11/16/2024

# Project Phase 2

Proof of Concept for Inventory Management System

Rutu Shah
005026421

## Table of Contents

# Report on Phase 2: Proof of Concept Implementation for Inventory Management System

## 1. Introduction

In this phase, I have implemented a partial proof of concept (PoC) for the inventory management system using Python's Flask framework which is lightweight and WSGI web application framework (Pallets. (n.d.). *Flask documentation (Version stable)*) and a Hash Map data structure for efficient inventory lookup. The PoC highlights the key operations of item management, including insertion, deletion, searching, and listing of items. This report provides an overview of the core functionality, including implementation details, test cases, challenges encountered, and future steps for completing the full implementation.

## 2. Partial Implementation Overview

### 2.1. Data Structure and Application Design

The core data structure used for managing the inventory is a **Hash Map (Dictionary in Python)**. This allows for quick lookups, additions, and deletions based on item IDs, making it an ideal structure for an inventory system. The **InventoryItem** class represents individual items in the inventory, encapsulating essential details such as item_id, name, category, quantity, and price.

The Inventory class manages the inventory using a dictionary where the key is the item_id, and the value is an instance of the **InventoryItem** class. The following key operations are implemented:

- **add_item**: Adds an item to the inventory if it doesn't already exist.
- **delete_item**: Deletes an item from the inventory based on its item_id.

- **search_item**: Retrieves an item from the inventory based on its item_id.
- **list_items**: Lists all inventory items, sorted by their name.

## 2.2. Key Functions

In the key functions, I have added the demonstration of core functionality as mentioned below

1. **Insertion**: Allows to add unique inventory items, ensuring no duplicate inventory is added into the system.
2. **Deletion**: Removes the items by their item_id.
3. **Search**: Retrieves an item based on its item_id.
4. **Traversal**: Listing all items in the inventory sorted by name.

In addition to this, I am also attaching the python code of InventoryItem class below defining the item_id,name,category,quantity and price.

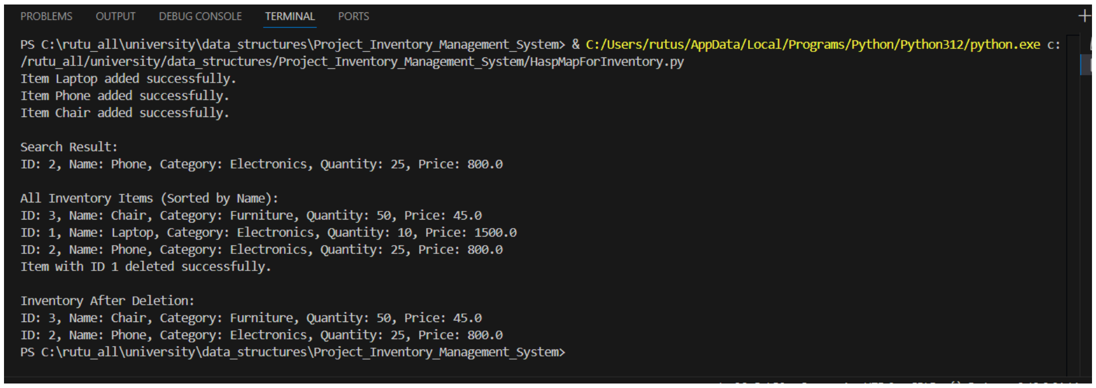
```python
class InventoryItem:
    def __init__(self, item_id, name, category, quantity, price):
        self.item_id = item_id
        self.name = name
        self.category = category
        self.quantity = quantity
        self.price = price

    def __str__(self):
        return f"ID: {self.item_id}, Name: {self.name}, Category: {self.category}, Quantity: {self.quantity}, Price: {self.price}"
```

Adding the code of my inventoryclass, add_item, search_item, delete_item and list_item code.

```
class Inventory:
    def __init__(self):
        self.items = {}  # Hash map for storing inventory items

    def add_item(self, item_id, name, category, quantity, price):
        if item_id in self.items:
            print(f"Item with ID {item_id} already exists.")
            return False
        self.items[item_id] = InventoryItem(item_id, name, category, quantity, price)
        print(f"Item {name} added successfully.")
        return True

    def delete_item(self, item_id):
        if item_id not in self.items:
            print(f"Item with ID {item_id} does not exist.")
            return False
        del self.items[item_id]
        print(f"Item with ID {item_id} deleted successfully.")
        return True
```

```python
def search_item(self, item_id):
    return self.items.get(item_id, None)


def list_items(self):
    for item in sorted(self.items.values(), key=lambda x: x.name):
        print(item)
```

Output of the code:



```
PS C:\rutu_all\university\data_structures\Project_Inventory_Management_System> & C:/Users/rutus/AppData/Local/Programs/Python/Python312/python.exe c:
/rutu_all/university/data_structures/Project_Inventory_Management_System/HaspMapForInventory.py
Item Laptop added successfully.
Item Phone added successfully.
Item Chair added successfully.

Search Result:
ID: 2, Name: Phone, Category: Electronics, Quantity: 25, Price: 800.0

All Inventory Items (Sorted by Name):
ID: 3, Name: Chair, Category: Furniture, Quantity: 50, Price: 45.0
ID: 1, Name: Laptop, Category: Electronics, Quantity: 10, Price: 1500.0
ID: 2, Name: Phone, Category: Electronics, Quantity: 25, Price: 800.0
Item with ID 1 deleted successfully.

Inventory After Deletion:
ID: 3, Name: Chair, Category: Furniture, Quantity: 50, Price: 45.0
ID: 2, Name: Phone, Category: Electronics, Quantity: 25, Price: 800.0
PS C:\rutu_all\university\data_structures\Project_Inventory_Management_System>
```

## 3. Demonstration of test case to show the basic operations, such as handling edge case and correctness of code.

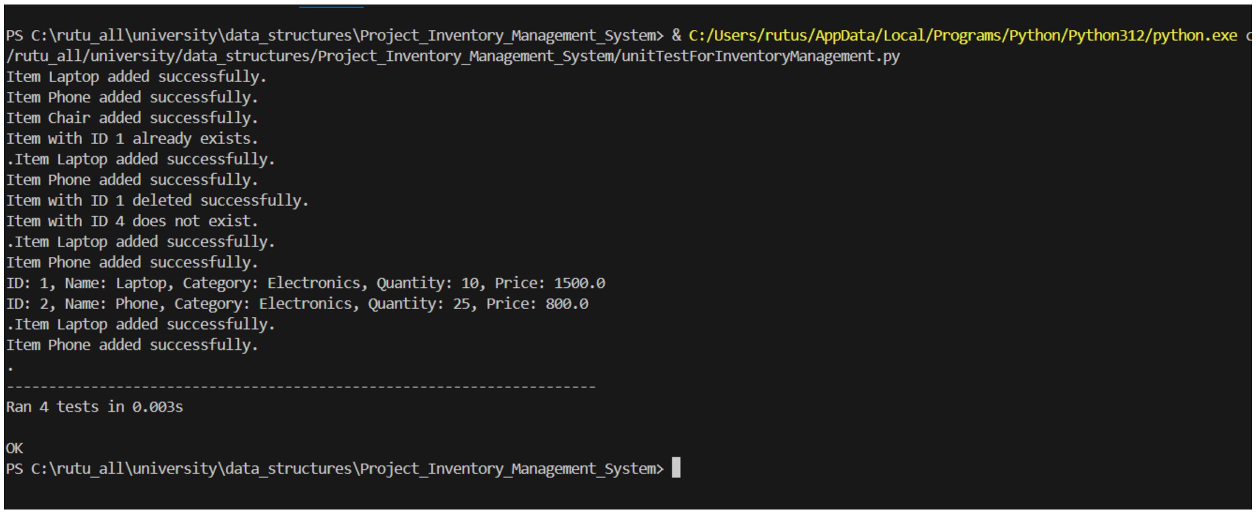
To ensure the functionality of the inventory system, I have written a bunch of test cases in a separate file name unitTestForInventoryManagement.py which imports unittest package of python which is inbuilt test framework in python that is available as standalone library. These test cases verify the correct operation of key methods such as adding, searching, deleting, and listing items.

### 3.1 Below is the attached code of my unitTestForInventoryManagement.py

```
# Created By : Rutu Shah

# Created Date : 15th November 2024

# This is the unitTestForInventoryManagement.py file which i have implemented to test the working of

# Inventory Lookup developed using Hash Map Data Structure by importing unittest package, providing in built testing framework in python



import unittest

from HaspMapForInventory import  InventoryItem,  Inventory


class TestInventory(unittest.TestCase):

    def setUp(self):

        self.inventory = Inventory()

        self.inventory.add_item(1, "Laptop", "Electronics", 10, 1500.0)
```

```python
        self.inventory.add_item(2, "Phone", "Electronics", 25, 800.0)

    def test_add_item(self):
        self.assertTrue(self.inventory.add_item(3, "Chair", "Furniture", 50, 45.0))
        self.assertFalse(self.inventory.add_item(1, "Duplicate Laptop", "Electronics", 5, 1400.0))

    def test_search_item(self):
        item = self.inventory.search_item(1)
        self.assertIsNotNone(item)
        self.assertEqual(item.name, "Laptop")

    def test_delete_item(self):
        self.assertTrue(self.inventory.delete_item(1))
        self.assertFalse(self.inventory.delete_item(4))

    def test_list_items(self):
        self.inventory.list_items()  # Manually verify output if necessary

if __name__ == "__main__":
    unittest.main()
```

**Output of the unitTest**



```
PS C:\rutu_all\university\data_structures\Project_Inventory_Management_System> & C:/Users/rutus/AppData/Local/Programs/Python/Python312/python.exe c
/rutu_all/university/data_structures/Project_Inventory_Management_System/unitTestForInventoryManagement.py
Item Laptop added successfully.
Item Phone added successfully.
Item Chair added successfully.
Item with ID 1 already exists.
.Item Laptop added successfully.
Item Phone added successfully.
Item with ID 1 deleted successfully.
Item with ID 4 does not exist.
.Item Laptop added successfully.
Item Phone added successfully.
ID: 1, Name: Laptop, Category: Electronics, Quantity: 10, Price: 1500.0
ID: 2, Name: Phone, Category: Electronics, Quantity: 25, Price: 800.0
.Item Laptop added successfully.
Item Phone added successfully.
.
----------------------------------------------------------------------
Ran 4 tests in 0.003s

OK
PS C:\rutu_all\university\data_structures\Project_Inventory_Management_System>
```

## 3.2 How the Data Structures Support the Application's Requirements

The HashMap implementation works perfectly with the basic application needs as follows:

1. Fast Lookup: Hashmaps have O(1) average time complexity in insertion, deletion and search so that you will have a quick inventory.
2. Easy Use: Hashmaps are key-value types that makes it easy to represent inventory items with item_id as the key and the InventoryItem object contains the item details.
3. Scalability: The hashmap will grow with your inventory without any performance hit.
4. Flexibility: Other functionality like categorical filtering or updating the items details can be added on top of the existing solution.

Even in its narrow use case, the HashMap shows how a scalable, efficient data model underlies essential processes such as inventory fetch and update.

## 4. Implementation Challenges and Solutions

4.1. Challenge: Handling Duplicate Items Initially, I faced a challenge when adding items to the inventory. The system needed to ensure that items with the same item_id weren't added multiple times. I addressed this by checking if the item_id already exists in the inventory before inserting a new item.

4.2. Challenge: Item Deletion Another issue was ensuring that the program correctly handles the deletion of items. If an item did not exist, the system needed to return an appropriate error message. This was solved by checking if the item exists before attempting to delete it.

4.3. Flask Integration Since this PoC only focuses on the backend functionality, integrating it with Flask will be the next step. However, integrating it into a web interface poses challenges such as managing session states, handling user inputs securely, and ensuring scalability for future development.

## 5. Next Steps

The following steps are necessary to complete the full implementation of the inventory management system:

1. **Flask Integration**: Integrate the current inventory system with a Flask web interface, where users can interact with the inventory system through a user-friendly web interface.
2. **Enhanced Error Handling**: Implement better error handling for edge cases, such as invalid input formats, empty inventory, and database issues.
3. **Persistent Storage**: Currently, the inventory is stored in memory. I plan to integrate a database, named mysql to persist inventory data between sessions.

4. **User Authentication**: Implement user authentication features to restrict access to the inventory system.
5. **Reporting and Analytics**: Add reporting capabilities, such as tracking inventory changes over time, low-stock alerts, and cost analysis.

## 6. Code Snippets and Documentation

For a detailed walkthrough of the code, please refer to my GitHub repository: https://github.com/rutus-code/Project_Inventory_Management_System

## 7. Conclusion:

This report provides an overview of the PoC implementation for an inventory management system using a Hash Map Data structure. The core functionality is working as expected, and the next phase will focus on integrating it into a web application with Flask and added the database design structure for my inventory Management System by including tables like category, inventory, products, users for ensuring my application is accessible to authorized users only.

## 8. References

1. Knuth, D. E. (2011). *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (4th ed.). Addison-Wesley.
2. Beazley, D. M. (2013). *Python Cookbook: Recipes for Mastering Python 3.* O'Reilly Media.
3. Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual.* Python Software Foundation.
4. GeeksforGeeks. (n.d.). *Hash map in Python*
5. Pagh, R., & Rodler, F. F. (2001). Cuckoo hashing. *Journal of Algorithms*, *51*(2), 122-144.