



Computer Engineering Department (2024-2025)
Distributed Operating Systems (10636456)

Microservices Project Part1

Prepared By:

Jood Hamdallah(12028227) & Saleh Sawalha(12042238)

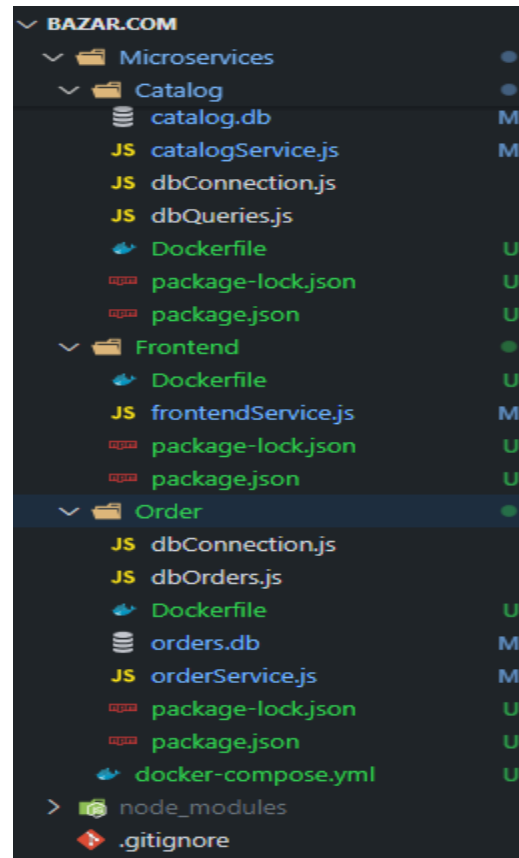
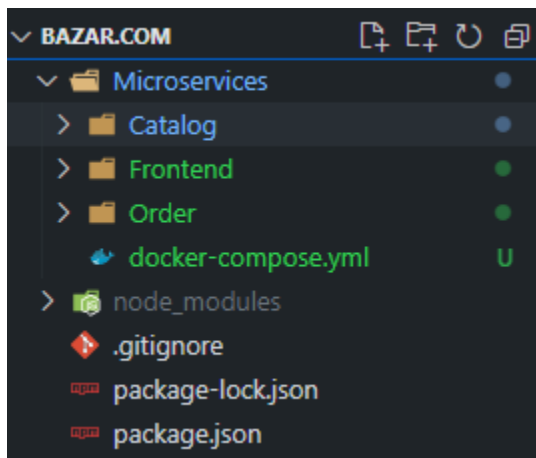
1. Overview:

This project involves building Bazar.com, a simple online bookstore using a two-tier, microservices architecture. With a front-end for user interactions and back-end services for catalog and order management, each service is containerized using Docker. This setup demonstrates RESTful APIs, SQLite data persistence, and inter-service communication in a distributed environment.

2. Implementation:

We used node js framework to complete this project as it provides powerful microservices that are asynchronous and lightweight. Our project consists of three services, catalog service, order service and frontend service which is used to accept user requests and perform initial processing.

This is the project hierarchy, each service has its own database, server and Docker file.



❖ Frontend Service:

This service runs on port 4000. It handles user interactions, allowing users to search for books by topic, view book details, and initiate purchases. It communicates with two back-end services—catalog and order.

It supports the following operations:

1. **search(topic):** Allows users to search for books by a specified topic, returning titles and item numbers for matching books. It can be accessed by the following url: **GET http://localhost:4000/search/{topic_name}**
2. **info(item_number):** Retrieves detailed information about a specific book, including its stock quantity and price, based on the item number. It can be accessed by the following url:
GET http://localhost:4000/info/{item_number}
3. **purchase(item_number):** Initiates a purchase request for a specified book by item number, verifying availability and processing the order if the item is in stock. It can be accessed by the following url:
POST <http://localhost:4000/purchase> , the item number is sent inside the body of the request such as **`{"item_number":1}`**

❖ Catalog Service:

Running on port 3000, the catalog service manages bookstore inventory, including stock, price, and topic data. It handles RESTful requests from the front-end for book searches, item details, and updates stock on purchases.

It supports the following operations:

1. **query-by-subject(topic):** Returns all books under a specified topic, including their titles and item numbers. It can be accessed by the following url: **GET http://localhost:3000/search/{topic_name}**
2. **query-by-item(item_number):** Provides detailed information for a specific book based on its item number, including stock quantity and price. It can be accessed by the following url:
GET http://localhost:3000/info/{item_number}

3. **update(item_number, newStock, newPrice):** Allows modifications to a book's stock or price, reflecting changes in inventory. It can be accessed by the following url: **PUT <http://localhost:3000/update>**
The request body should contain the item_number, newStock, and newPrice as follows: **`{"item_number":1, "newStock": 5, "newPrice": 25 }`**

❖ Order Service:

This service runs on port 3001 and is responsible for processing purchase requests. When a user initiates a purchase through the front-end, the order service verifies item availability by querying the catalog service and then updates the stock upon successful order completion. It supports the following operation:

1. **purchase(item_number):** Processes a purchase request for a specified book. It first verifies the item's availability by querying the catalog service, then decreases the stock by one if the item is available. The operation can fail if the item is out of stock.

This operation can be accessed by the following url:

POST <http://localhost:3001/purchase>

The request body should contain the item number as follows:

`{"item_number": 1 }`

Containerization in the Project:

For deploying Bazar.com, containerization is used to isolate each service (frontend, catalog, and order) within its own Docker container, allowing them to run independently on separate environments. This approach simplifies deployment and ensures that dependencies for each service are contained, making the system more portable and manageable.

We did the following steps to achieve containerization:

- Create a Dockerfile for each service (frontend, catalog, order) to define how to set up and run each one in its own container.
- Build each Docker image with docker build, packaging the necessary files and dependencies for each service using **docker build**.








```
C:\Users\joodh\OneDrive\Desktop\Bazar.com\Microservices>cd Catalog

C:\Users\joodh\OneDrive\Desktop\Bazar.com\Microservices\Catalog>docker build -t catalog .
[+] Building 65.9s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 341B
=> [internal] load metadata for docker.io/library/node:14
```

- Run each container using **docker run** to start each service and map its internal port to the host, allowing external access.
- Set up docker-compose.yml to configure and link all services, specifying their ports, dependencies, and shared network.
- Run docker-compose up to automatically build and launch all services together, making the entire application ready to use in one command. Docker Compose automates and simplifies the deployment process, making it easier to develop, test, and scale a multi-container application like Bazar.com.

```
C:\Users\joodh\OneDrive\Desktop\Bazar.com\Microservices>docker-compose up --build
time="2024-10-28T18:26:38+02:00" level=warning msg="C:\\Users\\joodh\\OneDrive\\Desktop\\Bazar.com\\Microservices\\docker-compose.yml: 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 4.8s (28/28) FINISHED
=> [catalog internal] load build definition from Dockerfile
=> => transferring dockerfile: 341B
=> [frontend internal] load metadata for docker.io/library/node:14
=> [catalog auth] library/node:pull token for registry-1.docker.io
```

```
=> => unpacking to docker.io/library/microservices-frontend:latest
=> [frontend] resolving provenance for metadata file
[+] Running 3/3
✔Container catalog    Recreated
✔Container order      Recreated
✔Container frontend   Recreated
Attaching to catalog, frontend, order
catalog | Catalog service running on port 3000
catalog | Connected to the catalog.db database
order   | Order service running on port 3001
order   | Connected to the orders database.
order   | Purchases table initialized.
frontend | Frontend service running on port 4000
```

<input type="checkbox"/>	▼  microservices	Running (3/3)
<input type="checkbox"/>	 catalog 8e823bb37014  microservices-catalog:<none>	Running 3000:3000 ↗
<input type="checkbox"/>	 order 397e0193259d  microservices-order:<none>	Running 3001:3001 ↗
<input type="checkbox"/>	 frontend 18a22d18a242  microservices-frontend:<none>	Running 4000:4000 ↗

Possible improvements and extensions:

Expanding the database and adding microservices would make Bazar.com more powerful and adaptable. With a larger database storing user profiles, book details, and order histories, we could support personalized features like recommendations and purchase records. Adding separate microservices, such as a user service for accounts, a cart service for multi-item purchases, and a notification service for stock updates, would let each part of the app work independently, making it easier to scale and maintain. This modular setup keeps the system organized, reliable, and ready to grow with user needs.