FACULTY OF COMPUTING
UTM Johor Bahru

SECJ1023 – PROGRAMING TECHNIQUE 2 ||

Semester 2 – 2023/2024

Lecturer: Dr. Lizawati Binti Mi Yusuf

Group: 11

Name	Matric No.
Muhammad Zayyad Bin Badrul Hisham	A23CS3015
Mohamed Omar Makhlouf	A23CS4014
Mahmoud Mustafa Elganzory	A23CS0291

Section: 04

Problem Analysis and Design Phase 2

Section B: Problem Analysis:

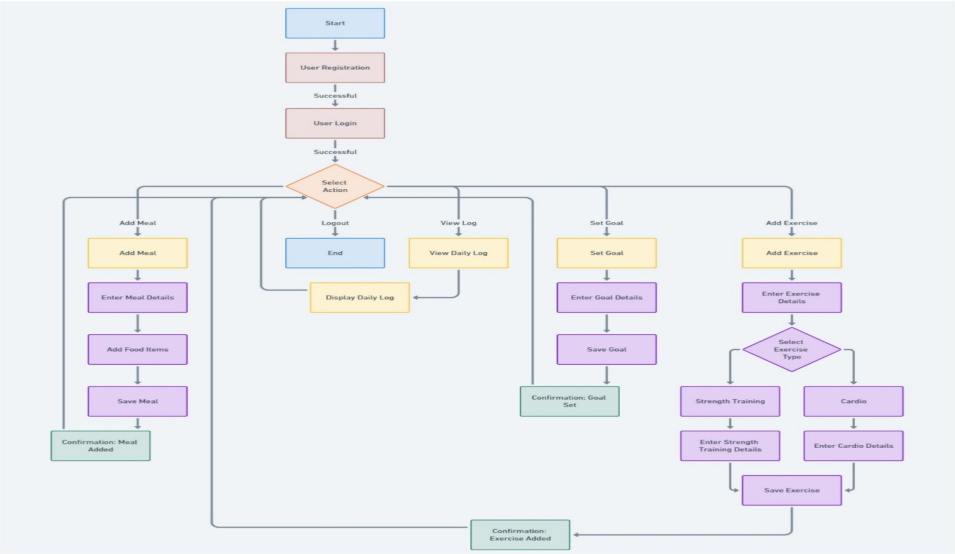
First let’s have an overview about the project:

As a human it is by default that we crave and love foods, it is essential and necessity for every person on earth. Hence, one of the reasons that Nutritional Tracker System is developed is to help and guide users monitor their calorie intake as well as achieving their dietary goals and ideal weight. The Nutritional Tracker System is built with multiple features that surely will help users to not only achieve their dietary goals but also have fun while monitoring their health.

Section A: Flow Chart

The workflow combining Input from the users and expected output from the system:

Based On Flow Chart diagram:



The "Nutrition Tracker Flowchart" illustrates the complete journey of a user interacting with a fitness tracking application. The process starts with creating a new user profile, where users can enter their personal details. From there, the flowchart guides us through how users can manage their meals and exercises.

Expected inputs and outputs

Analyzing of Flow chart:

Users can log their meals, specifying food items and their nutritional details. They can also track their exercise routines, detailing the type of exercise, duration, and calories burned. The system then calculates the total calorie intake and allows users to review this information. Additionally, users can set and manage their fitness goals, updating their progress and reviewing their achievements. Throughout this process, the flowchart highlights key decision points, such as adding or removing items, and options to view or print the logged information. Overall, the flowchart provides a clear and organized view of how users can navigate through the various features of the application, ensuring they can efficiently manage their health and fitness data.

Here is a detailed Table to elaborate more:

Category	Details
User inputs	User Profile Creation: **Personal details: userID, name, age, height, weight Optional details: gender, activity level**
	Meal Logging: **Meal details: mealName, food items (name, calories, sugar, sodium, protein, carbs, fats), Food item count**
	Exercise Logging: **Exercise details: exerciseName, exerciseType, duration, caloriesBurnt, Cardio: intensity, Strength training: setsCount, repsCount, weight**
	Goal Setting: **Goal details: goalName, startDate, goalStatus**
User outputs	Profile Information: **Display user personal details, Display calculated BMI**
	Meal Information: **Summary of logged meals, Nutritional breakdown, Total daily calorie intake**
	Exercise Information: **Summary of logged exercises, Detailed exercise session information, Total calories burned**
	Goal Information: **List of user goals, Progress, Status of goals (active or completed)**
System expectations	Data Validation: **Validate user inputs (non-negative or empty value), Check required fields during profile creation, meal logging, and exercise logging**
	Calculations: Calculate BMI: BMI= $\frac{weight\ (kg)}{height\ (m))2}$ BMI **Calculate total daily calorie intake, calculate total calories burned**
	Data Management: **Efficient storage of profiles, logs, and goals, Allow updates and deletions**
	User Interface: **User-friendly data input and review interface, Options for adding, removing, and viewing logs, Easy navigation**
	Feedback and Reporting: **Real-time feedback on daily calorie balance, generate fitness and nutrition reports, Print or export data**

Section B: Problem Analysis Using OOP Approach

First Identifying the Objects and Classes: Based on Our UML diagram for the system:

1. **User**

- **Attributes**: **“userID”**, **“name”**, **“age”**, **“height”**, **“weight”**, **“bmi”**, **“password”**, **“userLog”**, **“userGoal”.**

- **Methods**: **User (string, int, double, double)**, **getName()**, **getAge()**, **getHeight()**, **getWeight()**, **getBMI()**, **setPassword(string)**, **displayInfo()**

2. **Meal**

- **Attributes**: **“mealName”**, **“foodItems”**, **“foodItemsCount”**, **“foodItemsCapacity”**

- **Methods**: **Meal(string, int, int)**, **getMealName()**, **getFoodItems()**, **getFoodItemsCount()**, **getFoodItemsCapacity()**, **printMealInfo()**, **addFoodItem()**, **calculateCalorie()**

3. **Goal**

- **Attributes**: **“goalName”**, **“goalDesc”**, **“startDate”**, **“goalStatus”**

- **Methods**: **Goal(string, string, tm*)**, **getGoalName()**, **getGoalDesc()**, **getStartDate()**, **getGoalStatus()**, **setGoalStatus(bool)**

4. **FoodItem**

- **Attributes**: **“name”**, **“calories”**, **“sugar”**, **“sodium”**, **“protein”**, **“carbs”**, **“fats”**

- **Methods**: **FoodItem()**, **FoodItem(&FoodItem)**, **setName()**, **setKcal()**, **setCarb()**, **setProtein()**, **setFat()**, **setSugar()**, **setSodium()**, **addFood()**, **printItem()**, **getName() const**, **getKcal() const**, **getCarb() const**, **getProtein() const**, **getFat() const**, **getSugar() const**, **getSodium() const**, **getFoodItemInfo()**

5. **DailyLog**

- **Attributes**: **“data”**, **“meals”**, **“exercise”**

- **Methods**: **DailyLog()**, **addMeal()**, **addExercise()**, **getData()**, **printAllMeals()**, **printAllExercises()**, **displayLogs()**

6. **UserRecord**

- **Attributes**: **“userProfiles”**, **“userProfileCount”.**

- **Methods**: **UserRecord()**, **addUser(UserProfile*)**, **removeUser(UserProfile*)**

7. Cardio

- **Attributes**: “**intensity**”, “**distance**”
- **Methods**: **Cardio(string, double)**, **getIntensity()**, **getDistance()**, **setIntensity(string)**, **setDistance(double)**

8. Exercise

- **Attributes**: **exerciseName**, **exerciseType**, **timeSpent**, **calorieBurnt**

- **Methods**: **Exercise()**, **getExerciseName()**, **getExerciseType()**, **getTimeSpent()**, **getCalorieBurnt()**, **getExerciseDetail()**, **setExerciseName(string)**, **setExerciseType(string)**, **setTimeSpent(double)**, **setCaloriesBurnt(double)**

9. StrengthTraining

- **Attributes**: **setsCount**, **repsCount**, **weight**

- **Methods**: **StrengthTraining(int, int, double)**, **getSetsCount()**, **getRepsCount()**, **getWeight()**, **setSetsCount(int)**, **setRepsCount(int)**, **setWeight(int).**

Second **identifying what relation classes have:**

RELATION	DETAILS
Inheritance Relationship	Cardio and Exercise: Justification: **Cardio is a type of exercise that requires specific attributes like intensity and distance while also should have similar properties like exercise name and calorie burnt, hence Cardio is created as subclass of Exercise.** Type of Inheritance: Public **Inheritance - Cardio inherits from Exercise where Cardio object have access to all the Exercise public members**.
	StrengthTraining and Exercise: Justification**: Strength training is a type of exercise with specific attributes like sets, reps, and weight while also required similar properties with the Exercise Class such as exercise name and calorie burnt, hence like Cardio, StrengthTraining is created as subclass of Exercise.** Type of Inheritance: **Public inheritance - StrengthTraining inherits from Exercise where StrengthTraining object have access to all the Exercise public members.**
	User and DailyLog: Justification: Each user will need to maintain a log of daily activities and meals, to accommodate User and DailyLog objects, User class needs to have a property that could store all the DailyLog objects such as an array of DailyLog. Hence User will need to have a composition association with DailyLog, where dailyLog object will be created daily to record user meals and exercise under the User.

Association Relationships	Type of Association: Composition- A User contains array of DailyLog object as part of its attributes, where dailyLog objects cannot exist without users.
	User and Goal: Justification: Each user will have a set of specific fitness goals; to associate goal and user objects, users need to have properties consisting of array of goals object. Hence that is why the User will have a composition association with Goal class, where goals object will be create under the user. Type of Association: Composition- A User contains an array of Goal objects as part of its attributes where Goal objects cannot exist without user.
	Meal and FoodItem: Justification: Each meal is composed of multiple food item objects. Meal class will have a property like an array of FoodItem objects, which logically could also exist in other meals object. Hence Meal has an aggregation association with FoodItem, where meals will contains a reference to the FoodItem objects. Type of Association: Aggregation- A Meal is made up of FoodItems array and each FoodItem object can exist without being part of a Meal.
	DailyLog and Meal: Justification: Each daily log will contain multiple meals User consumes for the day, to facilitate the relationship between DailyLog and Meal. DailyLog class should contains a list of Meal user consumed. This Meal could already exist without the DailyLog class as it doesn't have any specific relationship. Hence DailyLog will have an aggregation association with

Association Relationships	Meal class, where DailyLog contains a reference to the Meal objects. Type of Association: Aggregation - A DailyLog contains Meals as part of its attributes, but Meal can exist independently of DailyLog
	DailyLog and Exercise: Justification: Each daily log contains multiple exercises users do for the day. Similar to a Meal class, Exercise object could already exist, as both classes doesn't have specific relationship requirement. Hence DailyLog has an aggregation association with Exercise class, where DailyLog will contains a reference of Exercise class. Type of Association: Aggregation - A DailyLog contains array of exercises as part of its attributes, but Exercise can exist independently of DailyLog.
	UserRecord and User: Justification: A user record will maintain profiles of all the users in the system, hence UserRecord needs to contain an array of User objects, User object can be created without being recorded, which shows that UserRecord will have an aggregation association with User class, where UserRecord will contains a pointer to User objects. Type of Association: Aggregation - UserRecord contains an array of User objects as part of its attributes, but Users object can exist independently of UserRecord.

Justification for Relationships:

Association Relationships:

These are chosen based on real-world relationships where one class logically contains or is composed of objects of another class but both classes can exist independently in some cases (example: a User can exist without DailyLog but uses DailyLog to track activities).

Inheritance Relationships:

These are chosen based on the relationship where a subclass is a unique version of the base class while also sharing some similarities. For example, Cardio is a specific type of Exercise with additional properties specific to cardio exercises.

Section C: Class Diagram