

Project Documentation

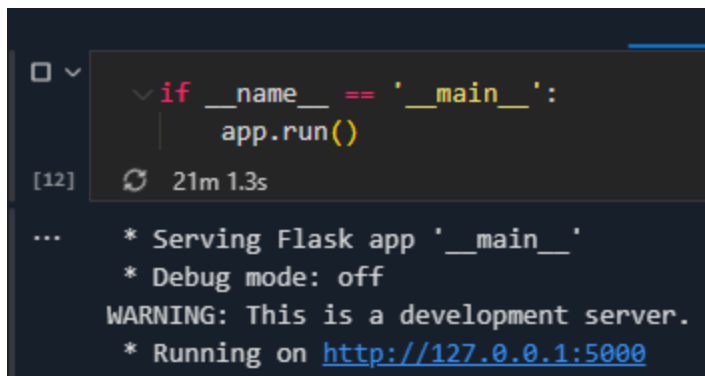
Overview

Currently, the WMS provides information regarding how much money can be saved by using the smart configuration over the random configuration.

The smart configuration currently sorts the items based on demand per day, but is meant to be more sophisticated in the future, taking into consideration proximity (similar items should be placed together), weight, size etc. Another extension includes treating the layout as a multi-level system, adding a vertical dimension to the problem.

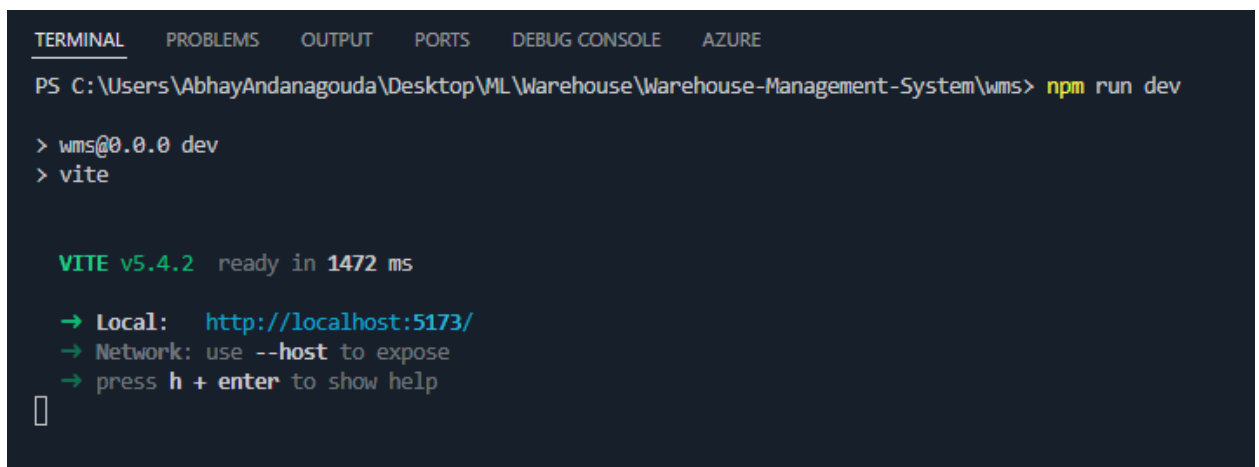
How to run?

- Go to ...
- **WMS-Visualizer** contains the backend (Python) and **wms** contains the frontend (React application)
- To start the backend, open the file **wms-visualizer.ipynb** and run all the cells



```
if __name__ == '__main__':  
    app.run()  
[12] 21m 1.3s  
... * Serving Flask app '__main__'  
      * Debug mode: off  
      WARNING: This is a development server. It is not recommended for production.  
      * Running on http://127.0.0.1:5000
```

- To start the frontend, go to **./wms** and run **npm run dev** and visit the url: **<http://localhost:5173/>**

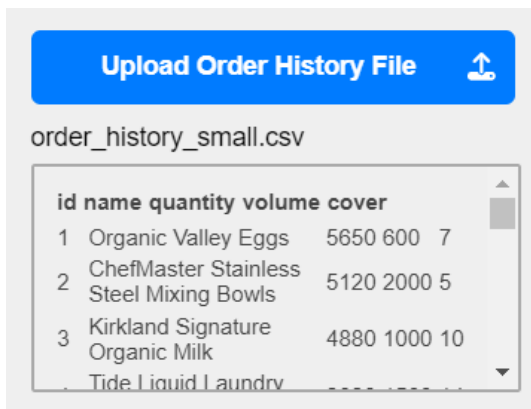


```
TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  AZURE  
PS C:\Users\AbhayAndanagouda\Desktop\ML\Warehouse\Warehouse-Management-System\wms> npm run dev  
  
> wms@0.0.0 dev  
> vite  
  
VITE v5.4.2 ready in 1472 ms  
  
→ Local:   http://localhost:5173/  
→ Network: use --host to expose  
→ press h + enter to show help  
█
```

How to use?

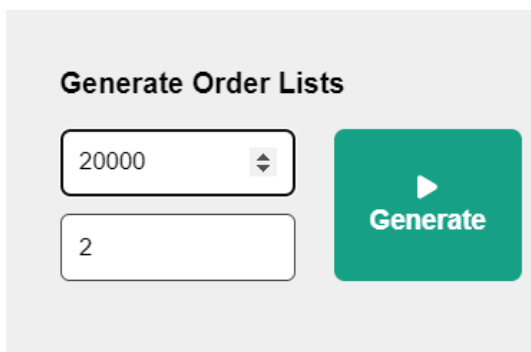
- **Order History**

Order history file for the previous 6 months. File should have the columns - id, name, quantity, cover. Here, quantity refers to the total number of orders in the last 6 months. Daily demand is calculated based on quantity ($\text{demand} = \text{quantity} / 6 * 30$). Inventory size i.e. how many items must be stored in the warehouse, is calculated as ($\text{inventory} = \text{cover} * \text{daily demand}$).



id	name	quantity	volume	cover
1	Organic Valley Eggs	5650	600	7
2	ChefMaster Stainless Steel Mixing Bowls	5120	2000	5
3	Kirkland Signature Organic Milk	4880	1000	10
	Tide Liquid Laundry			

- **Generate Order Lists**

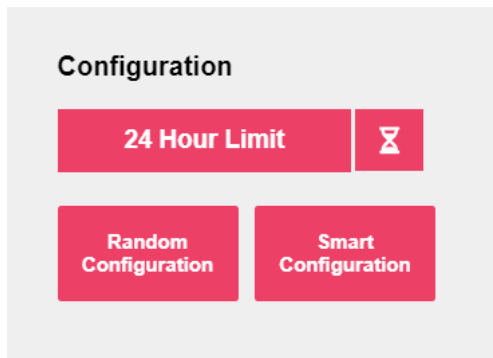


The generated order list files are stored in **./data/orders/**

In this example, we are generating 2 such files. Each of these files contains 20,000 orders, and each order can contain multiple items. For more accurate results, more

files can be generated, but this will take more time, especially if the number of orders per file is large.

- **Configuration**



Here, you can opt for a one day time limit. If this option is chosen, the time is fixed and the number of orders fulfilled will vary. If no time restriction is set, all the orders will be fulfilled and the time taken will vary.

Warehouse Info

Cells Travelled:	323941.00
Distance Travelled:	242955.75 meters
Order Fulfilment Time:	71.99 hours
Order Processed:	10466

With Smart Config, you can

Process 6512 more orders

Wages per Hour

Profit per Order

Money saved: \$39069.00

Warehouse Info

Cells Travelled:	598661.00
Distance Travelled:	448995.75 meters
Order Fulfilment Time:	133.04 hours
Order Processed:	20000

With Smart Config, you can

Save 48.5 hours

Wages per Hour

Profit per Order

Money saved: \$0.00

Code

class Warehouse

Contains information regarding the layout of the warehouse, including number of bays, aisles, cells per bay, cell length, cell capacity etc.

Contains an instance of bay iterator designed for efficient fetching of random cells. This iterator will be used in the **sprinke_aisle()** function for random distribution of items across the cells of the warehouse.

Contains a list containing **worker** objects. For this layout, there are 3 workers.

aisle_items maps item id to the item object, so that all details about the item can be fetched from the id.

dist_batched_order_list() distributes the orders to the workers

class Worker

min_max_order_list() gets the minimum (closest) and maximum (farthest) cell along the bay of the warehouse. Only max_cell is used here.

fulfill_batched_order() will have the worker fulfill an order and update the distance traveled for that worker based on **max_cell** (the farthest cell he will have to pick items from). It returns the number of orders processed, so that we can keep track of how many orders have been fulfilled in total.


Flow

Api.jsx file handles all the API requests.

Inputs.jsx forms the left panel in the frontend. Under the Generate Order Lists section, clicking Generate triggers **handleGenerate()** which makes an API call to **datagen** endpoint. This will create the necessary order list files.

Under the configuration section, there are two buttons - "Random Configuration" and "Smart Configuration". Clicking on either of these calls the **handleSubmit()** function in **Api.jsx** which makes a request to the **upload** API endpoint. The **upload** endpoint initializes the warehouse object and calls **wms_runner**, which distributes the items across the warehouse layout and generates the item to cell location mapping.

App.jsx uses **fetchCellData()** in **Api.jsx** to get the mapping, which is passed as **cellData** to **Layout.jsx** which displays the items in their designated cells.



Similarly, **App.jsx** uses **fetchWarehouseData()** to pass information like orders processed, time taken, etc to **CellView** which forms the right panel of the frontend.