Supr Machine Coding Round

Time Limit: 2 hours

Description

SuprDaily does daily delivery of milk and groceries every morning. Customers can order by 11 pm, get deliveries before 7 am. Customers order items which are products like Apple (1pc), Onion (0.5kg), Parle-G Biscuit (200g), and so on.

Design & Build APIs that provide the capability to check and reserve orders that we can serve on particular *delivery date*. An order can be served only if all items (with requested quantities) can be served. In order to do this, there are a bunch of criteria that we want to consider :

1. We want to have a check based on available items in the Warehouse : A customer order is served from a Warehouse (a place used to pack items to be sent to customer). Warehouse has items which are available for different delivery dates. If item(s) are not available, we cannot take the order. Items start to become unavailable as and when we reserve orders.

   Eg. Warehouse has Item i1 :

   100 units for 15 Oct 20

   200 unit for 16 Oct 20

   If we have reserved 99 units of i1 for orders on 15 Oct 20, we can take only order for 1 quantity more for i1.

2. We also want to have a check based on item category : Every Item belongs to a Category. Eg. All fruits, vegetable items belong to a category called "F_N_V". We want to be able to throttle maximum items that can be shipped for a given category on a given delivery date. This is irrespective of the Warehouse that serves the customer.

   Eg.

Category "F_N_V":

150 units on 15 Oct 20

200 units on 16 Oct 20

Combined orders for a date should not be more than 150 units

We can fulfil the order only if all the criteria are met.

Note:

- You can assume an order once reserved is never cancelled (for the purpose of this problem).
- For implementing each criterion, please assume all relevant data/configs to be present in-memory and define relevant data structures for them. The data can be assumed to be static (defined in code)

Input Data Models & Interfaces

```
class ItemRequest {
    Integer itemId;

    String itemName;

    String category; // Category that the items belongs to.

    Integer quantity; // Quantity trying to be ordered.
}


class OrderRequest {
    String customerId; // Customer ID

    String warehouseId; // ID of Warehouse which serves the Customer

    Date deliveryDate; // Delivery Date

    List<ItemRequest> items; // Items that Customer is trying to Order
}
```

Interfaces to be implemented (Programming language does not matter, Java is just an example):

```
public interface OrderFulfilmentService {

    /**
     * Takes an order request and checks if we can fulfil that order based
     * on various criterias mentioned in the problem
     *
     * @param orderRequest Order Request that customer is trying to order.
     * @return true if we can take this order, false if we cannot.
     */
    Boolean canFulfilOrder(OrderRequest orderRequest);


    /**
     * Reserves the items in the order request. This is usually called by client
     * after checking the above method. If an order is reserved,
     * that inventory is not available for any other order.
     *
     * @param orderRequest Order Request that customer is trying to order.
     * @throws OrderReservationException if we cannot take that order.
     */
    void reserveOrder(OrderRequest orderRequest) throws OrderReservationException;

}
```

Please define relevant HTTP APIs to expose these functionalities

API 1: Check If We can Fulfil An Order

Input:

```
{
  "customer_id": 10001,
  "delivery_date": "2020-10-13",
  "warehouse_id": 100,
  "items": [
    {
      "item_id": 1,
      "item_name": "Washington Apple (1 pc)",
      "category": "F_N_V",
      "quantity": 3
```

```
    },
    {
      "item_id": 2,
      "item_name": "Banana (0.5kg)",
      "category": "F_N_V",
      "quantity": 1
    },
    {
      "item_id": 3,
      "item_name": "Parle-G Biscuit (200g)",
      "category": "Grocery",
      "quantity": 1
    }
  ]
}
```

Output:

```
{
  "can_fulfil": true
}
```

API 2: Reserve An Order

Input:

```
{
  "customer_id": 10001,
  "delivery_date": "2020-10-13",
  "warehouse_id": 100,
  "items": [
    {
      "item_id": 1,
      "item_name": "Washington Apple (1 pc)",
      "category": "F_N_V",
      "quantity": 3
    },
    {
      "item_id": 2,
      "item_name": "Banana (0.5kg)",
      "category": "F_N_V",
      "quantity": 1
    },
```

```
    {
      "item_id": 3,
      "item_name": "Parle-G Biscuit (200g)",
      "category": "Grocery",
      "quantity": 1
    }
  ]
}
```

Output:

```
{
  "code": "Success",
  "data" : {
    "reserved": true,
    "message": "Success"
  }
}
```

In case of any error:

```
{
  "code": "Success",
  "data" : {
    "reserved": false,
    "message": "Insufficient quantities!"
  }
}
```

Guidelines

- Databases are not required, you can keep everything in memory
- Don't spend much time on I/O formats
- You can assume that a single process running for this (and not deal with distributed environment)

Evaluation Criteria

- Interviewer will evaluate your code - functionality, design etc.
- Code should be modular and readable
- Code should be demo-able and extensible
- In the Review round, you need to explain your approach, models and why that was chosen
- Interviewer might give you some extra cases and ask clarifying questions
- You're expected to submit the solution at the end of 2 hours (even if incomplete) by zipping the code and replying on email to the interviewer. (Can use GDrive if attachment not working)