

An-Najah National University
Computer Engineering Department



Distributed Operating System (DOS)

BookStore MicroServices System

Name of Student :

Afaf Nasr & Yasmine Saad

Dr .Samer Arandi

Introduction

This project demonstrates a simple bookstore system based on microservices using Node.js, Express, SQLite, and Docker. It is designed to illustrate inter-service communication, isolation, and deployment using container-based architecture

Technologies Used

Node.js → Backend runtime

Express.js → Web framework

SQLite → Lightweight database per service

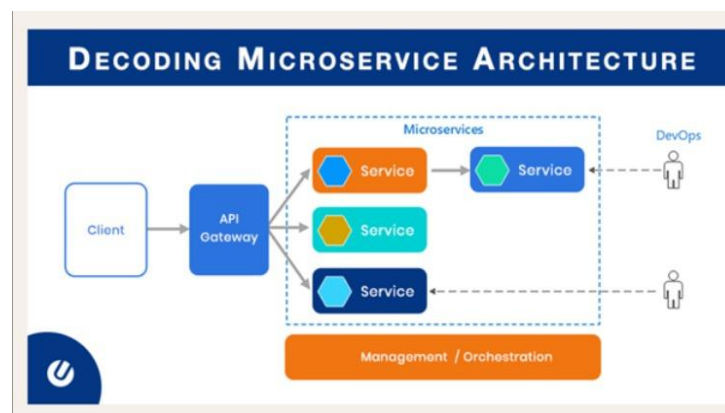
Docker → Containerization

Postman → Testing APIs

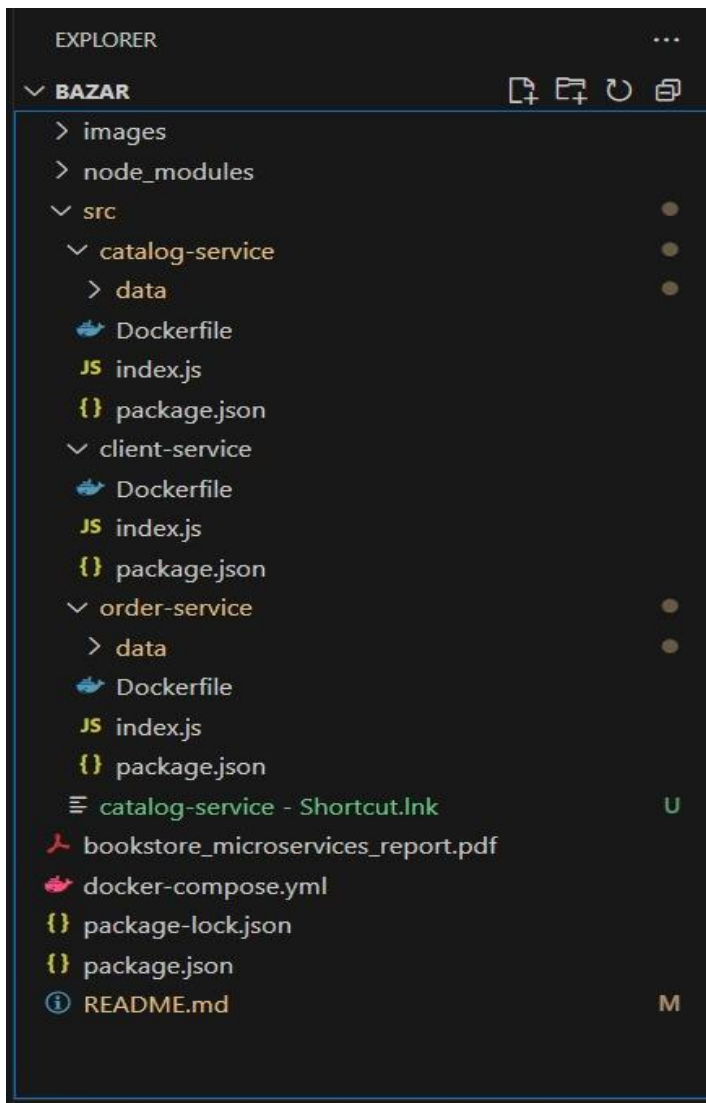
Git/GitHub → Version control

System Architecture

The client interacts with the system via the front-end (client service). Requests are routed to either the catalog or order services using REST API calls. Each service handles its own database and logic independently



Project Structure



src/order-service : Implements the Order Service — handles book purchases and order tracking

src/order-service/index.js : Main entry point; processes `/purchase/:id` and records orders

src/order-service/Dockerfile : Builds the Docker image for the Order Service

src/order-service/data : Holds the SQLite database for order management

src/order-service/data/orders.db : Stores order history and purchased book records

src/client-service : Implements the Client Service — acts as API gateway and entry point for users

src/catalog-service : Manages book data: search, details, and catalog browsing

src/client-service : Acts as an API gateway or frontend for interacting with users

API Endpoints & Screenshots

Catalog Service — <http://localhost:5001>:

GET /search/:topic → Search books by topic

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:5000/search/distributed`
- Method:** `GET`
- Status:** `200 OK` (40 ms, 341 B)
- Response Body (JSON):**

```
1 [
2   {
3     "id": 1,
4     "title": "How to get a good grade in DOS in 48 minutes a day"
5   },
6   {
7     "id": 2,
8     "title": "RPCs for Noobs"
9   }
10 ]
```

GET /info/:id → Get book details

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:5000/info/2
- Status:** 200 OK
- Response Time:** 18 ms
- Response Size:** 288 B
- Body:** A JSON object representing book details:

```
{  "title": "RPCs for Noobs",  "quantity": 2,  "price": 25.99}
```

POST /reserve/:id → Reserve a book

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5001/reserve/1
- Status:** 200 OK
- Response Time:** 42 ms
- Response Size:** 308 B
- Body:** A JSON object representing reservation details:

```
{  "title": "How to get a good grade in DOS in 40 minutes a day",  "price": 30}
```

PUT /update/:id → Update book price/quantity

The screenshot shows a REST client interface for a service named "My Bazar / Catalog Service". The endpoint is `http://localhost:5001/update/2` and the method is `PUT`. The request body is a JSON object: `{ "price": 20 }`. The response is a `200 OK` status with a response time of 40 ms and a body size of 274 B. The response body is a JSON object: `{ "message": "Book updated successfully" }`.

```
1 {
2   "price": 20
3 }
```

```
1 {
2   "message": "Book updated successfully"
3 }
```

Order Service — <http://localhost:5002>

POST /purchase/:id → Purchase and store order

The screenshot shows a REST client interface for a service named "My Bazar / Client Service". The endpoint is `http://localhost:5000/purchase/2` and the method is `POST`. The response is a `200 OK` status with a response time of 85 ms and a body size of 302 B. The response body is a JSON object: `{ "status": "ok", "message": "bought book RPCs for Noobs", "order_id": 9 }`.

Key	Value	Description
Key	Value	Description

```
1 {
2   "status": "ok",
3   "message": "bought book RPCs for Noobs",
4   "order_id": 9
5 }
```

GET /health → Health check

The screenshot shows a REST client interface for a service named "My Bazar / Catalog Service / Health Check". The request is a GET to `http://localhost:5001/health`. The response is a 200 OK status, received in 8 ms with a body size of 245 B. The response body is displayed in JSON format:

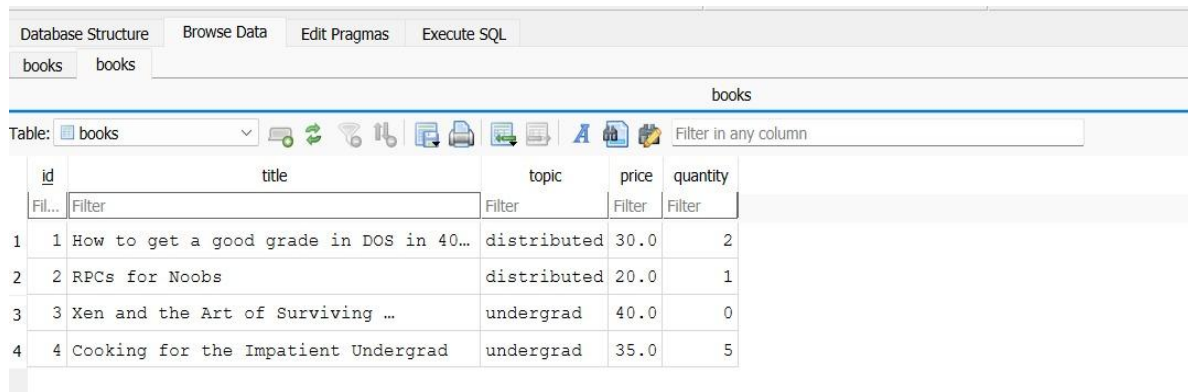
```
1 {
2   "ok": true
3 }
```

If the user want to purchash a book :

The screenshot shows a REST client interface for a service named "My Bazar / Order Service / Purchase Book". The request is a POST to `http://localhost:5002/purchase/3`. The response is a 200 OK status, received in 74 ms with a body size of 339 B. The response body is displayed in JSON format:

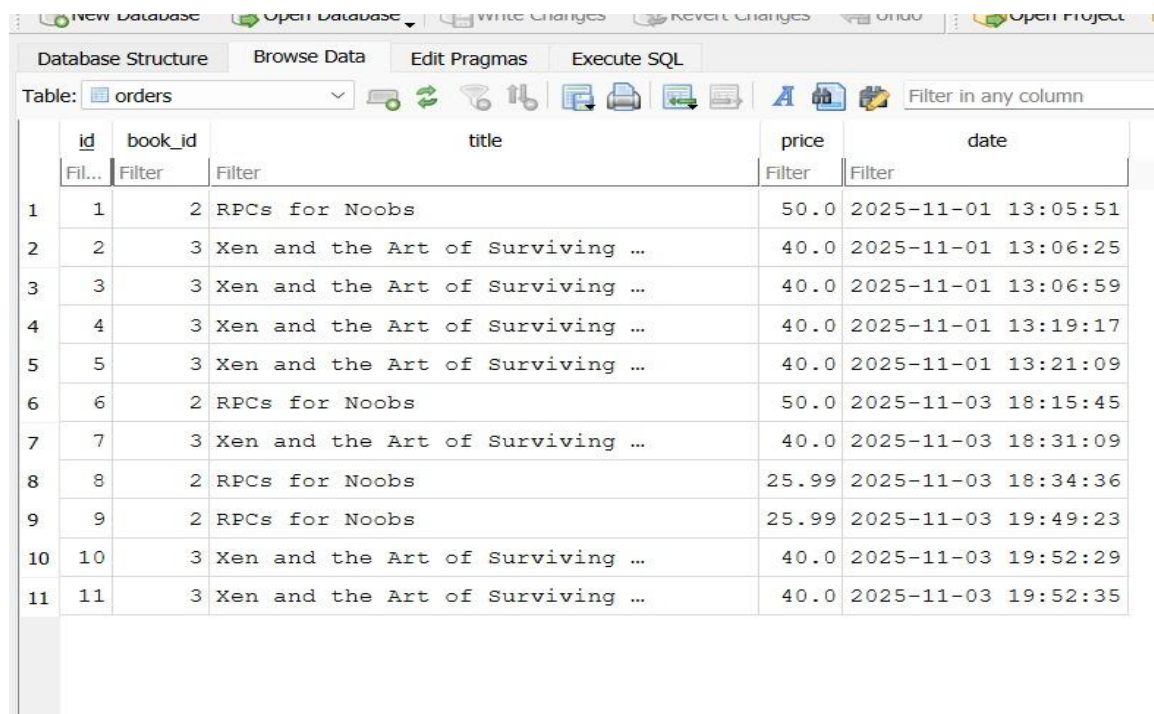
```
1 {
2   "status": "ok",
3   "message": "bought book Xen and the Art of Surviving Undergraduate School",
4   "order_id": 11
5 }
```

The quantity decrease by 1 :



	id	title	topic	price	quantity
1	1	How to get a good grade in DOS in 40...	distributed	30.0	2
2	2	RPCs for Noobs	distributed	20.0	1
3	3	Xen and the Art of Surviving ...	undergrad	40.0	0
4	4	Cooking for the Impatient Undergrad	undergrad	35.0	5

And this is the order table that contains all the order :



	id	book_id	title	price	date
1	1	2	RPCs for Noobs	50.0	2025-11-01 13:05:51
2	2	3	Xen and the Art of Surviving ...	40.0	2025-11-01 13:06:25
3	3	3	Xen and the Art of Surviving ...	40.0	2025-11-01 13:06:59
4	4	3	Xen and the Art of Surviving ...	40.0	2025-11-01 13:19:17
5	5	3	Xen and the Art of Surviving ...	40.0	2025-11-01 13:21:09
6	6	2	RPCs for Noobs	50.0	2025-11-03 18:15:45
7	7	3	Xen and the Art of Surviving ...	40.0	2025-11-03 18:31:09
8	8	2	RPCs for Noobs	25.99	2025-11-03 18:34:36
9	9	2	RPCs for Noobs	25.99	2025-11-03 19:49:23
10	10	3	Xen and the Art of Surviving ...	40.0	2025-11-03 19:52:29
11	11	3	Xen and the Art of Surviving ...	40.0	2025-11-03 19:52:35

Note : When the user want to purcash book and the quantity for the book = 0 the user can't do this and the message " Item out of stock " appear for him

Conclusion

This project deepened our understanding of microservices, inter-process communication, and container orchestration. It also provided hands-on experience with backend design and API testing.