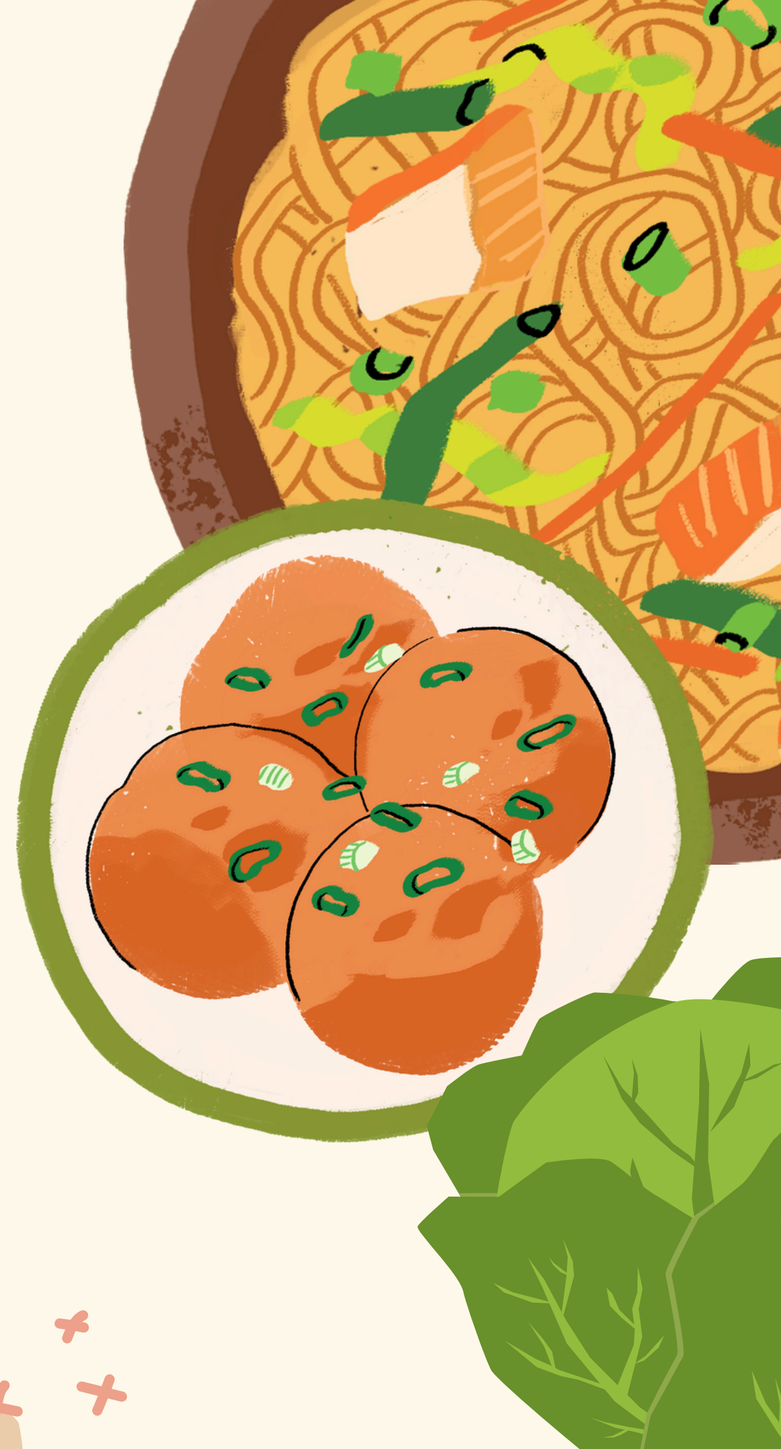
2E1i

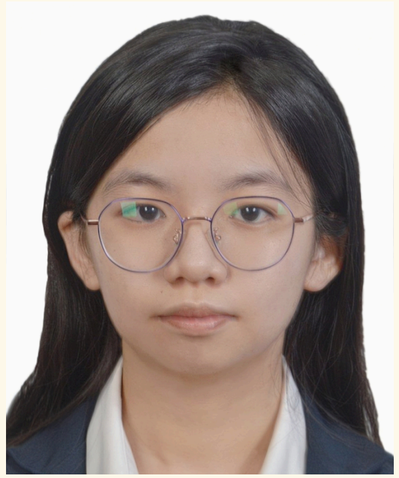
NUTRITION TRACKER SYSTEM



Our team :)

OW YEE HAO
A23CS0261

CHANG WEI LAM
A23CS0212

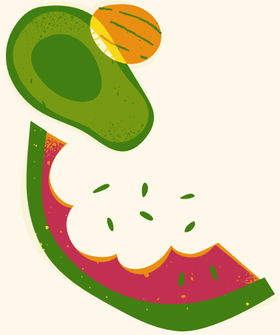


YAP JIA XIN
A23CS0199

WHAT IS THE OBJECTIVE AND PURPOSE OF OUR SYSTEM?

Personalized calories goals

based on :
age, weight, gender, and activity level



Daily consumption tracking

calories, macronutrients (carbohydrates, fats, proteins) and fiber

Tracking Report

on users' progress towards their calories intake and health conditions

Professional Feedback

from nutritionists to optimize users' dietary habits

PROGRAM DESIGN

- User Registration / Login
- Profile Setup
- Goal Setting
- Daily Tracking
- Feedback

USER REGISTRATION , LOGIN

Regular Users

- Sign up for an account
- Providing basic information :
  - age
  - gender
- Existing users can log in with their credentials.

Nutritionist

- Register for specialized accounts
  - additional functionalities
  - provide guidance
- Existing nutritionists can log in with their credentials

PROFILE SETUP

User Profile

- Users set up their profiles by entering relevant details :
  - height, weight, and activity level.

- To determine suggested nutrition goals tailored to users' lifestyle.

GOAL SETTING

Users have the option to set personalized nutrition goals, such as calorie intake targets.

The system provides a recommended daily calorie budget based on users' individual information.

- ✓ Recommended daily calories intake
- ✓ Personalized nutritional goal

DAILY TRACKING

FOOD INTAKE

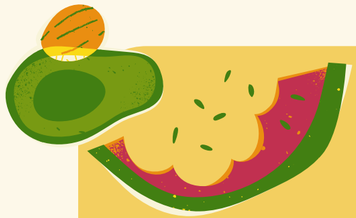
Daily calorie intake

PHYSICAL ACTIVITY

Calorie burnt

How to use the system

DAILY TRACKING



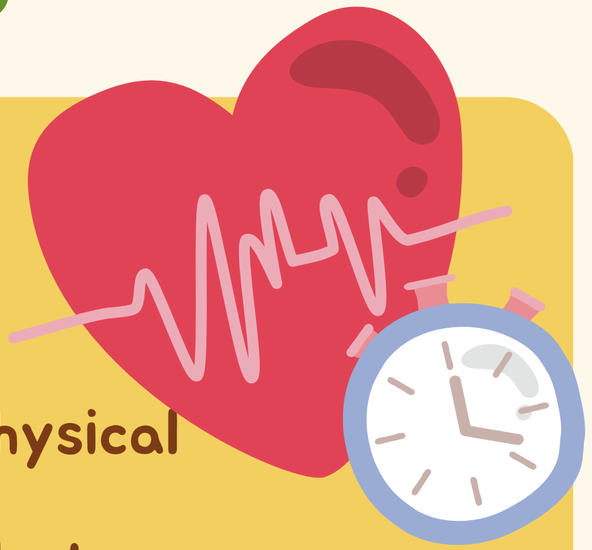
FOOD INTAKE

- Distinct meal categories:
  - breakfast, lunch, dinner, and snack.
- Log consumption by 2 methods:
  - searching for items in the system's database
  - manually entering nutritional information.

- Track calories, macronutrients (carbohydrates, fats, proteins), and fiber consumed throughout the day.

DAILY TRACKING

PHYSICAL ACTIVITY



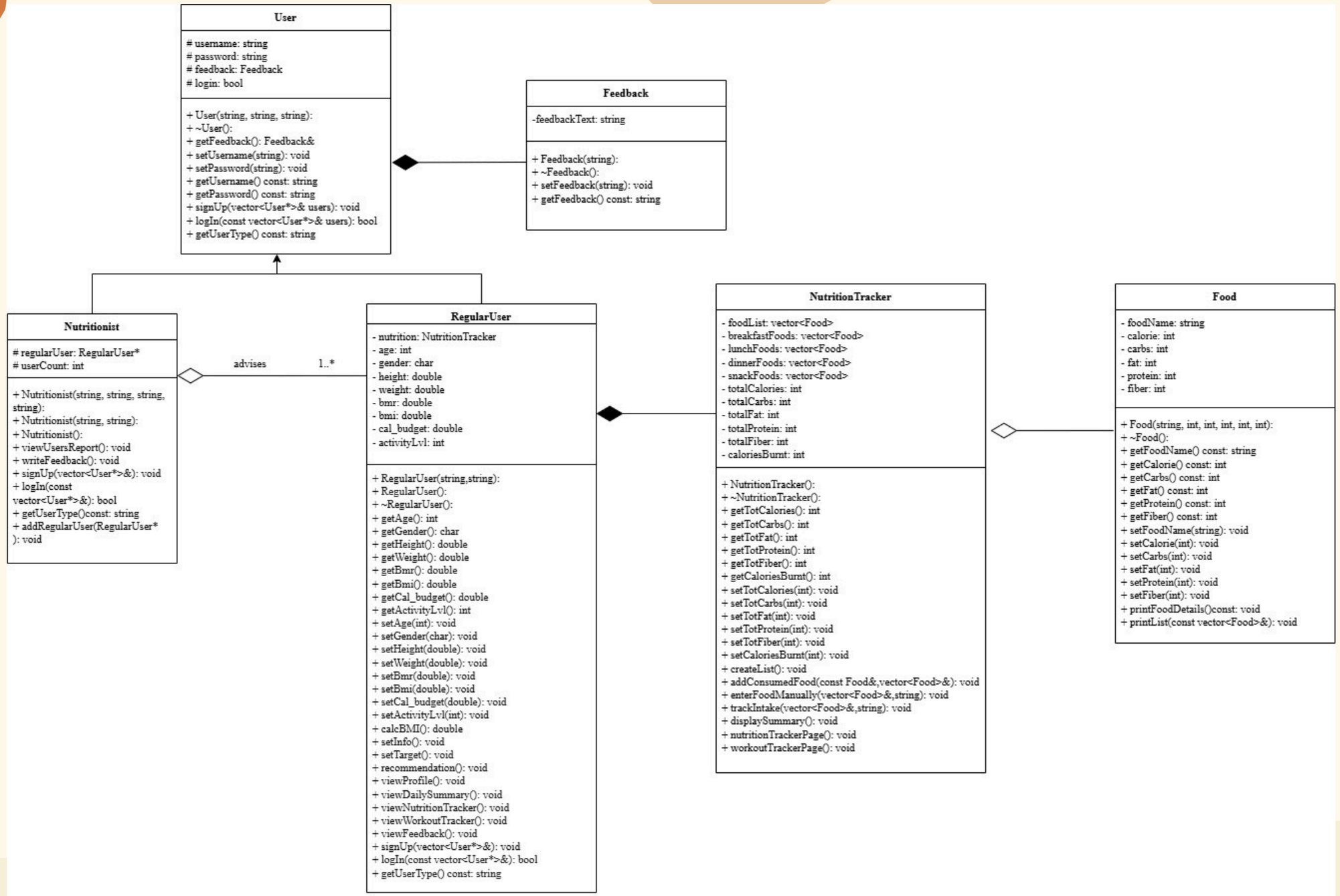
- Track daily calories burned that contributed by physical movement. This includes
    - exercises, workouts, or any other activity that contribute to energy expenditure.

- Users can directly input the calories burnt through physical activity.

FEEDBACK FROM NUTRITIONIST

- Real-time overview on users' progress towards their nutrition goals.
- Personalized feedback from nutritionists based on users' diet patterns.

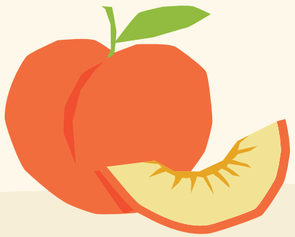
UML CLASS DIAGRAM



ARRAY OF OBJECTS

- ✓ Nutritionist has a array of object (regular user) to manage regular user

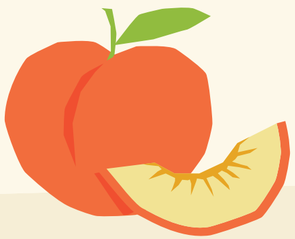
```
172   class Nutritionist:public User {
173       protected:
174           RegularUser* regularUser[MAX];
175           int userCount;
176       public:
177           Nutritionist(string, string, string, string);
178           Nutritionist(string, string);
179           Nutritionist();
```



ENCAPSULATION

Encapsulation refers to combining attributes and methods in one package, known as a class and hiding the implementation of the data from the user of the object. This ensures data security and prevents unauthorized access.

```
12  class Feedback {
13      private:
14          string feedbackText;
15      public:
16          Feedback(string);
17          ~Feedback();
18          void setFeedback(string);
19          string getFeedback() const;
20          void display();
21  };
22
23  class Food {
24      private:
25          string foodName;
26          int calorie;
27          int carbs;
28          int fat;
29          int protein;
30          int fiber;
31
32      public:
33          Food(string,int,int,int,int,int);
34          ~Food();
```



ASSOCIATION (COMPOSITION)

1. User and Feedback:
   - User class has an attribute feedback of type Feedback.
   - Justification: A nutritionist can provide a feedback and it can be viewed by regular user, and thus, there is a direct relationship where each user can be associated with feedback.

```
92  class User{
93      protected:
94          string username;
95          string password;
96          Feedback feedback;
97          bool login;
98      public:
99          User(string, string, string);
100         ~User();
```

```
586 void RegularUser::viewFeedback(){
587     system("cls");
588     cout << "\t\t=========================================================================\n";
589     cout << YELLOW << "\n\t\t\t\t\t***Feedback***\n\n" << RESET;
590     cout << "\t\t=========================================================================\n";
591
592     cout << "\t\t" << feedback.getFeedback();
593     cout << endl;
594     cout << ORANGE << "\n\t\tPress Enter to continue..."<<RESET;
595     cin.ignore();
596     cin.get();
597 }
```

ASSOCIATION (COMPOSITION)

2. RegularUser and NutritionTracker:
   - RegularUser class has an attribute calories of type NutritionTracker.
   - Justification: Regular users need to track their calories, and this tracking is managed through the NutritionTracker class, thus associating these two classes.

```
114    class RegularUser:public User {
115        protected:
116            NutritionTracker nutrition;
```

ASSOCIATION (AGGREGATION)

1. NutritionTracker and Food:
   - NutritionTracker class has an attribute food of type Food by using vector
   - Justification: The nutrition tracker needs to manage different food items, hence associating it with the Food class

```
53  class NutritionTracker {
54      private:
55          vector<Food> foodList;
56          vector<Food> breakfastFoods;
57          vector<Food> lunchFoods;
58          vector<Food> dinnerFoods;
59          vector<Food> snackFoods;
```

ASSOCIATION (AGGREGATION)

2. Nutritionist and RegularUser:
   - Nutritionist class has an attribute regularUser of type RegularUser *.
   - Justification: A nutritionist manages multiple regular users, providing recommendations and viewing their calorie catalogs, thereby creating an association relationship.

```
171   class Nutritionist:public User {
172       protected:
173           RegularUser* regularUser[MAX];
```

INHERITANCE AND POLYMORPHISM

1. RegularUser and User:
   - RegularUser inherits from User.
   - Justification: A regular user is a specific type of user with additional attributes and methods related to health and calorie tracking, thus inheriting basic user properties.
2. Nutritionist and User:
   - Nutritionist inherits from User.
   - Justification: A nutritionist is also a type of user with specialized methods for managing regular users, making inheritance from the User class appropriate

```
171   class Nutritionist:public User {
172       protected:
173           RegularUser* regularUser[MAX];
```

```
class RegularUser:public User {
    protected:
        NutritionTracker nutrition;
        int age;
        char gender;
```

INHERITANCE AND POLYMORPHISM

Polymorphism:

- Login and Sign Up method
- Pure abstract method inside User class and override in child class of regular user and nutritionist that has own specification

```
114  class RegularUser:public User {
115      protected:
116          NutritionTracker nutrition;
117          int age;
118          char gender;
119          double height;
120          double weight;
121          double bmr;
122          double bmi;
123          double cal_budget;
124          int activityLvl;
125
126      public:
127      void signUp(vector<User*>&);
128      bool logIn(const vector<User*>&);
129      string getUserType() const;
130
```

```
class Nutritionist:public User {
    protected:
        RegularUser* regularUser[MAX];
        int userCount;
    public:
        Nutritionist(string, string, string, string);
        Nutritionist(string, string);
        Nutritionist();

        void viewUsersReport();
        void writeFeedback();

        void signUp(vector<User*>&);
        bool logIn(const vector<User*>&);
        string getUserType()const;
```

```
virtual void signUp(vector<User*>& users) = 0;
virtual bool logIn(const vector<User*>& users) = 0;
virtual string getUserType() const = 0;
```

EXCEPTION HANDLING:

We apply on Regular User class that use to check age, height, weight, gender and activity level that entered by the user to avoid invalid data and accepted by the system in order to avoid abnormal conditions in the system.

```
353     do
354     {
355         try
356         {   cout << "\t\tAge: ";
357             cin >> i;
358             setAge(i);
359             break;
360         }
361         catch(RegularUser::InvalidInput)
362         {
363             cout << RED << "\t\tInvalid input" << RESET << endl;
364             cin.clear();
365             cin.ignore();
366             continue;
367         }
368         break;
369     } while (true);
```

class RegularUser

System Demonstration

2E1i

THANK YOU FOR WATCHING!

Prepared by 2E1i group member