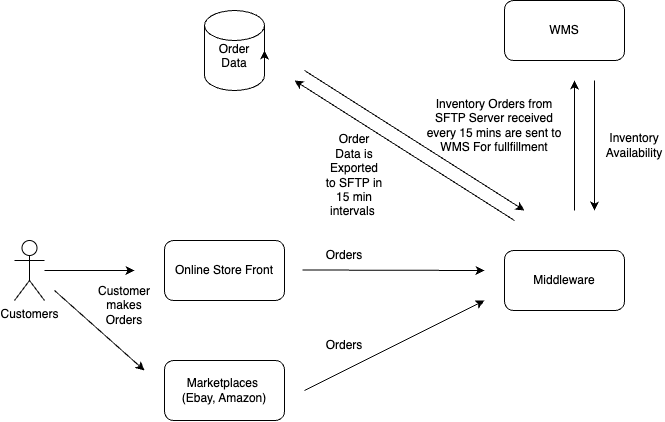
**Exercise 1**

Current State of System



Current Data Flow

1. Customers make orders via the Online Store Front and Marketplaces.
2. The orders are captured by the middleware servers and will compare inventory availability against the inventory updates received from the WMS which is every 15 mins.
3. In every 15 min intervals these orders will be exported to the SFTP Server, where Order data is stored.
4. The middleware is responsible for aggregating these orders and making fulfilments to the WMS in a separate 15 min frequency process.

Causes of over selling

The problem here is that more orders have been made than the available inventory capacity.

Following are the probable causes:

1. The WMS Inventory fulfilment batch process does not reflect the accurate number of inventory items that needs to be updated.

The reason for this could be batch process synchronisation issues. The inventory order csv reading process sftp may not have all the data yet, if the writing process is still writing to the SFTP CSV file. Appropriate CSV file locking mechanisms may not be in-place.

1. The WMS inventory availability process received from the WMS may not reflect the accurate inventory availability if the middleware to WMS write process is simultaneously happening. This would cause the Online Store Front to indicate to the customer a false positive inventory availability.
2. Probable network delays

Writing to files via SFTP and updating WMS Inventory would require a certain amount of network bandwidth. Any drop in network available or any other network failures can cause incorrect inventory data to be displayed.

1. Probable bugs in batch process codes, especially in the error handling mechanisms.

Solutions

Quick wins

* Increase the frequency of inventory writes from middle ware to WMS.
* Increase the frequency of inventory reads from WMS to middleware.
* Calculate the inventory availability and orders difference from historical data and incorporate an inventory buffer to prevent over selling.

Larger work

* Update the inventory in real time without relying on batch processes. For this we can introduce an event bus system using Rabbit MQ or Kafka, along with refined Rest end points.
* Build Order and Inventory management test suites that can simulate a live production order management workflow to verify accurate inventory updates.

**Recommendation:**

Refine the order and inventory management process with real time updates to the WMS and introduce event based streaming data with Apache Kafka

**Reasons for Recommendation:**

In the long term, implementing a solid system would be more cost effective for the business than quick wins.

**Exercise 2**

**Task1:**

|  |  |
| --- | --- |
| End Point: | POST /orders. |
| Request Headers | Content-Type: application/json Authorization: Basic <base64-code> |
| Request Body | {  "customer\_id": "ABC123",  "status": "pending",  "items":  [  {  "item\_number": "itemABC", "quantity": 2, "price": 19.99 },  }  ]  } |
| Response Body | {  "order\_id": "generated-order-id",  "customer\_id": " ABC123",  "placement\_date": "2024-07-26T12:34:56Z",  "status": "pending"  } |
| Response Codes | 201 – Created  400 - Bad Request  401 - Unauthorised |

|  |  |
| --- | --- |
| End Point: | GET /orders. |
| Request Headers | Content-Type: application/json Authorization: Basic <base64-code> |
| Request Params | customer\_id=<customer-id> |
| Response Body | [ { "order\_id": "order-id-1", "customer\_id": "12345", "placement\_date": "2024-07-26T12:34:56Z", "items": [ { "item\_number": "item-001", "quantity": 2, "price": 19.99 }, { "item\_number": "item-002", "quantity": 1, "price": 9.99 } ], "status": "pending" }, |
| Response Codes | 200 - Ok  401 - Unauthorised |

|  |  |
| --- | --- |
| End Point: | PUT /orders/{id} |
| Request Headers | Content-Type: application/json Authorization: Basic <base64-code> |
| Request Body | { "customer\_id": "12345", "items": [ { "item\_number": "item-001", "quantity": 3, "price": 19.99 }, { "item\_number": "item-002", "quantity": 1, "price": 9.99 } ], "status": "shipped" } |
| Response Body | {} |
| Response Codes | 200 – Ok  404 – Not Found  400 – Bad Request  401 - Unauthorised |

**Task2:**

See Golang code in folder. I am proficient with Java /Springboot as well.

How to run:

In mac:

./kindred

In Windows:

Run kindred.exe

**Test:**

**GET**

curl -u admin:password <http://localhost:8080/orders?customer_id=10001>

**POST**

curl -X POST http://localhost:8080/orders \

-u admin:password \

-H "Content-Type: application/json" \

-d '{

"customer\_id": "10001",

"items": [

{

"item\_number": "item-001",

"quantity": 2,

"price": 19.99

},

{

"item\_number": "item-002",

"quantity": 1,

"price": 9.99

}

],

"status": "pending"

}'