Hackathon problem statement

The objective of this challenge is to allow small retailers, kirana stores and similar vendors to order optimal quantities of items to reduce wastage and inventory costs.

Learnings from our conversations with small retailers

After the hackathon started, we went and talked to a bakery and grocery store to understand how inventory management happens now. Here are some learnings:

1. Most unorganized retailers use **local POS and ERP** systems to manage inward and manage inventory
    a. The POS systems are typically offline and not on cloud
    b. These systems are not smart enough to do any kind of inventory management backed by research.
2. Ordering is done by store manager manually with few inputs
    a. Understanding of demand and enough buffer to prevent stockouts
    b. The visiting distributor would simply look at the shelves and advise an estimated order quantity.
3. No automated way to know stock level or ordering directly due to lack of any integration
4. Orders are placed by retailers in specific periods - say one day of the week - for this reason, we decided it was a good idea to start with a “Periodic review” model of inventory management while solving this challenge, instead of a “Continuous review” model.
5. Typical lead time of 1-2 days, however can change by industry

Our solution

Based on the above observations, our solution does the following

- Minimize the **overall cost of managing inventory** by optimizing lot size and target service level to fulfill customer demand
    - Use of advanced statistical models to give research backed suggestions to small retailers
- **Ease of use** by small retailers

    - Calculates cost estimates for different service levels using past order data.
    - Use the same model of ordering (fixed periods, lead time) etc that the actual retailer uses
- **Easy portability to ONDC**
    - API based access to update orders data and model variables by seller apps

Assumptions

- Seller apps on Beckn network can integrate with third party apps and send order details.
- Retailers order in fixed frequency (say once a week) on fixed days of the week (learned this from our conversations with retailers).
- Retailers have some idea on what the holding cost of inventory is

Working demo of prototype

https://alme.tech/products

Instructions to build and operate the app

A demo version is hosted on https://alme.tech. You can use this to test the app, but in order to run the app locally, here are the instructions:

Prerequisites:
erlang >= 24.1.1
nodejs >= 17.3.0
elixir >= 1.12.3
PostgresDB running locally.

Steps to run:

1. Clone the github repository https://github.com/almetech/innovation-hackathon-jan22
2. Run “mix deps.get” to install the dependencies
3. Run “cd assets && npm i && cd ..” to install the JS packages
4. Run “mix ecto.setup” to run the DB migrations

5. Run “mix phx.server” to start the web server. Go to http://localhost:4000 to access the app.

Algorithms

1. Periodic review of inventory algorithm (review inventory after every period, stock upto target inventory level) - Based on this research paper on the simulation of continuous review and periodic review. Retailers restock at fixed intervals, so this algorithm is ideal.
2. Based on target service level **a**, review_prd **R,** lead_time **LT** , standard deviation of demand **std_demand** the formula to determine lot size **q** is

   *q = mean_demand * (R+LT) + std_demand * sqrt(R+LT) * service_factor(a)*

3. However the retailer can lower his cost by finding the optimum service level which balances his inventory holding cost and stock out cost

4. The formula to find the **optimum service level** is as per this in this paper

$$\alpha^* = 1 - \frac{hR}{b}$$

   alpha = targeted service level
   hR = holding cost over Review period
   b : stock out cost expressed as %age of product price

5. Now we find the **ideal service level** for the product and the corresponding inventory cost which comprised of the following costs

   total_cost = holding_cost + transaction_cost + stockout_cost

Cost	Formula	Variables

holding_cost	$h(dR/2 + z\,\sigma_x)$	h : holding cost d : avg demand R : review period z : service factor sigma_x: std deviation over review and lead time
transaction_cost	$\frac{k}{R}$	k : fixed cost per order (e.g. logistics) R: review period
stockout_cost	$\frac{b\,\sigma_x \mathcal{L}_N(z)}{R}$	b: backorder cost per unit sigma_x: std deviation over review and lead time R: review period Ln(z): Normal loss function for the service factor corres to the service level

6. We generate economic cost for current practice lot size, target service level (given by retailer) and ideal service level (suggested by retailer) and show the cost benefit to the retailer
7. The retailer can start ordering based on lot sizes suggested by us

Dataset

We use a generated random dataset based on some commonly occurring products in retail stores. Can be accessed here . Sample below

id	item	quantity	price	date
1	Maggie noodles	95	12	01/01/21
1	Maggie noodles	75	12	02/01/21
1	Maggie noodles	43	12	03/01/21
1	Maggie noodles	68	12	04/01/21
1	Maggie noodles	74	12	05/01/21

Sample output for one item - Shows comparison for 92% service level by retailer, ideal service level suggested by us, current practice of lot size = average demand for the review period

Maggie noodles (Avg Daily Demand: 62)	Service Level - 92%	**Ideal service level - 96%**	Lot size =Avg daily demand * review period
Lot qty	580	600	500
Cost over review period (INR)	138	114	217

Reusability (APIs, algorithms, data sets, etc)

- Using well known research algorithms as discussed above
- Works with any orders dataset from retailers.
- The algorithm is exposed to the end user via HTTP API endpoints.

Scalability (can this work at population scale)

- The data processing and mathematical calculations happen in separate threads. We have used the Erlang Virtual Machine which supports high concurrency.
- API endpoints can scale horizontally to serve more seller apps.

Compliance to Security standards and transaction guarantees (as applicable)

- All DB queries that are atomic in nature are wrapped in transactions
- HTTP JSON API endpoints are protected using Bearer token authentication
- All endpoints in the demo use SSL.