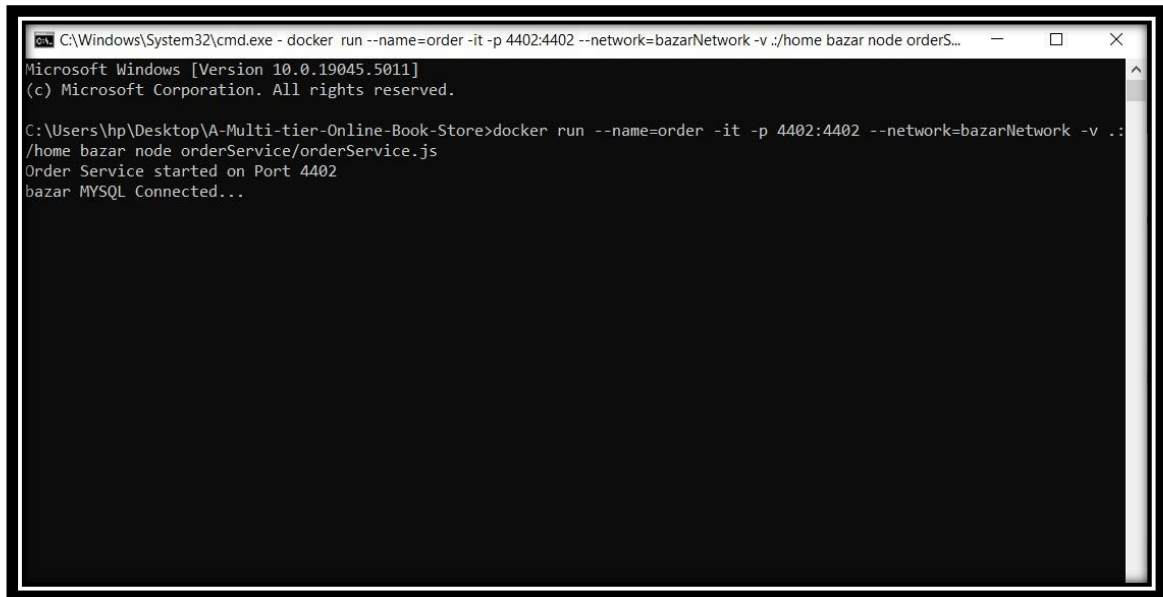


```
C:\Windows\System32\cmd.exe - docker run --name=catalog -it -p 4401:4401 --network=bazarNetwork -v ./home bazar node catal...
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp\Desktop\A-Multi-tier-Online-Book-Store>docker run --name=catalog -it -p 4401:4401 --network=bazarNetwork -v
./home bazar node catalogService/catalogService.js
Catalog Service started on Port 4401
bazar MYSQL Connected...
```

5- Running docker container for order server at port 4402 and hostname=order:

- `docker run --name=order -it -p 4401:4401 --network=bazarNetwork -v ./:/home bazar node orderService/orderService.js`



```
C:\Windows\System32\cmd.exe - docker run --name=order -it -p 4402:4402 --network=bazarNetwork -v ./:/home bazar node orderS...
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp\Desktop\A-Multi-tier-Online-Book-Store>docker run --name=order -it -p 4402:4402 --network=bazarNetwork -v ./:/home bazar node orderService/orderService.js
Order Service started on Port 4402
bazar MYSQL Connected...
```

Inside frontend container: `node frontendService.js`

Inside catalog container: `node catalogService.js`

Inside order container: `node orderService.`