Distributed and Operating Systems

Lab 1: Bazar.com: A Multi-tier Online Book Store

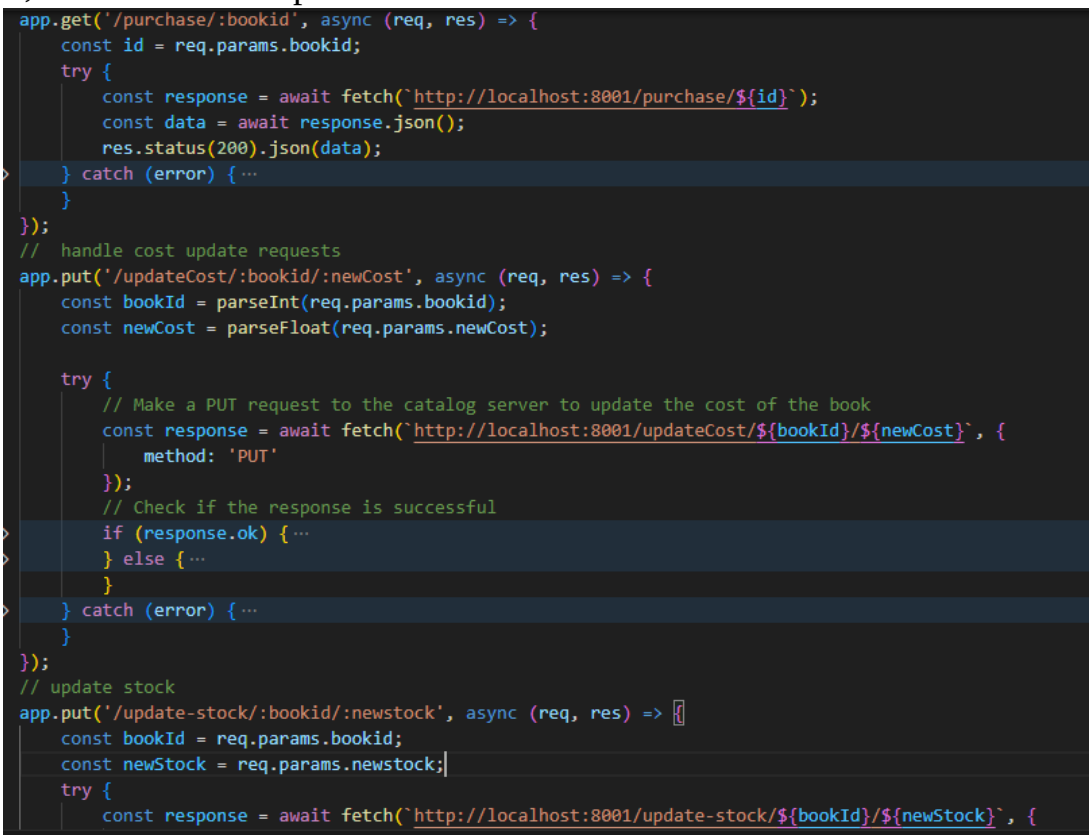
Names: Toqa Abdo & Alaa Yaish

In this project we use node js and VS code IDE/text file as a database.

- **Front end server:** it supports all search and info, is pass the query to catalog server and tack the resale from it, the server run at port 8000.

```
app.get('/search/:topic', async (req, res) => {
    const topic = req.params.topic;
    try {
        const response = await fetch(`http://localhost:8001/search/${topic}`);
        const data = await response.json();
        res.status(200).json(data);
    } catch (error) {
        console.error("Error processing search request:", error);
        res.status(500).json({ success: false, message: "Internal server error" });
    }
});
// // Route to search books by topic
app.get('/info/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const response = await fetch(`http://localhost:8001/info/${id}`);
        const data = await response.json();
        res.status(200).json(data);
    } catch (error) {
        console.error("Error processing search request:", error);
        res.status(500).json({ success: false, message: "Internal server error" });
    }
});
```

- **Order server:** The order server supports a purchase(item_number) and update, it passes the queries to the catalog server and take the output from it, the server run at port 8002.



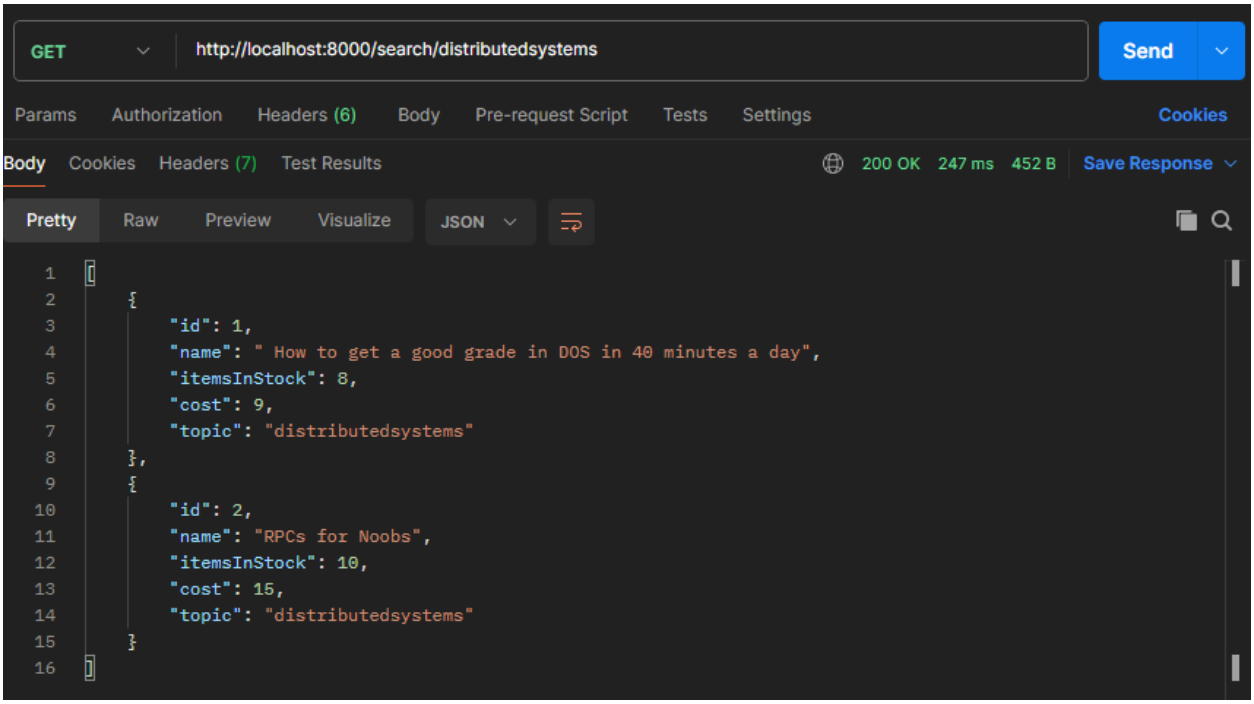
```
app.get('/purchase/:bookid', async (req, res) => {
    const id = req.params.bookid;
    try {
        const response = await fetch(`http://localhost:8001/purchase/${id}`);
        const data = await response.json();
        res.status(200).json(data);
    } catch (error) {···
    }
});
//  handle cost update requests
app.put('/updateCost/:bookid/:newCost', async (req, res) => {
    const bookId = parseInt(req.params.bookid);
    const newCost = parseFloat(req.params.newCost);

    try {
        // Make a PUT request to the catalog server to update the cost of the book
        const response = await fetch(`http://localhost:8001/updateCost/${bookId}/${newCost}`, {
            method: 'PUT'
        });
        // Check if the response is successful
        if (response.ok) {···
        } else {···
        }
    } catch (error) {···
    }
});
// update stock
app.put('/update-stock/:bookid/:newstock', async (req, res) => {
    const bookId = req.params.bookid;
    const newStock = req.params.newstock;
    try {
        const response = await fetch(`http://localhost:8001/update-stock/${bookId}/${newStock}`, {
```

- **Catalog server:** we implement all APIs in it, search, info, purchase and update the cost or stock of a book, the server run at port 8001.

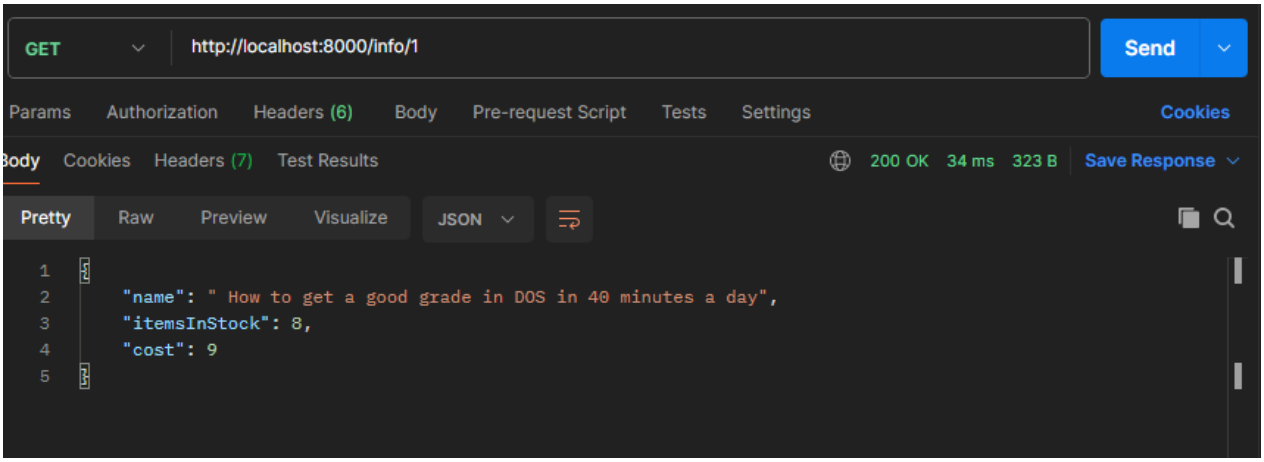
```
app.get('/search/:topic', async (req, res) => {···
});

app.get('/info/:id', async (req, res) => {···
});

// purchase
function readDatabase() {···
}

// Function to write to the database file
function writeDatabase(database) {···
}

// Route to handle purchase requests
app.get('/purchase/:bookid', async (req, res) => {···
});

// handle updating the cost of a book
app.put('/updateCost/:bookid/:newCost', (req, res) => {···
});


//  handle updating the number of items in stock
app.put('/update-stock/:bookid/:newstock', (req, res) => {···
});
```

Postman tests:

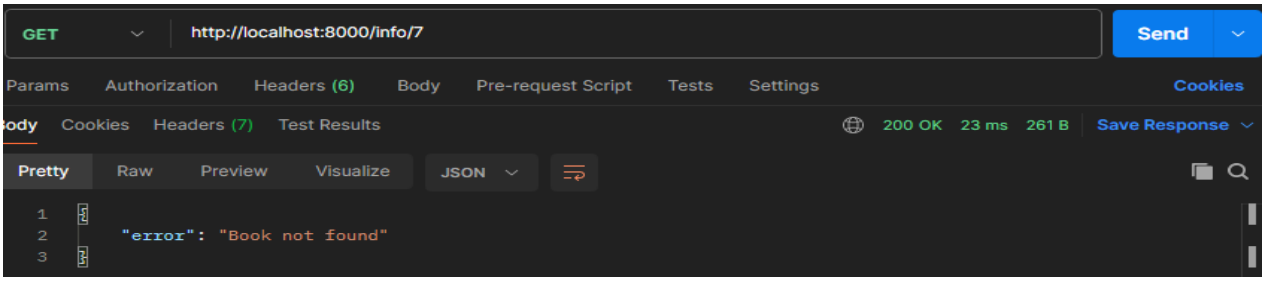
- Search (by the topic)



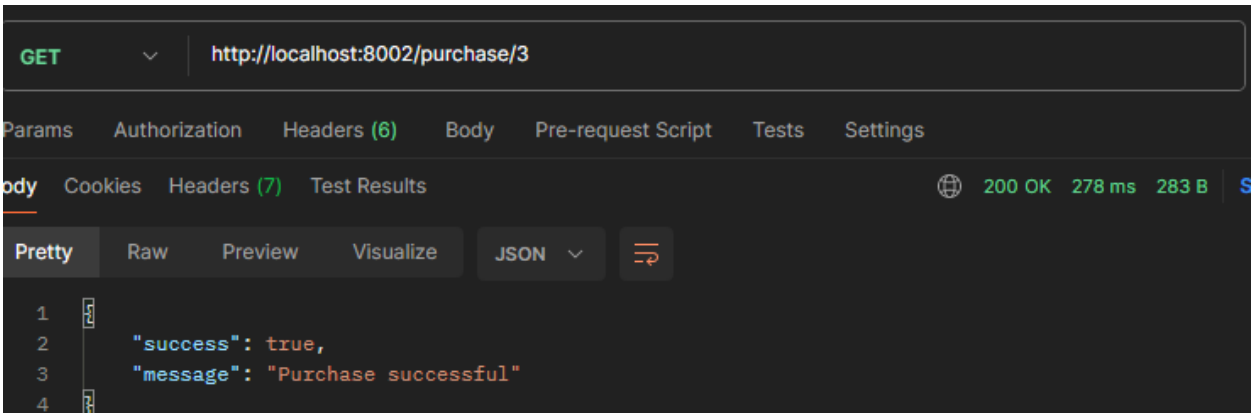
- Info (by id)



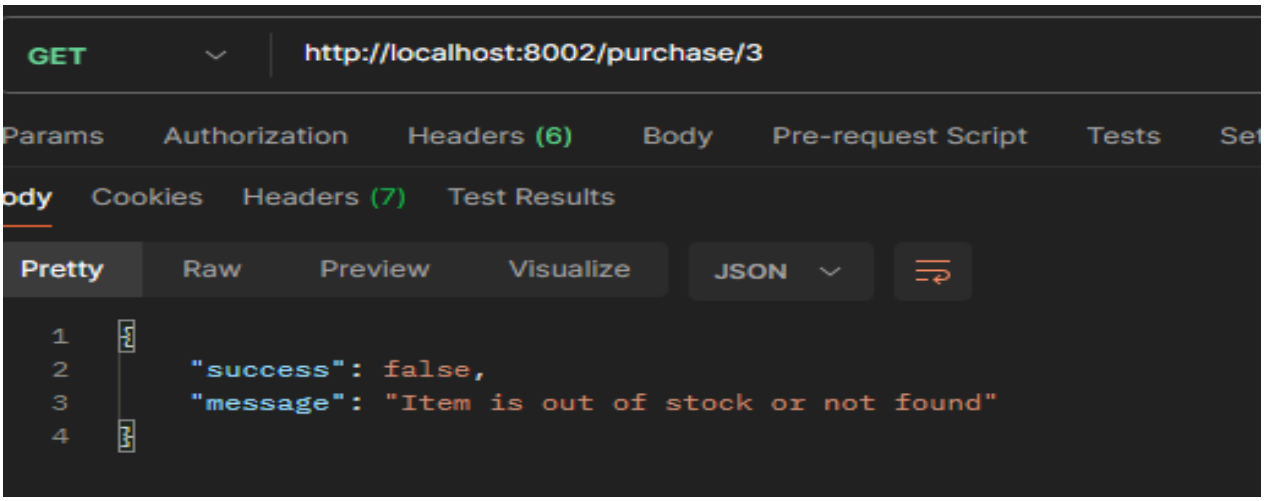
If the id not exists:



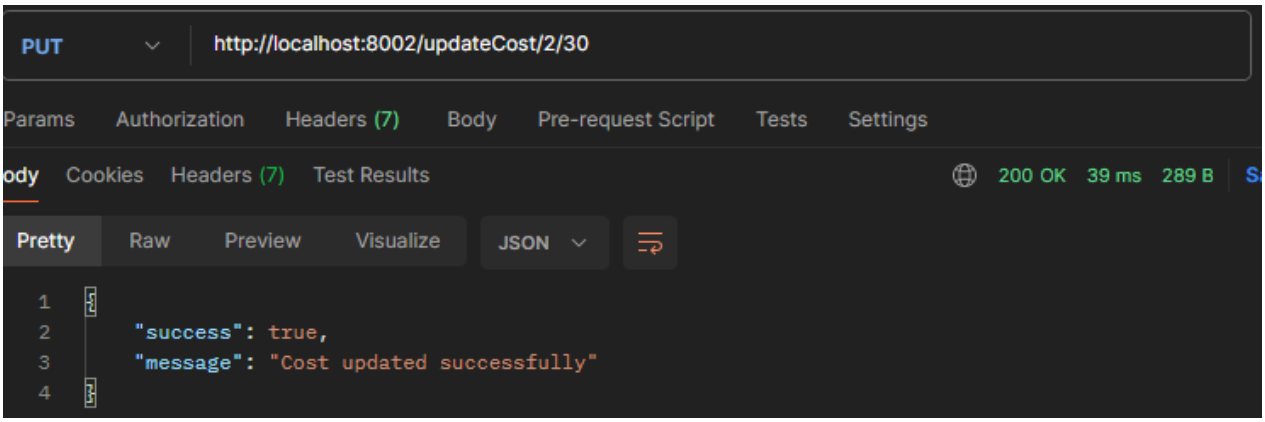
- Purchase



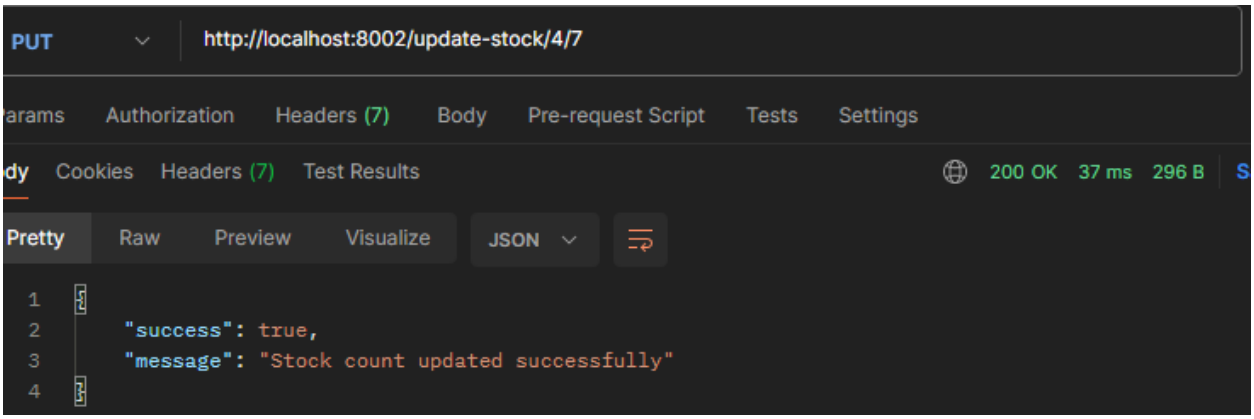
If the Purchase with not available (book not exists or the stock is 0)



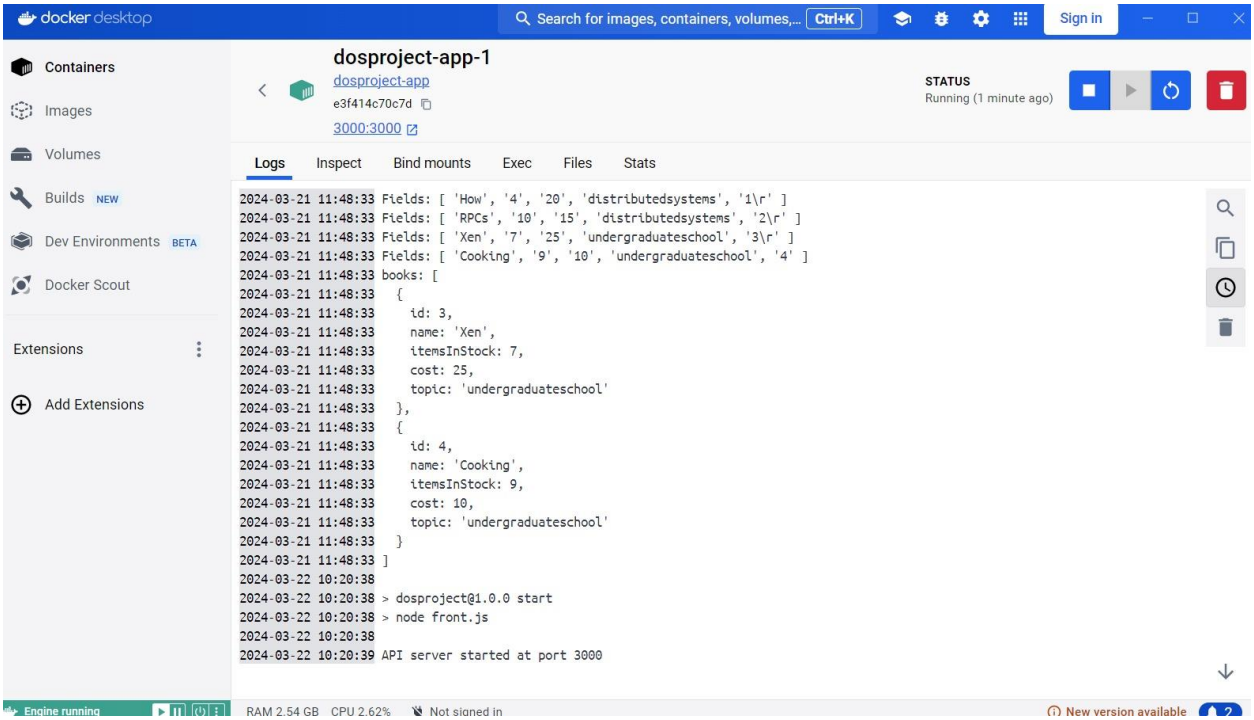
- Update cost:



- Update stock:



- Sample of docker output:



Thank You