

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

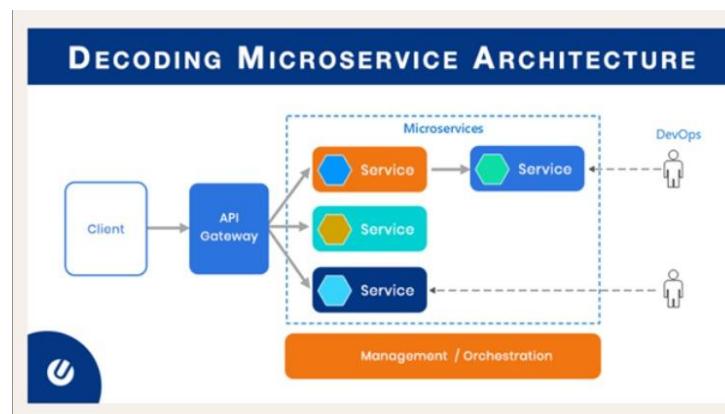
Docker → Containerization

Postman → Testing APIs

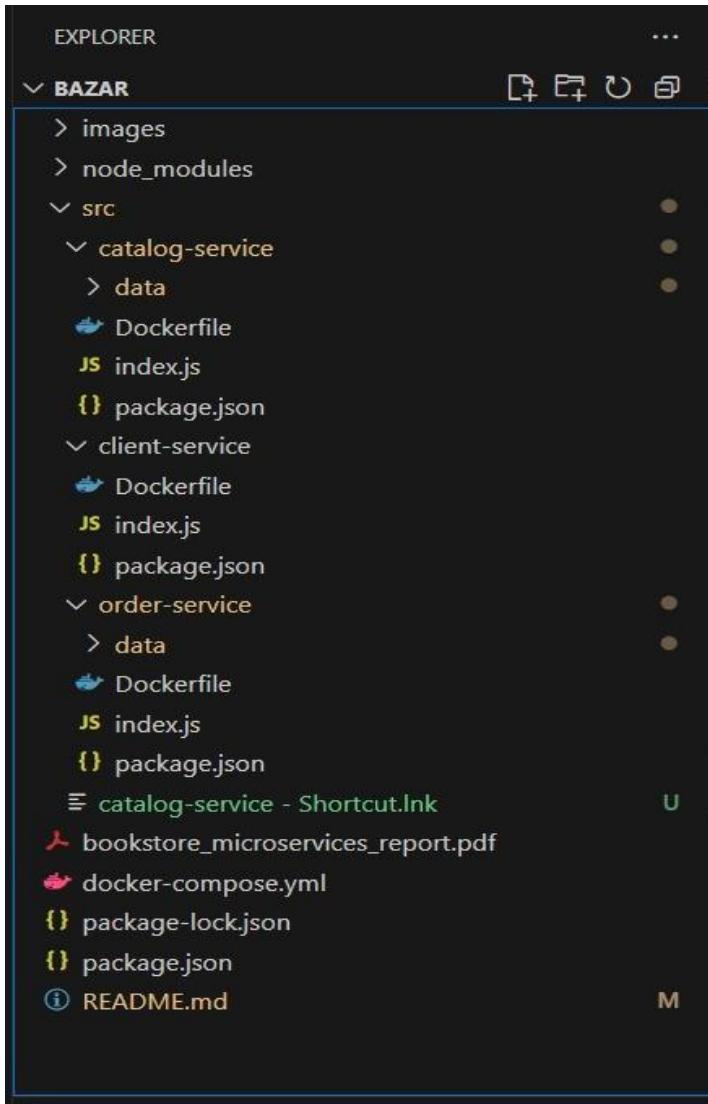
Git/GitHub → Version control

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

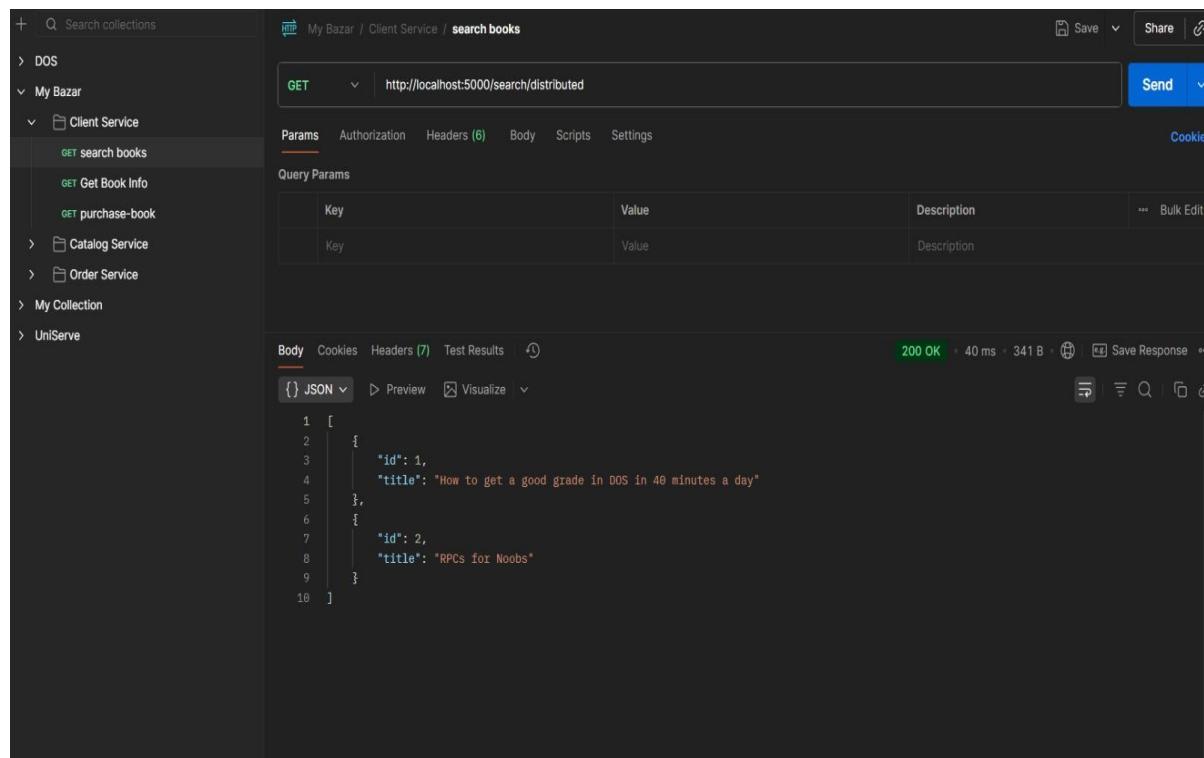
src/catalog-service : Manages book data: search, details, and catalog browsing

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 the Postman application interface. On the left, there's a sidebar with collections like DOS, My Bazar, Client Service, Catalog Service, Order Service, My Collection, and UniServe. Under Client Service, the 'search books' endpoint is selected. The main area shows a GET request to 'http://localhost:5001/search/distributed'. The 'Params' tab is active, showing a 'Query Params' table with a single row: 'Key' (Topic) and 'Value' (distributed). Below the table, there are tabs for Body, Cookies, Headers, Test Results, and a preview section. The preview section displays a JSON response with two items:

```
1 [  
2   {  
3     "id": 1,  
4     "title": "How to get a good grade in DOS in 40 minutes a day"  
5   },  
6   {  
7     "id": 2,  
8     "title": "RPCs for Noobs"  
9   }  
10 ]
```

GET /info/:id → Get book details

The screenshot shows the My Bazar API Client interface. The URL in the address bar is `http://localhost:5000/info/2`. The response status is **200 OK** with a response time of 18 ms and a body size of 288 B. The response body is a JSON object:

```
1 {  
2   "title": "RPCs for Noobs",  
3   "quantity": 2,  
4   "price": 25.99  
5 }
```

POST /reserve/:id → Reserve a book

The screenshot shows the My Bazar API Client interface. The URL in the address bar is `http://localhost:5001/reserve/1`. The response status is **200 OK** with a response time of 42 ms and a body size of 308 B. The response body is a JSON object:

```
1 {  
2   "title": "How to get a good grade in DOS in 40 minutes a day",  
3   "price": 30  
4 }
```

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

The screenshot shows the Postman interface with the following details:

- Method:** PUT
- URL:** http://localhost:5001/update/2
- Body:** Raw JSON (selected)

```
1 {  
2   "price": 20  
3 }
```
- Response Status:** 200 OK
- Response Time:** 40 ms
- Response Size:** 274 B
- Response Content:**

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

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

POST /purchase/:id → Purchase and store order

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:5000/purchase/2
- Params:** Key (selected)
- Response Status:** 200 OK
- Response Time:** 85 ms
- Response Size:** 302 B
- Response Content:**

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

GET /health → Health check

The screenshot shows a Postman interface for a 'Catalog Service' at 'http://localhost:5001/health'. The 'Params' tab is selected. The response status is 200 OK, with a response time of 8 ms and a body size of 245 B. The response body is JSON: { "ok": true }.

If the user want to purcash a book :

The screenshot shows a Postman interface for an 'Order Service' at 'http://localhost:5002/purchase/3'. The 'Params' tab is selected. The response status is 200 OK, with a response time of 74 ms and a body size of 339 B. The response body is JSON: { "status": "ok", "message": "bought book Xen and the Art of Surviving Undergraduate School", "order_id": 11 }.

The quantity decrease by 1 :

A screenshot of a SQLite database browser interface. The top menu bar includes 'Database Structure', 'Browse Data', 'Edit Pragmas', and 'Execute SQL'. Below the menu, there are tabs for 'books' and 'books'. The main area shows a table titled 'books' with columns: id, title, topic, price, and quantity. The data rows are:

	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 :

A screenshot of a SQLite database browser interface. The top menu bar includes 'Database Structure', 'Browse Data', 'Edit Pragmas', and 'Execute SQL'. Below the menu, there are tabs for 'orders' and 'orders'. The main area shows a table titled 'orders' with columns: id, book_id, title, price, and date. The data rows are:

	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.