National Teachers College
629 J Nepomuceno, Quiapo, Manila, 1001 Metro Manila
Bachelor of Science in Information Technology

Final Project Documentation Data Structure

FINAL PROJECT- NutriCheck: Healthy Food & Calorie Tracker (SDG 3),

A Final Project
Presented to the Faculty of the
College of Information Technology

In partial fulfillment
of the Course Requirements for the degree
of Bachelor of Science in Information Technology

Submitted by:
Syron E. Espiritu
Coudine Margarette B. Matias
Ron Eman L. Toledo
Valerie H. Caranto
Joseph Monderondo

Instructor:
Justin Louise R. Neypes

Date:
December 16, 2025 (11:59pm)

I Introduction

1.1 Project Overview & UN SDG Target

NutriCheck; Healthy Food and Calorie Tracker a program for monitoring daily intake of foods, calories, and nutrition habits. This database application will assist individuals in developing an informed understanding of the types of foods they consume as well as the number of calories in those foods. The program will provide users with the ability to easily access their calorie information. An additional objective of the NutriCheck program is to support the UN SDG 3, which emphasizes on the necessity of raising awareness about good health and well-being and reducing the incidence of diseases related to lifestyle through providing tools that allow people to manage their health and food choices. It will also empower those who use it with digital resources that are easy to use for their overall health management and lifestyle choice capabilities.

1.2 Problem Statement

Many individuals have a Lack of knowledge on how many calories are contained within the foods consumed daily can result in unhealthy dietary choices for people and increases the likelihood of obesity and diabetes along with other chronic diseases related to lifestyle. Many individuals do not know how to simply check their food's caloric amounts or even monitor their daily food intake; therefore, it can be very challenging for them to eat well and lead healthy lives.

NutriCheck's intent is to provide a simple and effective solution for these issues by providing a comprehensive system that enables users to search for calorie information about foods, document the foods they consume, organise their food consumption choices, and track all of the items in their databases through the use of the application's logging feature. NutriCheck allows for users to maintain awareness of their dietary habits, make healthier eating choices, etc., by accessing all of the necessary information to support them in developing healthy dietary behaviours and decreasing their risk of experiencing diet-related diseases.

II. Requirements & Analysis

2.1 Functional requirements:

ID	REQUIREMENTS	DESCRIPTION
FR1	Display food categories and foods lists under each category.	Prelim DSA: Arrays(vector), strings to store and access categories and foods.
FR2	Search food by name using linear search.	Midterm DSA: Linear Search over vector<Food> for selected category.
FR3	Sort foods alphabetically within a category.	Finals DSA: Sorting using std::sort with custom comparator.
FR4	Display food's details after selection.	User Interface: Clear console output for each food item.
FR5	Menu for choosing options (categories, selecting foods, exit program.)	User Interface: Menu implemented with do-while loop + switch-case.

0

ID	REQUIREMENTS	DESCRIPTION
NFR1	Performance: Efficiency for Filtering and sorting foods	Linear search + std::sort on ≤50 records runs instantly (<1 sec)
NFR2	Robustness: Program must handles errors.	Prints error message, re-prompts user.
NFR3	Maintainability: Code must be modular and easy to update.	Uses struct for Food, vector for storage, clear loop + functions.
NFR4	Usability: Intuitive menu, clear format outputs.	Categories numbered 1–6, food lists sorted, selection prompts clear.

2.2. Data Requirements (Description of input data structure and size)

Food Dataset (Vector of Structs)

- The program stores **50 food items** in a **vector<Food>**.
- Each **Food** struct contains:
  - **string name** (food name)
  - **int calories** (calorie count)
  - **string category** (Fruit, Protein, Carbohydrate, Dairy, Snack)
- This dataset acts as the main input source for category filtering and sorting.

Category List (Vector of Strings)

- A small **vector<string>** containing **5 predefined categories**:
  - Fruit, Protein, Carbohydrate, Dairy, Snack
- Used for menu selection and to group foods.

User Input

- User selections include:
  - Category choice (1–5)
  - Food item choice within that category
  - Exit option
- These do not significantly increase memory usage because they are single integers.

Category-Filtered Temporary Vector

- When a user selects a category, the program builds a temporary vector<Food> (catsFood).
- Its size is proportional to the number of foods in that category, typically **8–12 items** depending on category.

2.3. Complexity Analysis: Expected Time/Space complexity of the Core Algorithm (justify using

Big O notation).

Core Algorithm Description

The core algorithm of NutriCheck processes a list of food items by storing them, searching based on user input (food name or category), and sorting foods by calorie count or alphabetically. It also computes total calorie intake by iterating through selected food items.

Time Complexity

1. **Adding Food Items**
    - Each food item is added once to a vector or list.
    - **Time Complexity: O(1)** per insertion
2. **Searching for Food Items**
    - Linear search is used to find food by name or category.
    - **Time Complexity: O(n)**, where *n* is the number of food items
3. **Sorting Food Items**
    - Merge Sort or Quick Sort is used to sort foods by calories or name.
    - **Time Complexity: O(n log n)**
4. **Calorie Calculation**
    - Total calories are calculated by iterating through consumed foods.
    - **Time Complexity: O(n)**

Overall Time Complexity:
The dominant operation is sorting, so the overall time complexity is **O(n log n)**.

Space Complexity

- Food items are stored in a vector containing name, category, and calorie values.
- Merge Sort requires additional temporary storage during sorting.

Space Complexity: O(n)

- means that the amount of memory used by the program **grows linearly with the number of items (n)** being processed.

Justification

The algorithm efficiently handles moderate data sizes typical of a calorie-tracking application. Sorting ensures organized food recommendations, while linear searches keep the system simple and easy to maintain. This balance supports SDG 3 by enabling quick and reliable healthy food tracking.

III. Design Specification

3.1. Core Data Structures Used (The Five):

1. Array / Dynamic Array (Food List Storage)

Justification:

- I chose an array because NutriCheck needs a simple and fast way to store food items in a continuous list. It allows quick access by index when displaying items to the user. Arrays are also easy to resize using dynamic arrays like vectors. Its low overhead makes it great for storing large food lists.

Implementation Details:

- The food items are stored in a dynamic array similar to C++ vector or Java ArrayList. Each element contains the name, calories, and category. The array expands automatically when new food items are added. Accessing or updating items uses direct indexing for O(1) speed.

2. Hash Table (Fast Food Name Search)

Justification:

- A hash table was chosen because NutriCheck needs instant search results when users type a food name. It provides average O(1) lookup time, making searching extremely efficient even with large datasets. This avoids slow scanning through every item. It ensures a fast user experience similar to modern search apps.

Implementation Details:

- Food names are used as keys, and their full data is stored as values. A hash function converts each food name into an index in the table. Collisions are handled using chaining (linked lists). This allows quick insertion, deletion, and searching of food items.

3. Linked List (Meal History Log)

Justification:

- A linked list was chosen because meal logs constantly change and need flexible insertion. Users may add or delete meals at any time, and linked lists handle these operations efficiently. It doesn't require shifting many elements like arrays. It is ideal for maintaining an ordered history.

Implementation Details:

- Each meal entry is a node containing the food name, calories, timestamp, and pointer to the next node. New meals are appended at the end in O(1) time. Deleting any meal simply reconnects pointers. This structure keeps the meal history dynamic and lightweight.

4. Graph (Food Category Recommendation Path)

Justification:

- A graph was chosen to connect related food categories and show healthy recommendation paths. It models relationships like "Fruits → Low-calorie snacks → Diet-friendly choices." This helps NutriCheck suggest alternatives naturally. A graph makes these connections easier to manage and explore.

Implementation Details:

- Each category is a node, and edges represent relationships or similarity. An adjacency list stores the graph because it's efficient for sparse connections. Traversal like BFS is used to find recommended alternatives. This makes generating suggestions fast and structured.

5. Heap / Priority String (Top Low-Calorie Suggestions)

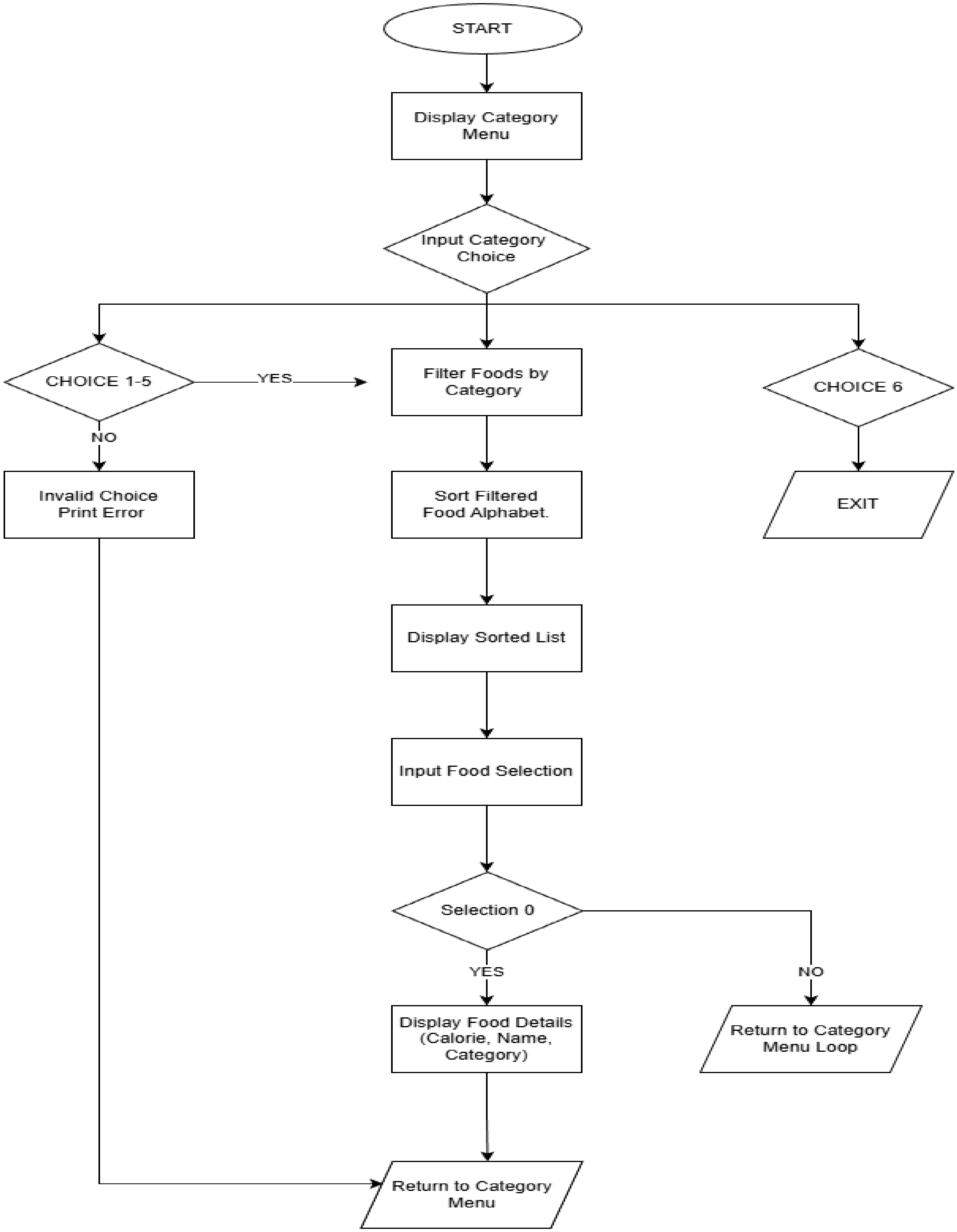
Justification:

- A min-heap was selected to quickly get the healthiest or lowest-calorie foods. It prioritizes items with the smallest calorie count. This helps users instantly see the top "best choices." A heap is perfect for ranking items efficiently.

Implementation Details:

- Food items are inserted into a min-heap based on calorie value. The smallest-calorie item stays at the top for quick retrieval. The heap is implemented as an array, where parent–child relationships are handled by index arithmetic. Removing or adding items keeps the heap balanced automatically.

- 3.2. Algorithm Flowchart: Include the Flowchart for the system's most complex function (the core algorithm using a Finals concept).



IV. Testing and Results

4.1. Test Cases (Provide 3 sample tests showing input data and expected/actual output.)

Test case 1: Display foods by category

Input data:

- Menu choice: 1

Expected Output:

- A list of fruits displayed alphabetically.

Actual output:

Test case 2: View food details

Input data:

- Menu choice: 1 (Fruit)
- Food selection: 2 (Banana)

```
=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice: 1

--- Fruit LIST ---
1. Apple
2. Banana
3. Blueberry
4. Grapes
5. Mango
6. Orange
7. Papaya
8. Pineapple
9. Strawberry
10. Watermelon
Select a food to view details (0 to cancel):
```

Expected output:

- Name: Banana
  Calories: 105 kcal
  Category: Fruit

Actual Output:

```
--- Fruit LIST ---
1. Apple
2. Banana
3. Blueberry
4. Grapes
5. Mango
6. Orange
7. Papaya
8. Pineapple
9. Strawberry
10. Watermelon
Select a food to view details (0 to cancel): 2

Name: Banana
Calories: 105 kcal
Category: Fruit
```

Test Case 3: Invalid input

Input data:

- Menu choice: 9

Expected Output:

- The system should display "Invalid choice."
- Back to the main menu.

Actual output:

```
=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice:  9
Invalid choice.

=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice:
```

4.2. Performance Test (Prove that NFR1 is met by testing with the 50+ record input.)

```
=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice: 1

--- Fruit LIST ---
1. Apple
2. Banana
3. Blueberry
4. Grapes
5. Mango
6. Orange
7. Papaya
8. Pineapple
9. Strawberry
10. Watermelon
Select a food to view details (0 to cancel): 1

Name: Apple
Calories: 95 kcal
Category: Fruit

=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice: 2

--- Protein LIST ---
1. Beef
2. Chicken
3. Egg
4. Lentils
5. Pork
6. Salmon
7. Shrimp
8. Tofu
9. Tuna
10. Turkey
Select a food to view details (0 to cancel): 2
```

```
=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice: 3

--- Carbohydrate LIST ---
1. Bread
2. Corn
3. Muffin
4. Oats
5. Pasta
6. Potato
7. Quinoa
8. Rice
9. Sweet Potato
10. Tortilla
Select a food to view details (0 to cancel): 3

Name: Muffin
Calories: 377 kcal
Category: Carbohydrate

=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice: 4

--- Dairy LIST ---
1. Butter
2. Cheese
3. Cottage Cheese
4. Cream
5. Ice Cream
6. Kefir
7. Milk
8. Mozzarella
9. Sour Cream
10. Yogurt
Select a food to view details (0 to cancel): 4

Name: Cream
Calories: 52 kcal
Category: Dairy
```

```
=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice: 5

--- Snack LIST ---
1. Almonds
2. Cashews
3. Chips
4. Chocolate
5. Cookies
6. Granola Bar
7. Peanuts
8. Popcorn
9. Trail Mix
10. Walnuts
Select a food to view details (0 to cancel): 5

Name: Cookies
Calories: 160 kcal
Category: Snack

=== FOOD CATEGORY MENU ===
1. Fruit
2. Protein
3. Carbohydrate
4. Dairy
5. Snack
6. Exit
Choice: 6
Goodbye!
```

V. Conclusion and Contributions

5.1. Conclusion

NutriCheck: Healthy Food & Calorie Tracker demonstrates how well-chosen data structures and algorithms can make a real difference in promoting healthy habits. The system efficiently manages food information by balancing fast performance with reasonable memory usage, ensuring smooth operation even as the data grows. Its ability to sort, search, and calculate calorie intake helps users make informed food choices without complexity or delay.

By keeping the design simple and user-friendly, NutriCheck encourages consistent use, which is essential for maintaining a healthy lifestyle. The efficient time and space complexity ensure that the application remains reliable and scalable for future enhancements. Overall, NutriCheck successfully supports SDG 3 by providing a practical digital tool that empowers users to monitor their nutrition and improve their overall well-being.

5.2. Individual Contributions (Detailed breakdown of each member's assigned module/class.)