CS 4003 - Problem Set 1: LionOrder Distributed Ordering System

Nabeel Mahmood, Joe Neal, & Joshua Tilley

Due Date: **March 10, 2025**

March 10, 2025

1 System Architecture

Docker Containers

Customer Container

The customer container handles customer-related actions like creating and retrieving customer details.

User Registration: Allows new customers to register by providing their personal details such

- First Name
- Last Name
- Username
- Email
- Password

Authentication (Login): This authenticates the email and password of returning customers to allow access to their account and place orders.

Order Container

The order container handles the creation, updating, and retrieval of orders.

- New order creation
- Updating order status and contents
- Retrieving order history for a customer

Inventory Container

The inventory container manages stock levels and product details.

- Retrieving inventory status
- Updating stock levels
- Adding new inventory items

Warehouse Container

The warehouse container manages the bulk storage and availability of products.

- Checking a product availability
- Updating the stock levels in the warehouse
- Deleting items from the warehouse

APIs per Container

Customer API

- **POST /customers/authenticate** — This authenticates the user by email and password.
- **POST /customers/register** — Registering a new user.
- **GET /customers/id** — This will retrieve user details by there unique ID.

Order API

- **POST /orders/insert** — Creates a new order.
- **GET /orders/orderID** — Retrieves order details by order ID.
- **GET /orders/customer/custID** — Retrieves all orders for a customer.
- **PUT /orders/update/orderID** — Updates an existing order.
- **DELETE /orders/delete/orderID** — Deletes an order.

Inventory API

- **GET /inventory/all** — Retrieves all inventory items.
- **POST /inventory/insert** — Inserts a new inventory item.
- **PUT /inventory/update/itemID** — Updates an inventory item.
- **DELETE /inventory/delete/itemID** — Deletes an inventory item.

Warehouse API

- **GET /warehouse/all** — Retrieves all items in the warehouse.
- **POST /warehouse/insert** — Inserts a new warehouse item.
- **PUT /warehouse/update/upc** — Updates warehouse item details.
- **DELETE /warehouse/delete/upc** — Deletes a warehouse item.

Data Entities

Customer Database

```
CREATE TABLE IF NOT EXISTS customers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    fname VARCHAR(100) NOT NULL,
    lname VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    user VARCHAR(100) UNIQUE NOT NULL,
    pw VARCHAR(255) NOT NULL
);
```

Order Database

```
CREATE TABLE IF NOT EXISTS orders (
    orderID INT AUTO_INCREMENT PRIMARY KEY,
    custID INT NOT NULL,
    orderStatus VARCHAR(50) NOT NULL,
    orderDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    orderContents TEXT NOT NULL
);
```

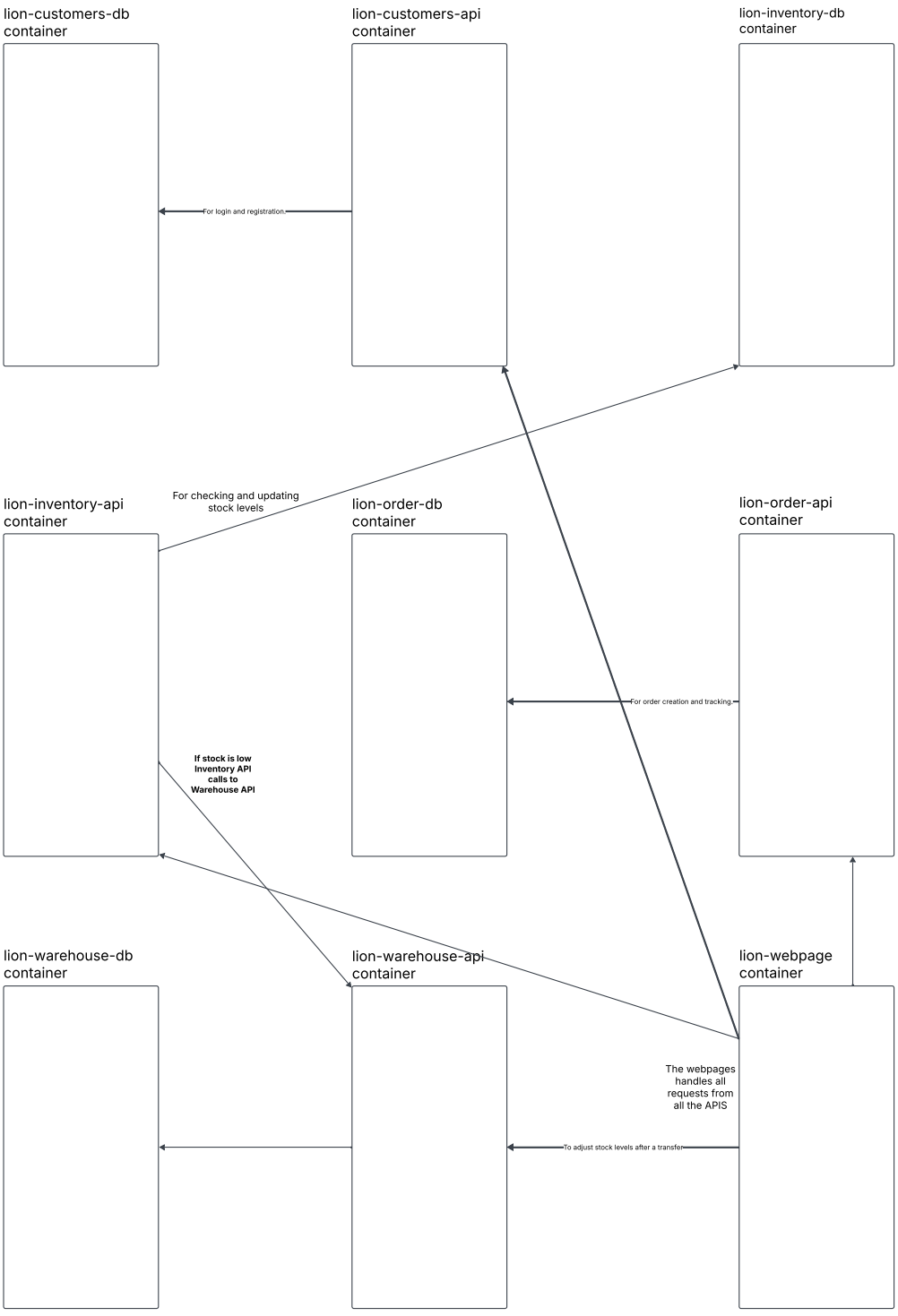
Inventory Database

```
CREATE TABLE IF NOT EXISTS inventory (
    itemID INT AUTO_INCREMENT PRIMARY KEY,
    itemName VARCHAR(100) NOT NULL,
    UPC VARCHAR(50) UNIQUE NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL
);
```

Warehouse Database

```
CREATE TABLE IF NOT EXISTS warehouse (
    upc VARCHAR(50) PRIMARY KEY,
    qty INT DEFAULT 0,
    availability BOOLEAN DEFAULT TRUE
);
```

System Architecture Diagram



2 APIs

Customer API

Authenticate

- **Description:** This will authenticate a customer by verifying their email and password.
- **Input:**

```
{
    "email": "nabeel@example.com",
    "password": "password"
}
```

- **Output:**

```
{
    "message": "Login successful",
    "user_id": 1
}
```

Registration

- **Description:** This will register a new customer by storing their details.
- **Input:**

```
{
        "fname": "Nabeel",
        "lname": "Mahmood",
        "user": "nabeel_mahmood",
        "email": "nabeel@example.com",
        "pw": "password"
    }
```

- **Output:**

```
{
    "id": 1,
    "fname": "Nabeel",
    "lname": "Mahmood",
    "user": "nabeel_mahmood",
    "email": "nabeel@example.com",
    "pw": "password"
}
```

Order API

Insert Order

- **Description:** This will insert a new order into the database.
- **Input:**

```
{
    "custID": 1,
    "orderStatus": "Pending",
    "orderContents": [
        {"item_id": 101, "quantity": 2},
        {"item_id": 102, "quantity": 1}
    ]
}
```

- **Output:**

```
{
    "orderID": 5001,
    "custID": 1,
    "orderStatus": "Pending",
    "orderContents": [
        {"item_id": 101, "quantity": 2},
        {"item_id": 102, "quantity": 1}
    ]
}
```

Inventory API

Get Item by UPC

- **Description:** This retrieves details of an inventory item using the UPC code.
- **Input:** /inventory/getByUPC/012345678901
- **Output:**

```
{
    "itemID": 101,
    "itemName": "Carrot",
    "UPC": "012345678901",
    "quantity": 50,
    "price": 1.50
}
```

Get All Inventory

- **Description:** This retrieves all available inventory items.
- **Input:** /inventory/all/
- **Output:**

```
[
    {
        "itemID": 101,
        "itemName": "Carrot",
        "UPC": "012345678901",
        "quantity": 50,
        "price": 1.50
    },
    {
        "itemID": 102,
        "itemName": "Lettuce",
        "UPC": "012345678902",
        "quantity": 15,
        "price": 0.75
    }
]
```

Check Inventory Availability

- **Description:** This checks the availability of a specific inventory item.
- **Input:**

```
/inventory/check/101
```

- **Output:**

```
{
    "itemID": 101,
    "available_quantity": 50
}
```

Warehouse API

Insert Item

- **Description:** Inserts a new item into the warehouse.
- **Input:**

```
{
    "upc": "012345678903",
    "qty": 25,
    "availability": "In Stock"
}
```

- **Output:**

```
{
    "message": "Item inserted successfully",
    "upc": "012345678903",
    "qty": 25,
    "availability": "In Stock"
}
```

Get Quantity by UPC

- **Description:** Retrieves the quantity of a specific item based on its UPC.
- **Input:**

```
/warehouse/qty/012345678901
```

- **Output:**

```
{
    "upc": "012345678901",
    "quantity": 50
}
```

3 API Standardization

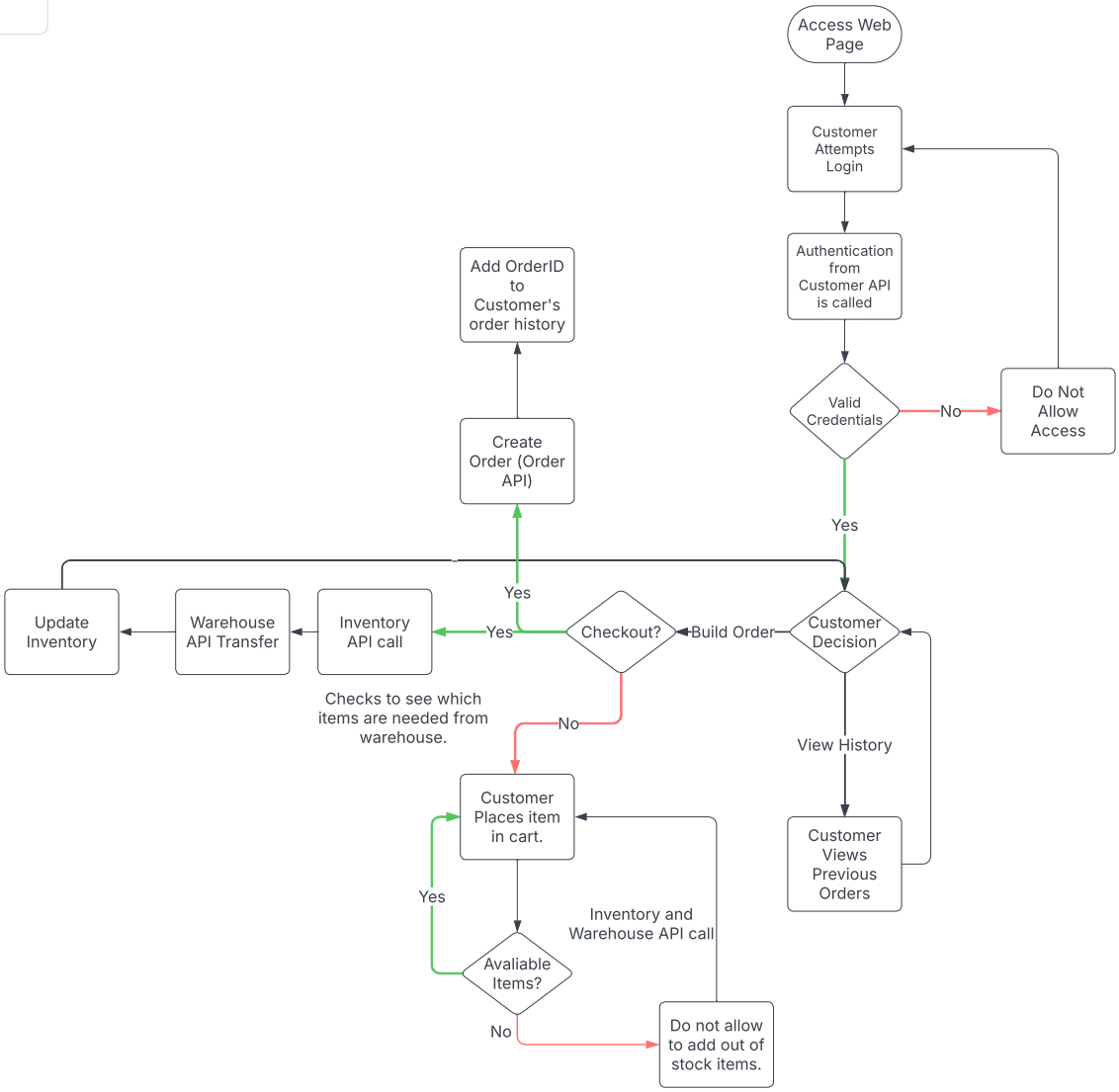
Exchanging Data

- Using JSON would be better for exchanging data compared to like HTTPS because it requires less lines of code and JSON is more simple to understand.
- We decided to follow a standard format to define and track fields, ensuring the data structure is consistent and the code remains clean and efficient.

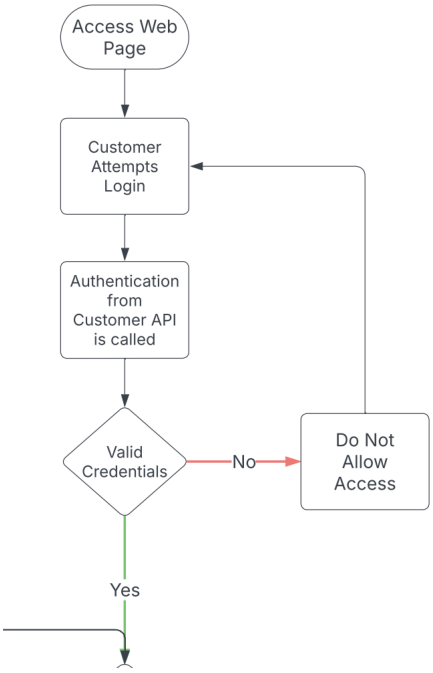
Naming Conventions

- For coding standards our team decided to follow a detailed plan and document each step of our project so we can keep track of it.
- We wanted to make sure that we label all files and code consistently using descriptive names to clearly indicate each files purpose. This will help us easily locate files when making adjustments or fixing errors, especially as the project grows.

4 Flowchart

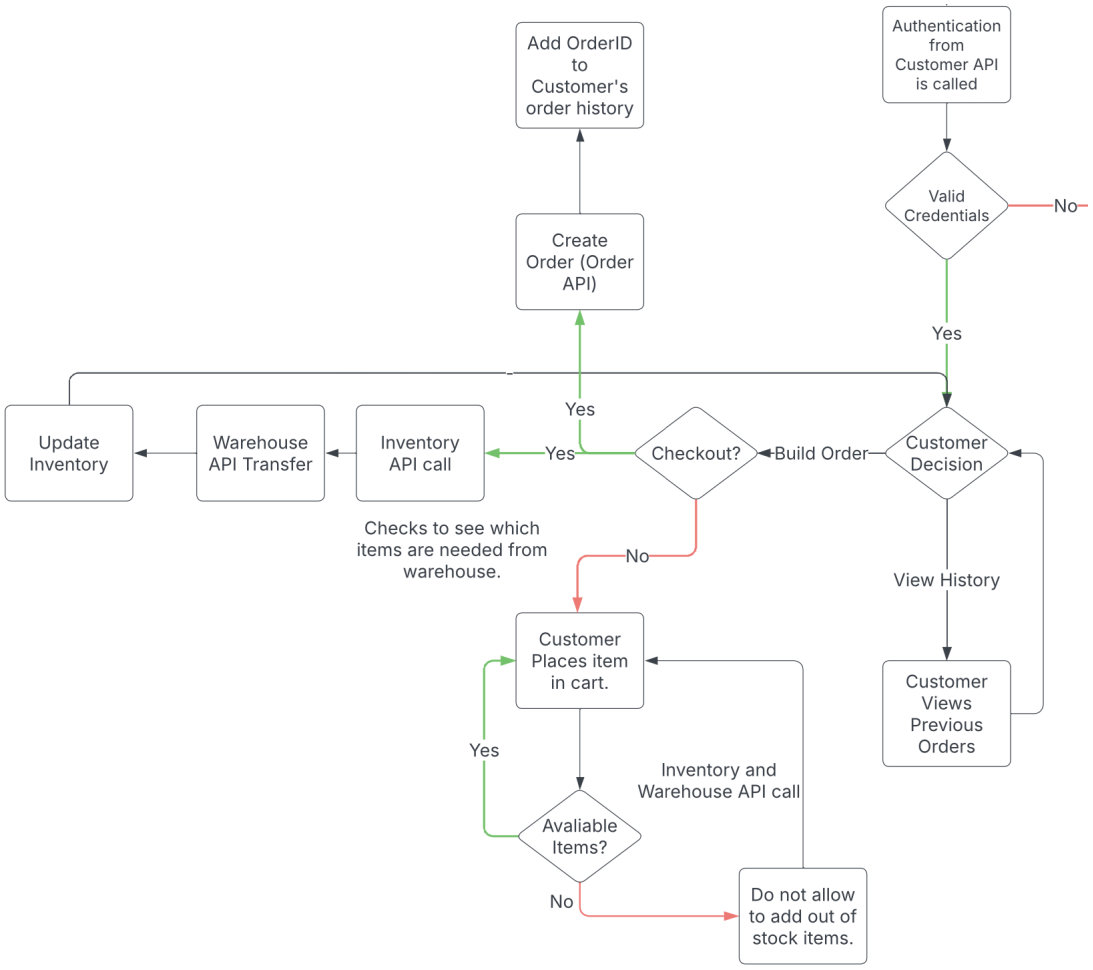

Access Web Page
Customer Attempts Login
Authentication from Customer API is called
Valid Credentials
No
Do Not Allow Access
Yes
Customer Decision
Build Order
View History
Customer Views Previous Orders
Checkout?
Yes
No
Checks to see which items are needed from warehouse.
Inventory API call
Warehouse API Transfer
Update Inventory
Create Order (Order API)
Add OrderID to Customer's order history
Customer Places item in cart.
Yes
Avaliable Items?
No
Inventory and Warehouse API call
Do not allow to add out of stock items.

Step 1



In our flowchart we first start off on when a customer accesses the website. When the customer accesses the website they will be attempted to log in and authentication from the Customer API will be called and it will make a decision if customer is authenticated then they will be able to go to the order page and if not they will go back to the login page.

Step 2



After the customer logs in they can either view their order history or build a new order. If they choose to view history a request is sent to the Order API to show their previous orders. If they then decide to build an order, the system checks item availability using the Inventory API. Available items are added to the cart, but out-of-stock items can't be added. If the customer proceeds to checkout, the system checks if more stock is needed. If so, a request is sent to the Warehouse API to transfer stock. Once the stock is ready, the order is created using the Order API and added to the customer's history. The inventory is then now updated.

5 Teamwork

Josh's Tasks

- Josh task was to create the docker containers and and use FastAPI library in Python to connect them to the various APIs.
- He also made sure that the backend was properly deployed and communicated with the frontend components.

Nabeel's Tasks

- My task was to work on the frontend, creating pages like the login page, order page, previous orders page, and manage account page, making sure everything functioned correctly.
- I created the project report and put together the presentation.
- Joe and I worked on improving the design and flow of the pages.

Joe's Tasks

- Some of Joe's Task he helped troubleshoot and debug various issues that came up during development.
- Joe was helping with the front end making sure it was functional.

Tasks Completed Together

- There were many tasks that we all did together like we all brainstormed different ideas and bounced them back and forth.
- We worked together to create the flowchart so we all understood how everything was connected.
- We worked together on updating our GitHub repository, reviewing each other's code, and making sure everything was running smoothly.
- Throughout the project, we kept in constant communication helping each other solve problems and stay on track.