System Requirements Specification Index

For

Food Waste Reduction System

Version 1.0

IIHT Pvt. Ltd.

fullstack@iiht.com

TABLE OF CONTENTS

Food Waste Reduction System

System Requirements Specification

1 PROJECT ABSTRACT

The Community Food Network (CFN) requires a specialized tracking system to monitor and reduce food waste across restaurants, grocery stores, and food banks. This assignment focuses on implementing modular functions to process food inventory data, track expiration dates, and identify donation opportunities. Each function will handle a specific aspect of data processing, following good programming practices like proper documentation, error handling, and return value consistency.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. System needs to process food inventory data through well-defined functions 2. Each function must handle a specific task (data validation, calculations, reporting) 3. Functions must include proper documentation (docstrings) 4. System must support Food item tracking and validation, Expiration date monitoring, Donation opportunity identification

3 CONSTRAINTS

3.1 INPUT REQUIREMENTS

1. Food Item Data:
    - Each food item record must include id, name, category, quantity, unit, expiration_date, storage_location
    - Example: `{"id": "F001", "name": "Apples", "category": "Produce", "quantity": 25, "unit": "kg", "expiration_date": "2023-12-15", "storage_location": "Cooler 3"}`
2. Donation Opportunity Data:
    - Each donation opportunity must include id, name, accepts_categories
    - Example: `{"id": "R001", "name": "City Food Bank", "accepts_categories": ["Produce", "Canned", "Bakery"]}`
3. Food Categories:
    - Must be one of: "Produce", "Dairy", "Bakery", "Meat", "Frozen", "Canned", "Dry Goods", "Prepared"
4. Date Format:
    - All dates must be in YYYY-MM-DD format (e.g., "2023-12-15")
5. Variable Names:
    - Food inventory must be stored in a variable named `inventory`
    - Recipients must be stored in a variable named `recipients`

3.2 FUNCTION CONSTRAINTS

1. Function Definition:

- Each function must have a clear purpose and responsibility
- Must use proper parameter naming
- Must include return value documentation
- Example: `def validate_food_item(food_item):`

2. Docstrings:
    - Each function must include a docstring describing:
    - Purpose of the function
    - Parameters (name, type, description)
    - Return value (type, description)
    - Example usage
3. Error Handling:
    - Functions must handle invalid inputs appropriately
    - Must return appropriate error messages or raise exceptions
    - Example: `if not isinstance(quantity, (int, float)) or quantity < 0: raise ValueError("Quantity must be a positive number")`
4. Parameter Validation:
    - Functions must validate parameter types and values
    - Example: `if not isinstance(food_id, str): raise TypeError("Food ID must be a string")`
5. Return Values:
    - Functions must return consistent data types
    - Validation functions must return validation result
    - Example: `return {"is_valid": True, "message": "Food item data is valid"}`

3.3 Output Constraints

1. Display Format:
    - Functions must return properly formatted data
    - Display functions must format output appropriately
    - Example: `return f"{food_item['name']} - {food_item['quantity']} {food_item['unit']} - Expires: {food_item['expiration_date']}"`

2. Output Format:
    - Console output must follow this format:
    - Show "== FOOD WASTE REDUCTION SYSTEM =="
    - Show function results in clearly labeled sections
    - Format quantities with appropriate units
    - Display dates in YYYY-MM-DD format

4. TEMPLATE CODE STRUCTURE:

1. Data Management Functions:
    - `validate_food_item(food_item)` - validates food item data structure
    - `calculate_days_until_expiration(expiration_date)` - calculates days until expiration
2. Data Processing Functions:
    - `identify_expiring_items(food_items, days_threshold)` - identifies items expiring soon
    - `sort_items_by_expiration(food_items)` - sorts items by expiration date
    - `match_donations(food_items, recipients)` - matches food items with recipients
3. Helper Functions:
    - `format_food_item(food_item)` - formats food item information for display
4. Program Control Functions:

- `main()` - main program function demonstrating all other functions

5. Detailed Function Structure:

5.1 Data Validation Functions

1. Write a Python function to validate food item data structure.

Define: **validate_food_item(food_item)**
This function should:

- Example: `{"id": "F001", "name": "Apples", "category": "Produce", "quantity": 25, "unit": "kg", "expiration_date": "2023-12-15", "storage_location": "Cooler 3"}`

- Check if the input is a dictionary and not None
- Verify all required fields are present: "id", "name", "category", "quantity", "unit", "expiration_date", "storage_location"
- Validate data types:
  - id: must be a string
  - name: must be a string
  - category: must be a string and one of the valid categories
  - quantity: must be a positive number (int or float, >= 0)
  - unit: must be a string
  - expiration_date: must be a string
  - storage_location: must be a string
- Check that category is one of: ["Produce", "Dairy", "Bakery", "Meat", "Frozen", "Canned", "Dry Goods", "Prepared"]
- Return a dictionary with format: {"is_valid": bool, "message": str}
- Return appropriate error messages for specific validation failures
- Example return: {'is_valid': True, 'message': 'Food item data is valid'}

2. Write a Python function to calculate days until expiration.

Define: **calculate_days_until_expiration(expiration_date)**

This function should:

- Accept an expiration date string in YYYY-MM-DD format
- Parse the date using datetime.strptime() with format "%Y-%m-%d"

- Calculate the difference between expiration date and current date using datetime.date.today()
- Return the number of days as an integer
- Return -1 for invalid date formats or None input
- Handle ValueError exceptions for malformed dates
- Positive numbers indicate future dates, negative numbers indicate past dates
- Example: if today is 2023-12-10 and expiration_date is "2023-12-15", return 5

3.2 Data Processing Functions

3. Write a Python function to identify food items expiring soon.

Define: **identify_expiring_items(food_items, days_threshold=7)**

This function should:

- Accept a list of food item dictionaries and a days threshold (default 7)
- Validate that food_items is a list and days_threshold is an integer
- Return empty list for invalid inputs
- Iterate through each food item in the list
- Skip items that don't have an "expiration_date" field
- Use calculate_days_until_expiration() to get days until expiration for each item
- Include items where days until expiration is between 0 and days_threshold (inclusive)
- Return a list of food items that will expire within the threshold
- Only include items with valid expiration dates (days >= 0)
- Example: items expiring in 0-7 days should be included in results

4. Write a Python function to sort food items by expiration date.

Define: **sort_items_by_expiration(food_items)**

This function should:

- Accept a list of food item dictionaries
- Return empty list if input is None or not a list
- Create a copy of the input list to avoid modifying the original
- Sort items by expiration_date field in ascending order (earliest dates first)
- Use the sorted() function with a lambda key function

- Handle missing expiration_date fields by using a default far-future date "9999-12-31"
- Return the sorted list of food items
- Items with earlier expiration dates should appear first in the result
- Example: item expiring 2023-12-10 should come before item expiring 2023-12-15

5. Write a Python function to match food items with donation recipients.

Define: **match_donations(food_items, recipients)**

This function should:

- Accept a list of food items and a list of recipient organizations
- Return empty list if either input is None or not a list
- Iterate through each food item
- Skip items that don't have a "category" field
- For each item, find the first recipient that accepts its category
- Check if item's category is in recipient's "accepts_categories" list
- Skip recipients that don't have "accepts_categories" field
- Create match dictionaries with format: `{"item": item_dict, "recipient": recipient_dict}`
- Each item should match with only the first suitable recipient found
- Return a list of all successful matches
- Items without matching recipients should not appear in results

5.3 Helper Functions

6. Write a Python function to format food item information for display.

Define: **format_food_item(food_item)**

This function should:

- Accept a food item dictionary
- Return "Invalid food item format" for None input or non-dictionary input
- Extract and format key information: id, name, quantity, unit, category, expiration_date
- Handle missing fields gracefully by returning error message
- Use try-except block to catch KeyError exceptions

- Return formatted string in this exact format: `"ID | Name | Quantity Unit | Category | Expires: Date"`
- Example output: `"F001 | Apples | 25 kg | Produce | Expires: 2023-12-15"`
- All fields should be separated by " | " (space-pipe-space)

5.4 Main Program Function

6. Write a Python function to demonstrate the food waste reduction system.

Define: **main()**

This function should:

- Print the system header: "===== FOOD WASTE REDUCTION SYSTEM ====="
- Use the provided sample data for inventory and recipients (DO NOT MODIFY)
- Demonstrate each function in numbered sections:
  - Validate all food items and print results
  - Calculate and display days until expiration for each item
  - Identify and display items expiring within 7 days
  - Sort and display items by expiration date
  - Find and display donation matches
  - Format and display all food items
- Print clear section headers for each demonstration
- Show meaningful output that demonstrates each function's capabilities
- Handle the sample data provided in the skeleton code

6. Execution Steps to Follow:

1. Run the program

2. Observe the execution of each function

3. View validation results

4. See expiration date calculations

5. View expiring items identification

6. Observe donation matching algorithm

7. See formatted output

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) then you need to use CTRL+Shift+B -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.

- 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**

- Before Final Submission also, you need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

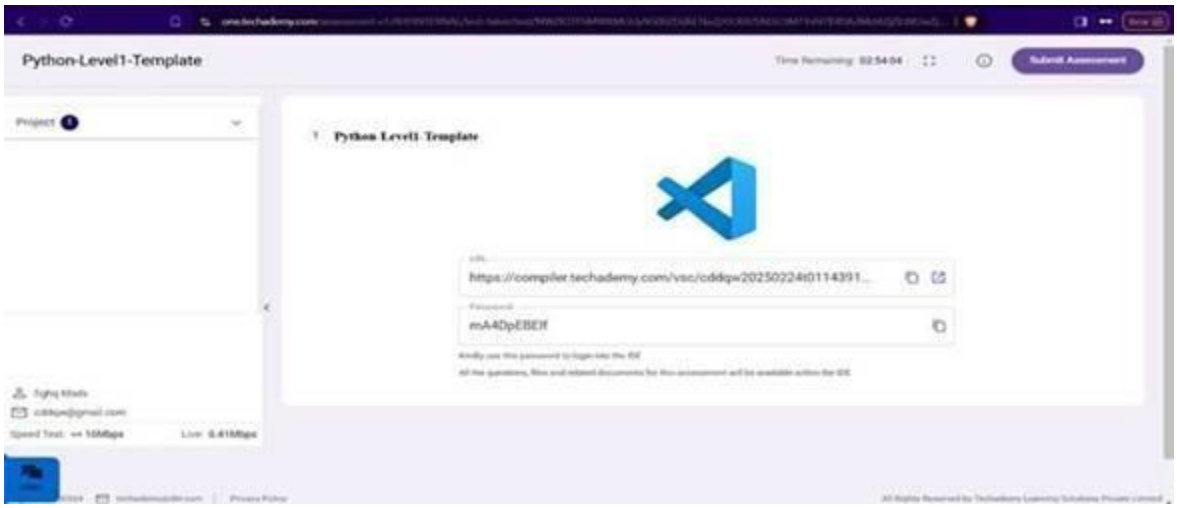
Screen shot to run the program

To run the application

Python3 filename.py

To run the testcase

python3 -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.