An-Najah National University
Faculty of Engineering and IT

جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا
المعلومات

Computer Engineering Department
DOS
Project - Part 1

Instructor Name: Dr.Samer Alarandi

Students Names: Adam Abahri (12113269), Ameed Salameh (12111988)

Academic Year: First 2025-2026

An-Najah National University
Faculty of Engineering and IT

جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا
المعلومات

1. Abstract

This project develops a distributed online bookstore using microservices architecture in order to demonstrate some key ideas in DOS, namely inter-service communication, concurrency management, data consistency, and containerized deployment

The system constitutes three different Node.js services: Client, Catalog, and Order. These services communicate with each other over REST APIs. The system uses CSV files for data permanence and a mutex-based locking technique in order to reduce race conditions and ensure synchronization. Each service exposes REST endpoints by using Express.js for smooth integration and operation, and deployment is coordinated with Docker Compose.

2. Project Objectives

Among the key objectives of the project are:

1.Design a Microservices Architecture: Decompose a monolithic bookstore application into a set of loosely coupled services

2.Inter-Service Communication: For reliable interactions between services, implement synchronous REST/HTTP queries

3.Distributed Data Management: In managing shared resources, consistency in data must be maintained across services

4.Establish Concurrency Control: Employ mutex locks to prevent data races due to concurrent updates

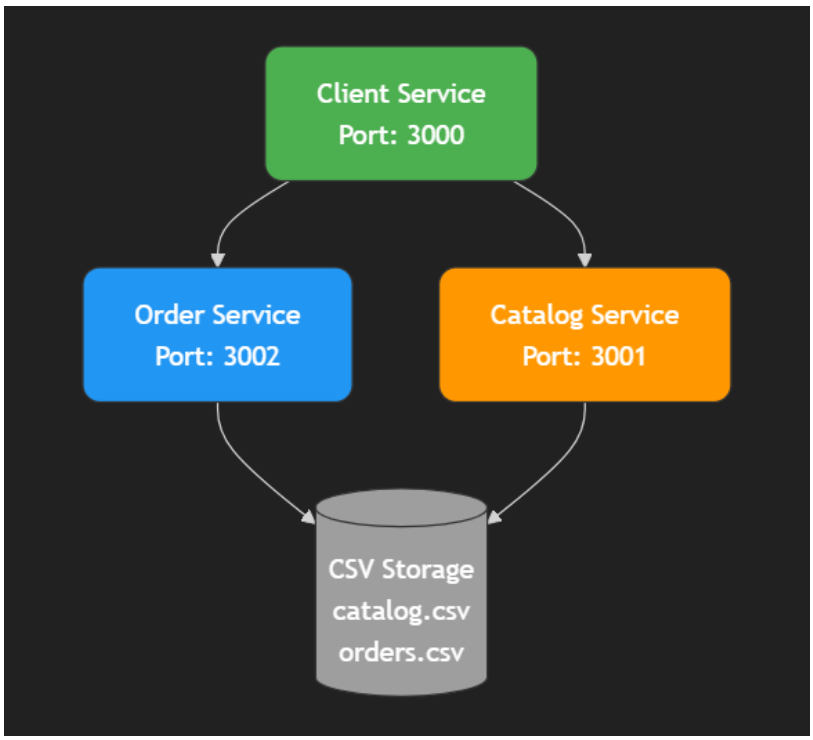
5.Implement Containerized Services: Use Docker and Docker Compose for effective, repeatable deployments

6. Ensure the Reliability of the System: For enhancing robustness, add error handling, health checks, and graceful failure management.

3. System Architecture

3.1 Architecture Overview

The system adopts a three-tier microservices architecture, as depicted below:



3.2 Service Responsibilities

A. Catalog Service (Port 3001)

Role: Product Inventory Manager **Responsibilities**:

- Store and update the product catalog.
- Handle search and product detail queries.
- Validate and update inventory levels.
- Enforce mutex-based concurrency control.
- Maintain product records in CSV format.

Product Model:

Field	Description
id	Unique identifier
title	Book title
topic	Subject category
quantity	Available stock
price	Unit price

B. Order Service (Port 3002)

Role: Transaction Processor **Responsibilities**:

- Process purchase requests.
- Communicate with the Catalog Service for inventory validation.
- Perform consistency checks during transactions.
- Record completed orders in orders.csv.
- Handle failures atomically to prevent inconsistencies.

Order Model:

Field	Description
customer_name	Customer's name
product_id	Purchased product ID
quantity	Quantity ordered
total_price	Total transaction amount
timestamp	Date and time of order

C. Client Service (Port 3000)

Role: API Gateway and Orchestrator **Responsibilities**:

- Provide a unified REST interface for external clients.
- Route requests to the Catalog and Order services.
- Aggregate responses and handle errors.
- Implement the API Gateway pattern for simplified access.

3.3 Communication Architecture

Purchase Workflow:

1. Client submits request to Client Service, which forwards to Order Service.
2. Order Service queries Catalog Service to validate and update inventory.
3. Order Service records the order in CSV upon successful update.
4. Response is propagated back to the client.

API Protocol:

- **Protocol**: HTTP
- **Architecture**: REST

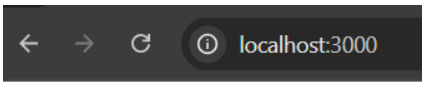
- **Format**: JSON
- **Implementation**: Node.js HTTP module

4. Detailed Implementation

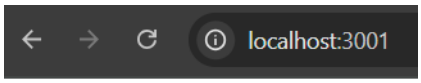
4.1 Catalog Service

Key Endpoints:

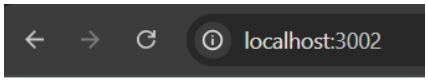
- GET /: Health check endpoint.
- GET /search?topic=: Search products by topic.
- GET /product/:id: Retrieve product details by ID.
- POST /update: Update product inventory.



client_service is running

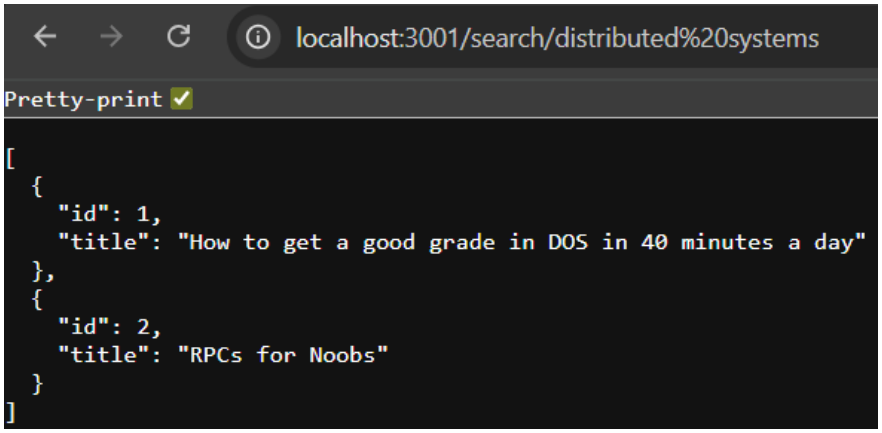


catalog_service is running

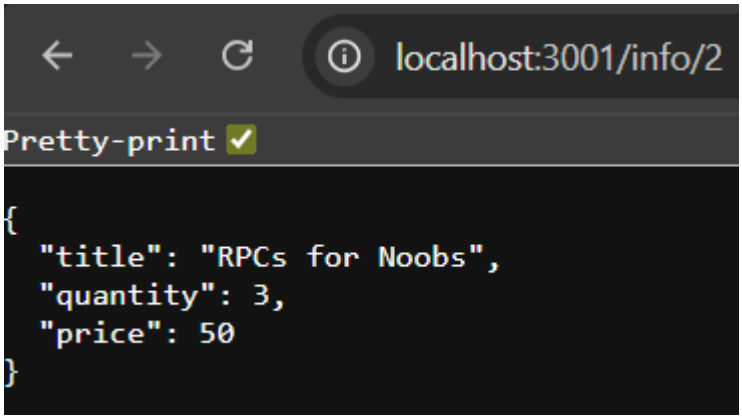


order_service is running

First Requierd REST API:

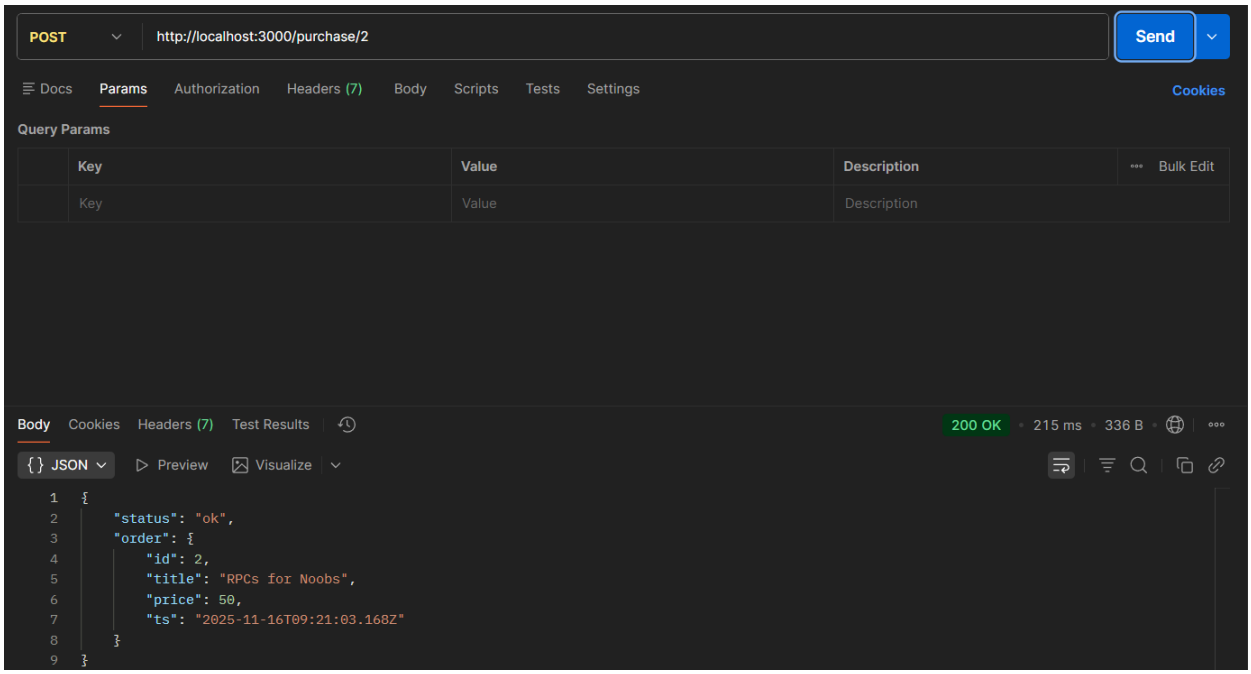


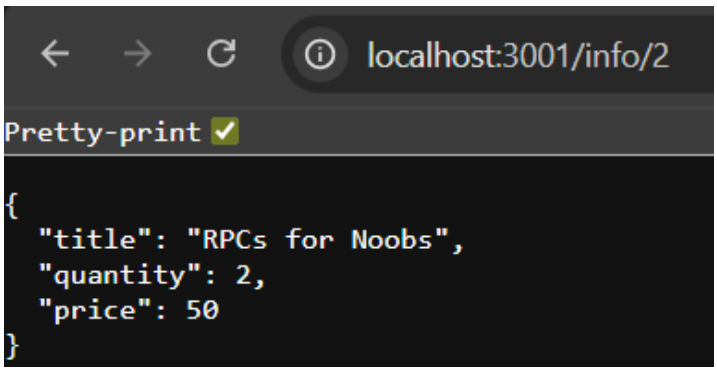
Second Requierd REST API:



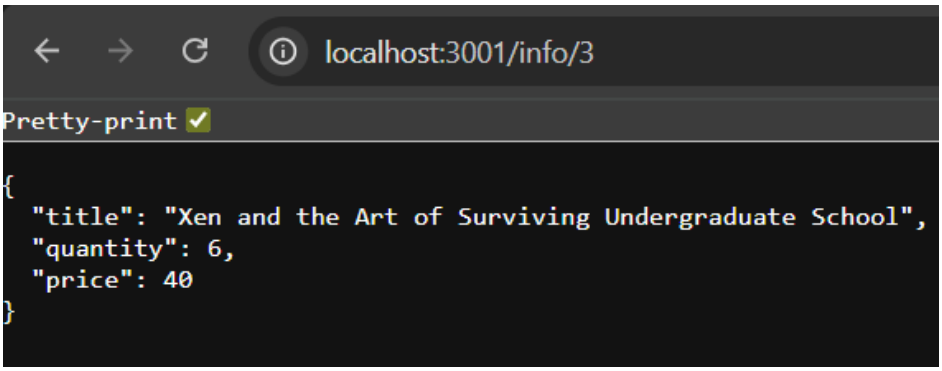
Third Requierd REST API:

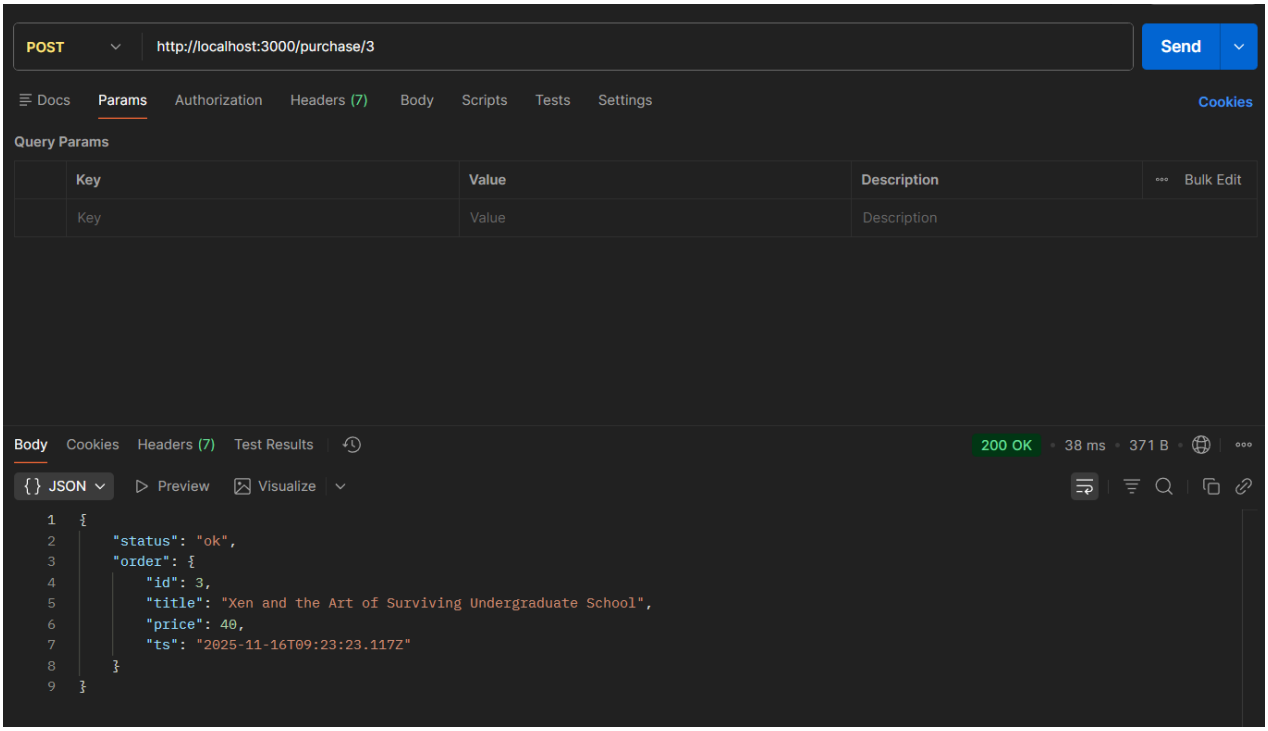
Last image (at Second API) is the quantity before running the pusrchase API, that's the result after that for the same catalog id:

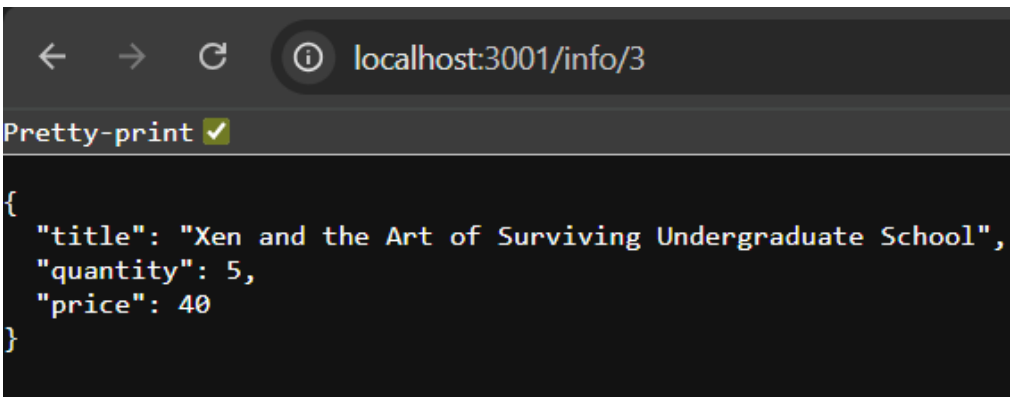




For id=3 :


localhost:3001/info/3
Pretty-print
{
"title": "Xen and the Art of Surviving Undergraduate School",
"quantity": 6,
"price": 40
}


POST
http://localhost:3000/purchase/3
Send
Docs
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
38 ms
371 B
{} JSON
Preview
Visualize
{
"status": "ok",
"order": {
"id": 3,
"title": "Xen and the Art of Surviving Undergraduate School",
"price": 40,
"ts": "2025-11-16T09:23:23.117Z"
}
}


localhost:3001/info/3
Pretty-print
{
"title": "Xen and the Art of Surviving Undergraduate School",
"quantity": 5,
"price": 40
}

```
catalog  | catalog_service listening on port 3001
order    | order_service listening on port 3002
client   | client_service listening on port 3000
client   | Info: { title: 'RPCs for Noobs', quantity: 5, price: 50 }
client   | Info: {
client   |   title: 'Xen and the Art of Surviving Undergraduate School',
client   |   quantity: 8,
client   |   price: 40
client   | }
order    | bought book RPCs for Noobs
client   | Purchase response: {
client   |   status: 'ok',
client   |   order: {
client   |     id: 2,
client   |     title: 'RPCs for Noobs',
client   |     price: 50,
client   |     ts: '2025-11-15T12:04:32.859Z'
client   |   }
client   | }
client   | Info: { title: 'RPCs for Noobs', quantity: 4, price: 50 }
client   | Info: { title: 'RPCs for Noobs', quantity: 4, price: 50 }
order    | bought book RPCs for Noobs

client   | Purchase response: {




client   |   status: 'ok',
client   |   order: {
client   |     id: 2,
client   |     title: 'RPCs for Noobs',
client   |     price: 50,
client   |     ts: '2025-11-15T12:04:57.454Z'
client   |   }
client   | }
client   | Info: { title: 'RPCs for Noobs', quantity: 3, price: 50 }
order    | bought book Xen and the Art of Surviving Undergraduate School
```

```
client   | Purchase response: {
client   |   status: 'ok',
client   |   order: {
client   |     id: 3,
client   |     title: 'Xen and the Art of Surviving Undergraduate School',
client   |     price: 40,
client   |     ts: '2025-11-15T12:05:16.300Z'
client   |   }
client   | }
client   | Info: {
client   |   title: 'Xen and the Art of Surviving Undergraduate School',
client   |   quantity: 7,
client   |   price: 40
client   | }
order    | bought book Xen and the Art of Surviving Undergraduate School
client   | Purchase response: {
client   |   status: 'ok',
client   |   order: {
client   |     id: 3,
client   |     title: 'Xen and the Art of Surviving Undergraduate School',
client   |     price: 40,
client   |     ts: '2025-11-15T12:05:34.156Z'
client   |   }
client   | }
client   |     ts: '2025-11-15T12:05:34.156Z'
client   |   }
client   | }
client   | Info: {
client   |   title: 'Xen and the Art of Surviving Undergraduate School',
client   |   quantity: 6,
client   |   price: 40
client   | }
order    | bought book RPCs for Noobs
client   |   title: 'Xen and the Art of Surviving Undergraduate School',
client   |   quantity: 6,
client   |   price: 40
client   | }
order    | bought book RPCs for Noobs

client   | Purchase response: {
client   |   status: 'ok',
client   |   order: {
client   |     id: 2,
client   |     title: 'RPCs for Noobs',
client   |     price: 50,
client   |     ts: '2025-11-16T09:21:03.168Z'
client   |   }
client   | }
order    | bought book Xen and the Art of Surviving Undergraduate School
client   | Purchase response: {
client   |   status: 'ok',
client   |   order: {
client   |     id: 3,
client   |     title: 'Xen and the Art of Surviving Undergraduate School',
client   |     price: 40,
client   |     ts: '2025-11-16T09:23:23.117Z'
client   |   }
client   | }
```

Mutex Lock Implementation: A per-product mutex ensures serialized access during updates, preventing concurrent modifications:

```
let locks = {};

async function updateWithLock(productId, newQty, newPrice) {
    while (locks[productId]) await wait(10);
    locks[productId] = true;

    try {
        let product = readFromCSV(productId);
        product.quantity = newQty;
        product.price = newPrice;
        writeToCSV(product);
        return { success: true };
    } finally {
        delete locks[productId];
    }
}
```

4.2 Order Service

Purchase Logic:

```
async function processPurchase(customer, productId, qty) {
    const product = await catalog.getProduct(productId);
    if (!product) return fail("Product not found");

    if (product.quantity < qty)
        return fail(`Available: ${product.quantity}`);

    const total = product.price * qty;

    const update = await catalog.update({
        id: productId,
        quantity: product.quantity - qty,
        price: product.price
    });

    if (!update.success) return fail("Inventory update failed");

    const orderId = generateOrderId();
    recordOrder({ orderId, customer, productId, qty, total, timestamp: now() });

    return success(orderId, total, product.title);
}
```

Guarantees:

- Inventory updates and order recordings are atomic.
- No partial transactions; failures roll back changes.

4.3 Client Service

Implements the API Gateway pattern, exposing unified endpoints such as:

- /search
- /product/:id
- /purchase
  (which shown in 4.1)

Requests are proxied via HTTP to the appropriate backend services.

4.4 CSV-Based Persistence

catalog.csv Example:

catalog.csv

```
id,title,topic,quantity,price
1,How to get a good grade in DOS in 40 minutes a day,distributed systems,5,30
2,RPCs for Noobs,distributed systems,5,50
3,Xen and the Art of Surviving Undergraduate School,undergraduate school,8,40
4,Cooking for the Impatient Undergrad,undergraduate school,8,35
```

orders.csv Example:

```
customer_name,product_id,quantity,total_price,timestamp
John Doe,1,2,60,2024-11-15T10:30:45Z
```

Advantages: Simplicity and portability. **Limitations**: Performance degradation under high concurrency; lacks full ACID compliance.

5. Configuration and Deployment

5.1 Multi-Layer Configuration

Configuration follows a priority order:

1. Environment variables.
2. config.json file.
3. Hardcoded defaults.

5.2 Dockerfiles

Each service uses a standardized Dockerfile:

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

S

5.3 Docker Compose Structure

```yaml
version: '3'
services:
  catalog:
    build: ./catalog_service
    ports: ["3001:3001"]

  order:
    build: ./order_service
    depends_on: [catalog]
    ports: ["3002:3002"]
    environment:
      - CATALOG_URL=http://catalog:3001

  client:
    build: ./client_service
    depends_on: [catalog, order]
    ports: ["3000:3000"]
```

Services communicate via Docker's internal DNS (e.g., catalog, order, client).

6. Conclusion

This project successfully provides a functional, microservices-based distributed online bookstore, implementing all the key DOS principles: mutex-driven concurrency control, coordinated distributed transactions, RESTful inter-service communication, data persistence, and containerized deployment. Its modular design encourages scalability and accurately reflects production-grade distributed systems, as well as a foundation for further improvements.

GitHub Repo. Link : https://github.com/Ameed-salameh/project_Distributed-and-Operating-Systems_part1