DOS Project Part#1

We used node js and npm must downloaded to use node, and we used nodemon package for Hot-Reload.

Docker must be downloaded, to run our containers:

docker-compose up -d –build → to build all containers
docker-compose down → to stop all containers

If we want to display the running Containers currently, we type:

docker ps

To Interact with our Container from Terminal:

docker exec -it container_name bash

Sqlite3 database must be downloaded, to dealing with Database:

sqlite> .tables;
sqlite> SELECT * FROM tablename;

We create 3 Services, 2 for Backend servers: Catalog & Order, 1 for Frontend Client Service. we create Dockerfile, to create our containers.

- catalog-service
  it contains 3 Requests: search by book topic, info by item number and purchase book

- order-service
  it contains purchase request

- localhost:3005/search/:bookTopic
  localhost:3005/info/:itemNumber
  localhost:3006/purchase

SQL ▾ 1 / 1 1 - 4 of 4

id	bookTopic	numberOfItems	bookCost	bookTitle
1	Distributed System	48	200	How to get a good grade in DOS in 40 minutes a day
2	Distributed System	44	150	RPCs for Noobs.
3	Undergraduate School	32	100	Xen and the Art of Surviving Undergraduate School
4	Undergraduate School	26	90	Cooking for the Impatient Undergrad

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs
Usage: CLI [options] [command]

CLI for DOS Project

Options:
  -V, --version                          output the version number
  -h, --help                             display help for command

Commands:
  search-book-title|search               search about specific book using book topic
  info-book-item-number|info             info about specific book using item number
  purchase-book-by-item-number|purchase  purchase specific book using item number
  help [command]                         display help for command
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service>
```

This is the table of books in Database and its columns. The command used for each service

node index.mjs purchase
node index.mjs info
node index.mjs search

Now, Let's test each part of our code:

search(topic) - which allows the user to specify a topic and returns all entries belonging to that category (a title and an item number are displayed for each match.

currently all books belong to one of two topics: distributed systems and undergraduate school.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs search
? Enter topic to get all books belonging to that:  Distributed System
Response: {
  items: [
    {
      id: 1,
      bookTopic: 'Distributed System',
      numberOfItems: 48,
      bookCost: 200,
      bookTitle: 'How to get a good grade in DOS in 40 minutes a day'
    },
    {
      id: 2,
      bookTopic: 'Distributed System',
      numberOfItems: 44,
      bookCost: 150,
      bookTitle: 'RPCs for Noobs.'
    }
  ]
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs search
? Enter topic to get all books belonging to that:   Undergraduate School
Response: {
  items: [
    {
      id: 3,
      bookTopic: 'Undergraduate School',
      numberOfItems: 32,
      bookCost: 100,
      bookTitle: 'Xen and the Art of Surviving Undergraduate School'
    },
    {
      id: 4,
      bookTopic: 'Undergraduate School',
      numberOfItems: 26,
      bookCost: 90,
      bookTitle: 'Cooking for the Impatient Undergrad'
    }
  ]
}
```

info(item_number) - which allows an item number to be specified and returns details such as number of items in stock and cost.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs info
? Enter item number to get all details about it:  1
Response: { item: [ { id: 1, numberOfItems: 48, bookCost: 200 } ] }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs info
? Enter item number to get all details about it:  3
Response: { item: [ { id: 3, numberOfItems: 32, bookCost: 100 } ] }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs info
? Enter item number to get all details about it:  2
Response: { item: [ { id: 2, numberOfItems: 44, bookCost: 150 } ] }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service>
```

The catalog server supports two operations: query and update. Two types of queries are supported: query-by-subject and query-by-item. In the first case, a topic is specified and the server returns all matching entries. In the second case, an item is specified and all relevant details are returned. The update operation allows the cost of an item to be updated or the number of items in stock to be increased or decreased.

The order server supports a single operation: purchase(item_number). Upon receiving a purchase request, the order server must first verify that the item is in stock by querying the catalog server and then decrement the number of items in stock by one. The purchase request can fail if the item is out of stock.

purchase(item_number) - which specifies an item number for purchase.

recieve item number & order cost from CLI (frontend) then send them to catalog service using axios post request.

When a book is purchased, the data base changes so that a book is deducted from the stock.

SQL ▾ < 1 / 1 > 1 - 4 of 4

id	bookTopic	numberOfItems	bookCost	bookTitle
1	Distributed System	46	200	How to get a good grade in DOS in 40 minutes a day
2	Distributed System	44	150	RPCs for Noobs.
3	Undergraduate School	32	100	Xen and the Art of Surviving Undergraduate School
4	Undergraduate School	26	90	Cooking for the Impatient Undergrad

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

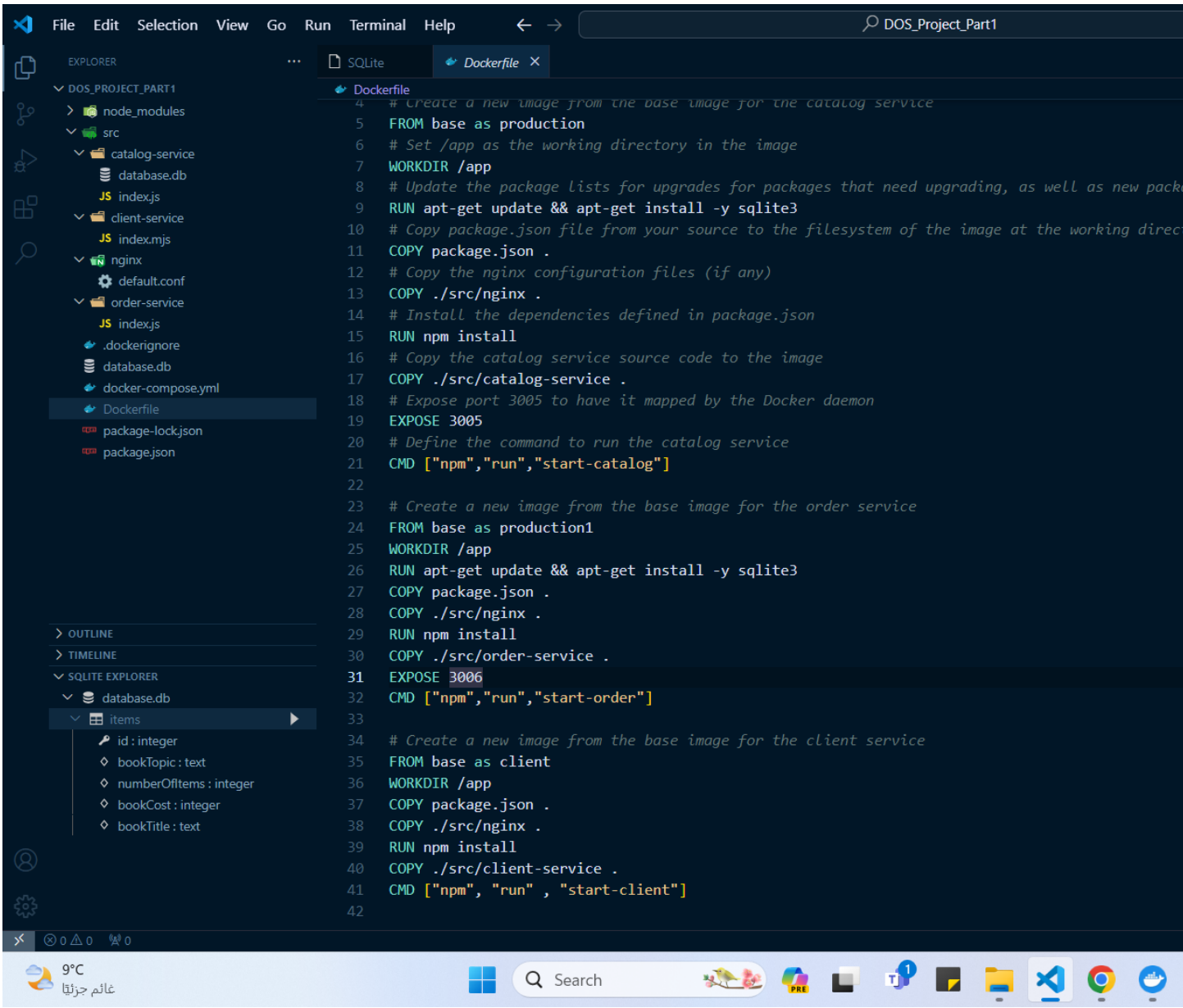
```
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs purchase
? Enter item number to purchase it:  1
? Enter amount of money to pay:  200
Response: { message: 'Send Request To Catalog Server' }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs purchase
? Enter item number to purchase it:  1
? Enter amount of money to pay:  200
Response: { message: 'Send Request To Catalog Server' }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service>
```

SQL ▾ < 1 / 1 > 1 - 4 of 4

id	bookTopic	numberOfItems	bookCost	bookTitle
1	Distributed System	46	200	How to get a good grade in DOS in 40 minutes a day
2	Distributed System	43	150	RPCs for Noobs.
3	Undergraduate School	32	100	Xen and the Art of Surviving Undergraduate School
4	Undergraduate School	26	90	Cooking for the Impatient Undergrad

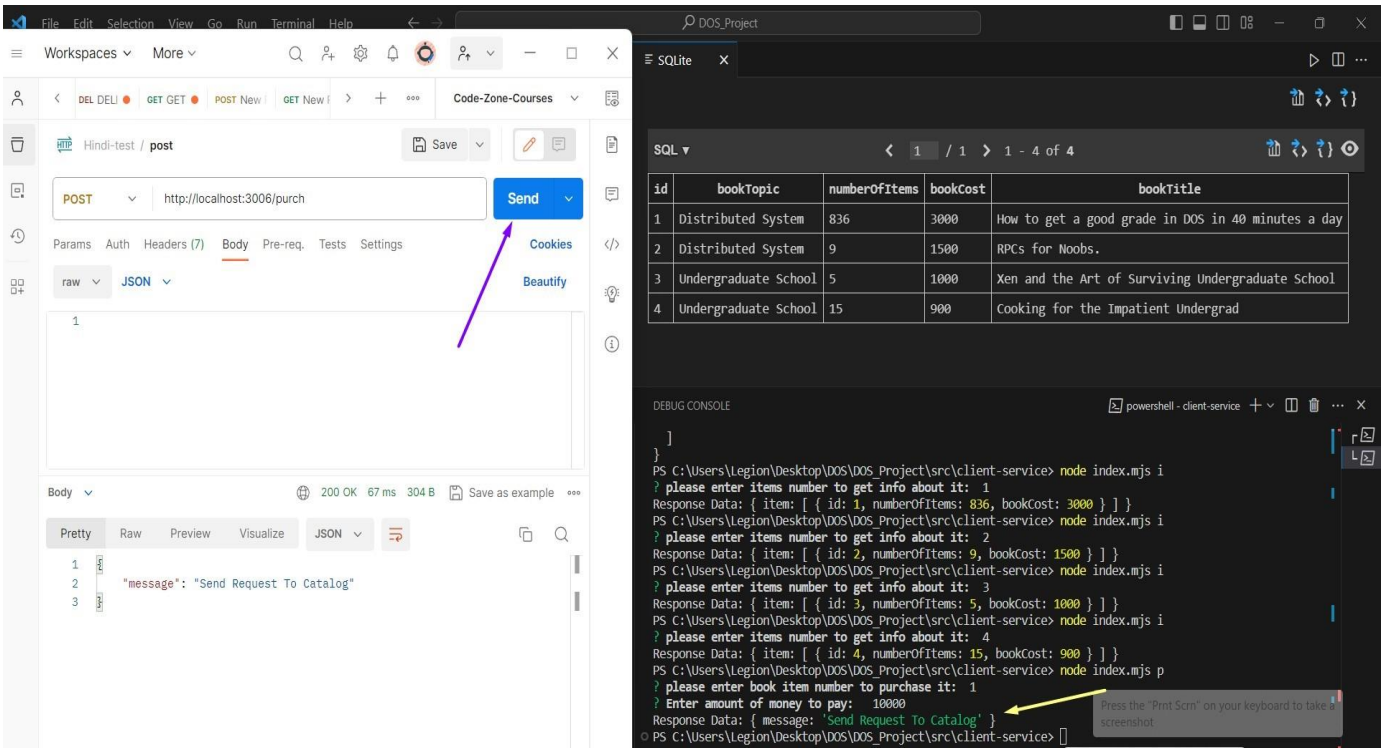
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs purchase
? Enter item number to purchase it:  1
? Enter amount of money to pay:  200
Response: { message: 'Send Request To Catalog Server' }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs purchase
? Enter item number to purchase it:  1
? Enter amount of money to pay:  200
Response: { message: 'Send Request To Catalog Server' }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service> node index.mjs purchase
? Enter item number to purchase it:  2
? Enter amount of money to pay:  150
Response: { message: 'Send Request To Catalog Server' }
PS C:\Users\Engineer\Desktop\DOS_Project_Part1\src\client-service>
```



This Dockerfile contain 3 part for each service, Docker builds images automatically by reading the instructions from a Dockerfile which is a text file that contains all commands, in order, needed to build a given image. A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.

use a Dockerfile to define your app's environment so it can be reproduced anywhere. Define the services that make up your app in docker-compose. yml so you can run them together in an isolated environment. Use docker compose up and Docker compose command to start and run your entire app.




```
// Define a POST endpoint for making a purchase
app.post("/purch", async (req, res) => {
  // Extract order details from the request body
  const order = {
    id: req.body.id, // Extract book ID
    orderCost: req.body.orderCost, // Extract order cost
  };

  try {
    // Make a POST request to the catalog server to process the order
    const response = await axios.post(
      `http://catalog-server:3005/order`, // URL of the catalog server
      order // Data to be sent in the request body
    );

    // Log the response from the catalog server
    console.log(response.data);

    // Send a response indicating that the request was successfully sent to the catalog server
    res.send({ message: "Send Request To Catalog Server" });
  } catch (err) {
    // Log any errors that occur during the request
    console.log(err);

    // Send an error response with status code 400 if an error occurs
    res.status(400).send({ error: err });
  }
});

// Start the server and listen on the specified port
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

The benefit of the Bazar.com project lies in its implementation of a lightweight, distributed online book store using microservices architecture and RESTful APIs.

In conclusion, the Bazar.com project demonstrates the design and implementation of a lightweight, distributed online book store using modern web development techniques. It emphasizes scalability, flexibility, efficiency, reliability, and ease of testing, while also promoting good software engineering practices such as version control, collaboration.

Ayman Emad Dwikat

11923734