System Requirements Specification Index

For

Inventory Management System Console Application

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

TABLE OF CONTENTS

Inventory Management System Console

System Requirements Specification

1 PROJECT ABSTRACT

EZRetail Solutions, a retail consultancy based in Chicago, is developing a lightweight inventory management tool for small and mid-sized businesses. Many of their clients struggle with tracking inventory efficiently, leading to frequent stockouts and overstock issues. To address this, we are building a Python console application that processes inventory data using optimized loop implementations. The system will handle product filtering, searching, counting, and calculations through parallel lists, ensuring accurate and efficient inventory management. Designed for businesses with limited technical resources, this tool will help retailers streamline stock tracking, identify low-stock items, and generate key inventory insights without requiring complex software solutions.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. The application must handle batch processing of inventory data efficiently. 2. Products below a specified stock threshold must be identified. 3. The system should compute total inventory valuation. 4. Users should be able to search for products using partial name matches. 5. The application must produce various inventory reports based on category, low stock, and valuation.

3 CONSTRAINTS

3.1 INPUT REQUIREMENTS

1. Inventory Records:
   - Stored as a list of dictionaries in inventory_records
   - Each record contains: `product_id`, `name`, `category`, `quantity`, `price`.
   - Example: [{"product_id": "P001", "name": "Rice 5kg", "category": "Grocery", "quantity": 45, "price": 250.00}]
2. Low Stock Threshold:
   - Must be stored as integer in variable low_stock
   - Must be between 1 and 100
   - Example: 10
3. Search Term:
   - Must be stored as string in variable search_term
   - Example: "Rice"
4. Report Type:
   - Must be stored as integer in variable report_type
   - 1: Low stock report
   - 2: Category summary
   - 3: Valuation report
   - 4: Reorder list
   - Example: 2

3.2 CALCULATION CONSTRAINTS

1. Inventory Filtering (Low Stock):

- Must use a loop to iterate through inventory_records
- Must identify items where quantity <= low_stock
- Return list of low stock items

- Example: All items with quantity <= 10

2. Product Search:
   - Must use a loop to iterate through inventory_records
   - Must find items where search_term is in name (case-insensitive)
   - Return list of matching items
   - Example: search_term="rice", return all items with "rice" in name
3. Category Summary:
   - Must use a loop to iterate through inventory_records
   - Must group items by category and count them
   - Return dictionary with categories as keys and counts as values
   - Example: {"Grocery": 15, "Electronics": 8}
4. Inventory Valuation:
   - Must use a loop to iterate through inventory_records
   - Must calculate total value for each item (quantity * price)
   - Return total valuation
   - Example: Sum of all (quantity * price)
5. Reorder List:
   - Must use a loop with conditions to iterate through inventory_records
   - Skip items where quantity > low_stock
   - Calculate reorder amount (3 * low_stock - quantity)
   - Return reorder list with item name and reorder quantity
   - Example: [{"name": "Rice 5kg", "reorder": 25}]

3.3 Output Constraints

1. Display Format:

- Show "Inventory Management System"
- Show "Report Type: {report_name}"
- Show report contents in tabular format

2. Low Stock Report:
    - Table columns: ID, Name, Quantity, Category
    - Show count of low stock items
    - Show "Critical" for items with quantity < low_stock/2
3. Category Summary:
    - Table columns: Category, Item Count, % of Inventory
    - Sort by Item Count (descending)
    - Show percentages with 1 decimal place
4. Valuation Report:
    - - Table columns: Category, Items Count, Total Value
    - - Show grand total at bottom
    - - Show currency symbol (₹) for values
5. Reorder List:
    - - Table columns: ID, Name, Current Stock, Reorder Quantity
    - - Sort by Reorder Quantity (descending)
    - - Show estimated cost (Reorder Quantity * price)

4. TEMPLATE CODE STRUCTURE:

1. Processing Functions:

- find_low_stock_items(inventory_records, low_stock)
- search_products(inventory_records, search_term)
- summarize_categories(inventory_records)
- calculate_inventory_value(inventory_records)
- generate_reorder_list(inventory_records, low_stock)

2. Input Section:

- Load inventory data
- Get low stock threshold
- Get search term (if needed)
- Get report type

3. Processing Section:

- Execute appropriate processing function
- Format results for display

4. Output Section:

- Display report header
- Show tabular data
- Display summary statistics

5. DETAILED FUNCTION IMPLEMENTATION STRUCTURE

5.1 Core Inventory Functions

1. Write a Python function to find items with low stock levels. Define: `find_low_stock_items(threshold)`

The function should:

- Accept one parameter: `threshold` (integer between 1-100)
- Validate input parameter is not None: `if threshold is None:`
- Raise `TypeError` with message "Threshold cannot be None" for None input
- Validate input parameter is integer: `if not isinstance(threshold, int):`
- Raise `TypeError` with message "Threshold must be an integer" for non-integer input
- Validate threshold range: `if threshold <= 0 or threshold > 100:`
- Raise `ValueError` with message "Threshold must be between 1 and 100" for invalid range
- Initialize five empty result lists: `low_stock_ids = []`, `low_stock_names = []`, `low_stock_categories = []`, `low_stock_quantities = []`, `low_stock_prices = []`
- Use for loop to iterate through inventory: `for i in range(len(product_ids)):`
- Check condition for low stock: `if quantities[i] <= threshold:`
- Add corresponding data to all result lists: `low_stock_ids.append(product_ids[i])`, etc.
- Return tuple of all five lists: `return low_stock_ids, low_stock_names, low_stock_categories, low_stock_quantities, low_stock_prices`
- Example: find_low_stock_items(10) should return items with quantity ≤ 10

2. Write a Python function to search products by name. Define: `search_products(term)`

The function should:

- Accept one parameter: `term` (string for search term)
- Validate input parameter is not None: `if term is None:`
- Raise `TypeError` with message "Search term cannot be None" for None input
- Validate input parameter is string: `if not isinstance(term, str):`
- Raise `TypeError` with message "Search term must be a string" for non-string input
- Initialize five empty result lists: `matching_ids = []`, `matching_names = []`, `matching_categories = []`, `matching_quantities = []`, `matching_prices = []`
- Convert search term to lowercase: `search_term = term.lower()`
- Handle empty search term case: `if search_term == "":` return copies of all lists
- Use for loop to iterate through names: `for i in range(len(names)):`
- Check if search term is in product name (case-insensitive): `if search_term in names[i].lower():`
- Add corresponding data to all result lists when match found

- Return tuple of all five lists: `return matching_ids, matching_names, matching_categories, matching_quantities, matching_prices`
- Example: search_products("rice") should find products containing "rice" (case-insensitive)

3. Write a Python function to summarize items by category. Define: `summarize_categories()`

The function should:

- Take no parameters
- Create set of unique categories: `category_set = set(categories)`
- Convert set to list: `unique_categories = list(category_set)`
- Initialize empty list for counts: `category_counts = []`
- Use outer loop for each unique category: `for category in unique_categories:`
- Initialize counter: `count = 0`
- Use inner loop to count occurrences: `for i in range(len(categories)):`
- Check if current category matches: `if categories[i] == category:`
- Increment counter: `count += 1`
- Add count to results: `category_counts.append(count)`
- Return tuple of two lists: `return unique_categories, category_counts`
- Example: summarize_categories() should return (["Grocery", "Electronics"], [2, 1])

4. Write a Python function to calculate total inventory value. Define: `calculate_inventory_value()`

The function should:

- Take no parameters
- Initialize total value: `total_value = 0.0`
- Use for loop to iterate through inventory: `for i in range(len(quantities)):`
- Validate quantity data type: `if not isinstance(quantities[i], (int, float)):`
- Raise `TypeError` with message "Quantity must be a number" for invalid type
- Validate price data type: `if not isinstance(prices[i], (int, float)):`
- Raise `TypeError` with message "Price must be a number" for invalid type
- Validate quantity is non-negative: `if quantities[i] < 0:`
- Raise `ValueError` with message "Quantity cannot be negative" for negative values
- Validate price is non-negative: `if prices[i] < 0:`
- Raise `ValueError` with message "Price cannot be negative" for negative values
- Calculate item value: `item_value = quantities[i] * prices[i]`

- Add to total: `total_value += item_value`
- Return total value as float: `return total_value`
- Example: calculate_inventory_value() should return sum of all (quantity × price)

5. Write a Python function to generate reorder list. Define: `generate_reorder_list(threshold)`

The function should:

- Accept one parameter: `threshold` (integer for low stock threshold)
- Validate threshold is positive: `if threshold <= 0:`
- Raise `ValueError` with message "Threshold must be positive" for non-positive values
- Initialize two empty result lists: `reorder_names = []`, `reorder_quantities = []`
- Use for loop to iterate through inventory: `for i in range(len(product_ids)):`
- Check if item needs reordering: `if quantities[i] <= threshold:`
- Calculate reorder quantity using formula: `reorder_quantity = (3 * threshold) - quantities[i]`
- Add item name to reorder list: `reorder_names.append(names[i])`
- Add reorder quantity to list: `reorder_quantities.append(reorder_quantity)`
- Return tuple of two lists: `return reorder_names, reorder_quantities`
- Example: generate_reorder_list(10) calculates reorder quantity as (3×10) - current_quantity

5.2 Display and Utility Functions

6. Write a Python function to display formatted reports. Define: `display_report(report_type, data)`

The function should:

- Accept two parameters: `report_type` (string) and `data` (tuple or value)
- Validate report type: `if report_type not in ["low_stock", "search", "categories", "value", "reorder"]:`
- Raise `ValueError` with message "Invalid report type" for invalid types
- Validate data is not None: `if data is None:`
- Raise `TypeError` with message "Report data cannot be None" for None data
- Use conditional logic to format output based on report type
- For "low_stock" reports: unpack data tuple and display in tabular format
- For "search" reports: display search results in table format
- For "categories" reports: display category counts
- For "value" reports: display total inventory value

- For "reorder" reports: display reorder list with quantities
- Handle empty data gracefully with appropriate messages
- Use proper string formatting for alignment and readability
- Example: display_report("low_stock", data) should show formatted table of low stock items

7. Write a Python function to get validated integer input. Define: `get_valid_integer_input(prompt, min_value=None, max_value=None)`

The function should:

- Accept three parameters: `prompt` (string), `min_value` (optional integer), `max_value` (optional integer)
- Use infinite loop for input validation: `while True:`
- Get user input: `value = input(prompt)`
- Convert to integer with try-except: `value = int(value)`
- Handle ValueError for non-numeric input: print "Please enter a valid number."
- Validate minimum value if provided: `if min_value is not None and value < min_value:`
- Print appropriate error message and continue loop
- Validate maximum value if provided: `if max_value is not None and value > max_value:`
- Print appropriate error message and continue loop
- Return valid integer when all checks pass: `return value`
- Example: get_valid_integer_input("Enter threshold (1-100): ", 1, 100)

5.3 Main Program Structure

8. Write a Python main program for inventory management. Define: `main()` function

The function should:

- Print welcome message: `print("\nWelcome to the Inventory Management System!")`
- Use infinite loop for menu interaction: `while True:`
- Display menu options with numbered choices (1-6)
- Get user choice: `choice = input("\nEnter your choice (1-6): ").strip()`
- Use try-except block for error handling
- **For choice '1' (Low Stock Items):**
    - Print section header

    - Get threshold using `get_valid_integer_input("Enter low stock threshold (1-100): ", 1, 100)`
    - Call `data = find_low_stock_items(threshold)`
    - Call `display_report("low_stock", data)`
- **For choice '2' (Search Products):**
    - Print section header
    - Get search term: `search_term = input("Enter search term: ")`
    - Call `results = search_products(search_term)`
    - Call `display_report("search", results)`
- **For choice '3' (Category Summary):**
    - Print section header
    - Call `data = summarize_categories()`
    - Call `display_report("categories", data)`
- **For choice '4' (Total Value):**
    - Print section header
    - Call `total_value = calculate_inventory_value()`
    - Call `display_report("value", total_value)`
- **For choice '5' (Reorder List):**
    - Print section header
    - Get threshold using `get_valid_integer_input("Enter low stock threshold (1-100): ", 1, 100)`
    - Call `data = generate_reorder_list(threshold)`
    - Call `display_report("reorder", data)`
- **For choice '6' (Exit):**
    - Print "Thank you for using the Inventory Management System!"
    - Break from loop using `break`
- **For invalid choices:**
    - Print "Invalid choice. Please enter a number between 1 and 6."
- Handle general exceptions: `except Exception as e:` print error message
- Add pause for user interaction: `input("\nPress Enter to continue...")`
- Include proper program entry point: `if __name__ == "__main__": main()`
- Example: Program should provide continuous menu-driven interaction until user exits

5.4 Implementation Requirements

Key Technical Requirements:

Loop Structure Requirements:

- **find_low_stock_items()** must use single loop with conditional filtering
- **search_products()** must use single loop with string matching operations

- **summarize_categories()** must use nested loops (outer for unique categories, inner for counting)
- **calculate_inventory_value()** must use single loop with accumulation pattern
- **generate_reorder_list()** must use single loop with conditional processing

Data Validation Requirements:

- All functions must validate input parameters for None values
- Type validation for appropriate data types (int, str, float)
- Range validation for threshold values (1-100)
- Business rule validation (non-negative quantities and prices)
- Proper exception raising with descriptive messages

Parallel List Management:

- Input: Five parallel lists (product_ids, names, categories, quantities, prices)
- Processing: Index-based iteration maintaining list relationships
- Output: Tuples of parallel lists or calculated values
- Consistent indexing across all parallel operations

Error Handling Patterns:

- TypeError for None inputs and wrong data types
- ValueError for invalid ranges and negative values
- Exception handling in main program to prevent crashes
- User-friendly error messages for invalid inputs

Return Value Requirements:

- find_low_stock_items(): tuple of 5 lists (ids, names, categories, quantities, prices)
- search_products(): tuple of 5 lists (ids, names, categories, quantities, prices)
- summarize_categories(): tuple of 2 lists (unique_categories, category_counts)
- calculate_inventory_value(): float value representing total inventory worth
- generate_reorder_list(): tuple of 2 lists (reorder_names, reorder_quantities)

6. EXECUTION STEPS TO FOLLOW:

1. Run the program

2. Load or enter inventory data

3. Set low stock threshold

4. Select report type

5. Enter search term (if applicable)

6. View generated report

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.

- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

- This editor Auto Saves the code

- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page)

- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

- To launch application: python3 filename.py

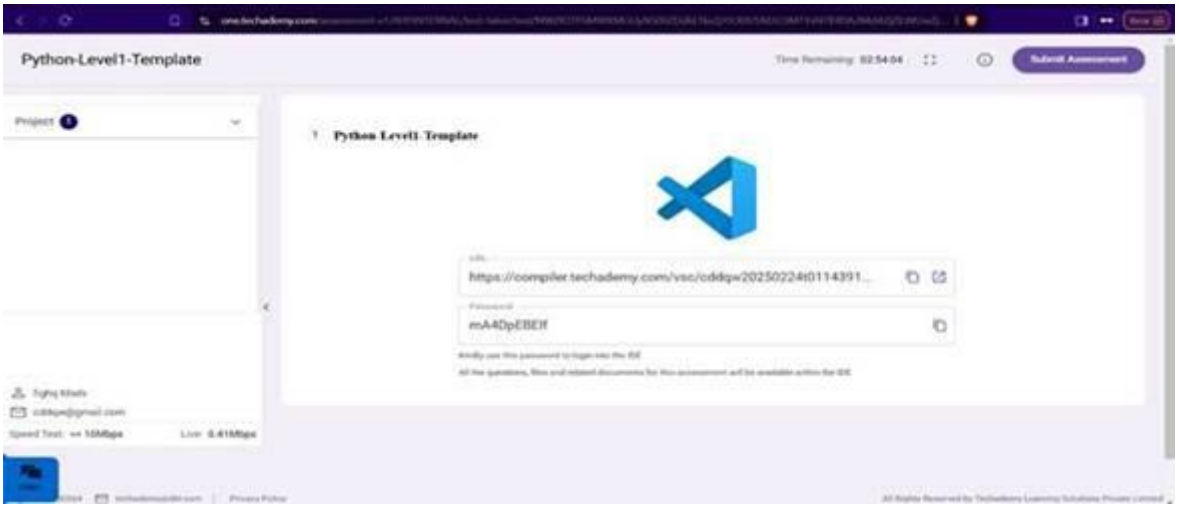
- To run Test cases: python3 -m unittest

Screenshot to run the program

To run the application

Python3 filename.py

To run the testcase python -m unittest



- Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.