DYNAMIC INVENTORY MANAGEMENT SYSTEM

Deliverable 2

JULY 27, 2025

BISHNU SHARMA
ID: 005036007

Proof of Concept Implementation for a Dynamic Inventory Management System Using Python

Introduction

The second phase of this project focuses on building a proof of concept for a dynamic inventory management system using Python. This phase translates the conceptual designs from Phase 1 into a working partial implementation. The goal is to demonstrate key features such as product management, stock prioritization, and price-based sorting using appropriate data structures like hash tables, heaps, and binary search trees. Although the application is not complete, this stage lays the groundwork for a fully functional and scalable system.

Partial Implementation Overview

The system's core architecture is divided into three main components:

- ProductManager: Manages product information using a hash table (Python dictionary) for efficient access and updates.
- StockHeap: Implements a min-heap to identify products with the lowest stock quantities using Python's heapq.
- PriceBST: Implements a basic binary search tree (BST) to sort products by price. This is currently unbalanced but supports insertion and in-order traversal**.**

Each class is defined in a separate module to ensure modularity and ease of scaling. For example, the ProductManager handles adding and storing product instances:

```
class ProductManager:
    def __init__(self):
        self.products = {}

    def add_product(self, id, name, quantity, price):
        if id not in self.products:
            self.products[id] = Product(id, name, quantity, price)

    def get_all_products(self):
        return list(self.products.values())
```

Demonstration and Testing

To test the functionality, a script was created to simulate realistic operations: adding products, retrieving low-stock items, and sorting by price.

```
manager = ProductManager()
manager.add_product("P001", "Laptop", 5, 1000)
manager.add_product("P002", "Mouse", 2, 20)
manager.add_product("P003", "Keyboard", 8, 50)
```

For low-stock filtering using a heap:

```
stock_heap = StockHeap(manager.get_all_products())
for item in stock_heap.get_low_stock(2):
```

Price-based sorting using BST:

```
bst = PriceBST()
for product in manager.get_all_products():
    bst.insert(product)

for p in bst.inorder():
    print(f"{p.name}: ${p.price}")
```

Testing focused on correctness of insertions, the ability to retrieve products with the lowest quantity, and confirming correct order in price-based traversal. Edge cases such as duplicate product IDs were handled gracefully.

Implementation Challenges and Solutions

One challenge was ensuring consistency between multiple structures. To maintain data integrity, the ProductManager was designed as the central authority for adding and accessing products. Another issue was compatibility with the heapq module, which required implementing a comparison method in the Product class:

```
class Product:
    def __lt__(self, other):
        return self.quantity < other.quantity
```

Python's lack of built-in self-balancing trees led to the use of a simple unbalanced BST. While functional for demonstration purposes, it may introduce performance issues as data grows.

Next Steps

To evolve this proof of concept into a production-ready system, the following enhancements are planned:

- Replace BST with an AVL or Red-Black Tree for efficient price-based operations.
- Introduce persistent data storage.
- Add delete and search capabilities.
- Create a command-line or graphical user interface.
- Implement comprehensive error handling and user input validation.
- Increase unit test coverage and integrate with version control (GitHub).

These improvements will allow the system to scale effectively and handle more complex inventory scenarios.

Conclusion

This phase successfully demonstrates the fundamental operations of the dynamic inventory management system, focusing on modular, testable components. The partial implementation shows how product records can be inserted, queried, and sorted using basic data structures. This foundation sets the stage for optimization and expansion in Phase 3.

GitHub

The GitHub link for implementation is provided below:

https://github.com/iambissnuu/MSCS532_InventotySystem

References

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Python*. Wiley.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley.