# Phase 2: Proof of Concept Implementation Report

Susan Subedi

Algorithms and Data Structure

Professor. Satish Penmatsa

University of the Cumberlands

February 2, 2025

# Phase 2: Proof of Concept Implementation Report

Partial Implementation Overview

In this implementation, the proof of concept implementation was used to create a dynamic inventory management system that is more efficient in stock management (Dunn, 2014). A number of data structures were used, as discussed below. First, the hash table was used for storing and retrieving the products from the database. It used the product SKU as a key so that there can be an O(1) lookup (Yu et al., 2022).

Queue (FIFO) was also included in the implementation for the purpose of handling the stock orders and processing them in the order they were received. This concept of first in, first out is discussed by Ganesan et al. (2009). Min-heap is the last structure used to prioritize low-stock products, ensuring there is no stockout (Yu et al., 2022).

We have used Python classes to implement each of these data structures and promote the modularity and scalability of the system. For example, the hash table will allow quick product lookups, improving the efficiency of the system.

Demonstration and Testing

The following is how the implementation is demonstrated and tested. In most cases, several test cases need to be executed to validate the implementation. After testing the system using the test cases, the following were the results.

The output of adding a product to the inventory was  as follows

Successfully added three products, A100, B200, and C300 to the hash table.

Output

This is how the inventory will look like after adding products

[Product(SKU: A100, Price: 20.5, Category: Electronics, Stock: 15),

 Product(SKU: B200, Price: 10.0, Category: Groceries, Stock: 5),

 Product(SKU: C300, Price: 50.0, Category: Clothing, Stock: 2)]

Fetching a Product

Successfully retrieved product B200 from the inventory.

Here is the output after successfully retrieving a product from the inventory

Fetching product B200

Product(SKU: B200, Price: 10.0, Category: Groceries, Stock: 5)

Removing a Product

After successfully removing product A100 from the inventory, the inventory was updated correctly. The following is the output.

Output

Here is how the inventory looks after removing the product

 [Product(SKU: B200, Price: 10.0, Category: Groceries, Stock: 5),

 Product(SKU: C300, Price: 50.0, Category: Clothing, Stock: 2)]

Processing Restock Orders

The FIFO queue was used to process restocking for B200, and the following is the output

Processing Restock

Product(SKU: B200, Price: 10.0, Category: Groceries, Stock: 5)

Identifying Low-Stock Products

The min-heap identified C300 as the lowest stock item, confirming the data structure implementation is working as expected. Here is the output

Product with the lowest stock

Product(SKU: C300, Price: 50.0, Category: Clothing, Stock: 2)

Implementation Challenges and Solutions

During the implementation of the data structure above, a number of challenges were faced. For example, one challenge was handling hash collisions, which were resolved using separate chaining to maintain performance. There was also an issue with processing keys. The implementation had to ensure FIFO integrity by maintaining proper indexing.

The following are useful code snippets for implementation and their documentation covering four functionalities

Adding Products

def add\_product(self, product):

    if product.sku not in self.products:

        self.products[product.sku] = product

        heapq.heappush(self.priority\_heap, product)

    else:

        raise ValueError("SKU already exists.")

Processing Restock Queue

def process\_restock(self):

    if self.restock\_queue:

        return self.restock\_queue.pop(0)

    return "No products to restock."

Fetching Low-Stock Items

def get\_low\_stock\_product(self):

    if self.priority\_heap:

        return heapq.heappop(self.priority\_heap)

    return "No low-stock products."

Additional implementation details can be found in the shared GitHub link.

# References

Yu, M., Tian, X., & Tao, Y. (2022). Dynamic model selection based on demand pattern classification in retail sales forecasting. *Mathematics, 10*(17), 3179. https://doi.org/10.3390/math10173179

Ganesan, S., George, M., Jap, S., Palmatier, R. W., & Weitz, B. (2009). Supply chain management and retailer performance: emerging trends, issues, and implications for research and practice. *Journal of Retailing, 85*(1), 84-94. https://doi.org/10.1016/j.jretai.2008.12.001

Dunn, I. W. (2014). *Proving correctness of actor systems using FIFO communication* (Doctoral dissertation, Rensselaer Polytechnic Institute, Troy, NY).