CSC207 Phase 1

Group 233
Group member: Yumeng He, Zixuan Zhou, Xinxue Guo,Yancheng Zhou, Yin Hao, Ray Xu, Yicao Song

Table of Contents

1. Updated Specification (Underlines are updates for completion or new function)

Once the individual downloads the app on a device, he/she can open the application and see the main **menu** on the screen of the device which includes three sections: **food**, **exercise** and **account**. If the user is interested in this app, he/she can sign up his/her individual account with username and password. The system will create an unique database for the user to record his/her own progress.

For the Exercise section, the app lists out all the workout types for the user to choose from. By choosing the workout type, the user can decide to start the exercise or record the exercise that he/she has already done. By doing so, the system will record workout load and calculator the amount of calories burnt for the user

Manage Exercise:

- If the user chooses to start the exercise, then the user needs to press the start button to begin and press the end button when finish (Button was implemented in Controller Class). This helps the app system to record the user's exercise time. Moreover, the user can also enter the amount of workout load by numbers they have done if the exercise type cannot be counted in proper time units.
- If the user chooses to record the exercise he/she has already done, then the user has to enter the exercise time or the amount of workload they have done directly into the app. This data will be counted as part of your today's workout load, together with the finished exercise that has already been recorded by the app.

Manage Food:

- The users can check the desired food's calories. Once they enter the food's name or tap the food's picture, they will get the food's calories in standard units. For example, a 100 grams bagel contains 310 cals. Therefore, users can get a sense of the meal calories based on this information.
- If users choose to record the food that they had already eaten or expected to eat, then they simply choose the food they ate and enter the weight of the food. After the first one is done, they can input other food from their meal.

The users can be guided according to their needs, not only the workout but also the food. Once they make the request of their need, the app will offer several workout plan recommendations and meal plan recommendations for users to choose.

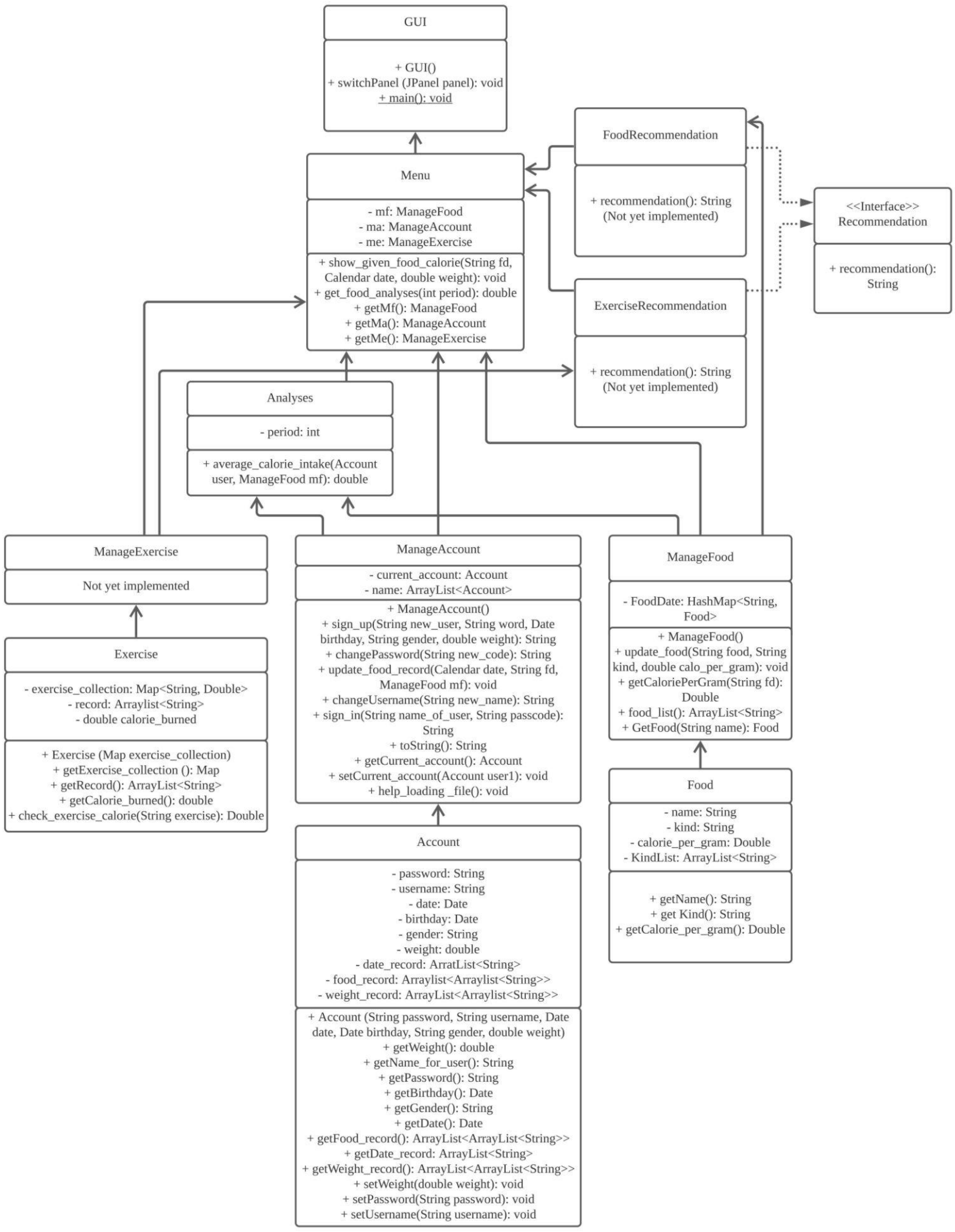
- For food recommendation, the user needs to enter the amount of calories that he/she wants to intake. By doing so, they will get several balanced meal plans.
- For workout plan recommendation, the user needs to enter the desired workout time and amount of calories that he/she wants to burn. By doing so, they will get a few recommended workout plans.

Moreover, this recommendation system can be automatically updated based on the user's preference, which can be discovered from the user's previous selections of workout plan recommendation and meal recommendation.

After finishing the exercise, the user can go to his/her account section on the main menu to check his/her daily exercise **analysis.** This daily analysis includes a table for the exercise time length, calories burnt and calories intake from his/her diet. To make the user feel convenient, the analysis can be viewed in different visualizations, such as pie charts or bar graphs to show the user's average workout length, amount of calories burnt, and amount of calories taken-in. In addition, the user can choose to view this analysis in the desired time unit, such as weekly and monthly. With this, the user can gain a better sense of his/her workout progress by checking the changes in calories with the best-fit visualizations.

Moreover, users can also manage personal information, like height, weight, or account information in the account section. At start, the user needs input his/her name and body index, such as height, weight, and age. Moreover, the email required for sign up and password can also be updated as well. All this information can be edited in the future if the user wants to change it. The user just simply goes to the account section in the main menu and chooses the edit button at the top.

2. Class Diagram


GUI
+ GUI()
+ switchPanel (JPanel panel): void
+ main(): void
Menu
- mf: ManageFood
- ma: ManageAccount
- me: ManageExercise
+ show_given_food_calorie(String fd, Calendar date, double weight): void
+ get_food_analyses(int period): double
+ getMf(): ManageFood
+ getMa(): ManageAccount
+ getMe(): ManageExercise
FoodRecommendation
+ recommendation(): String (Not yet implemented)
<<Interface>> Recommendation
+ recommendation(): String
ExerciseRecommendation
+ recommendation(): String (Not yet implemented)
Analyses
- period: int
+ average_calorie_intake(Account user, ManageFood mf): double
ManageExercise
Not yet implemented
Exercise
- exercise_collection: Map<String, Double>
- record: Arraylist<String>
- double calorie_burned
+ Exercise (Map exercise_collection)
+ getExercise_collection (): Map
+ getRecord(): ArrayList<String>
+ getCalorie_burned(): double
+ check_exercise_calorie(String exercise): Double
ManageAccount
- current_account: Account
- name: ArrayList<Account>
+ ManageAccount()
+ sign_up(String new_user, String word, Date birthday, String gender, double weight): String
+ changePassword(String new_code): String
+ update_food_record(Calendar date, String fd, ManageFood mf): void
+ changeUsername(String new_name): String
+ sign_in(String name_of_user, String passcode): String
+ toString(): String
+ getCurrent_account(): Account
+ setCurrent_account(Account user1): void
+ help_loading _file(): void
ManageFood
- FoodDate: HashMap<String, Food>
+ ManageFood()
+ update_food(String food, String kind, double calo_per_gram): void
+ getCaloriePerGram(String fd): Double
+ food_list(): ArrayList<String>
+ GetFood(String name): Food
Food
- name: String
- kind: String
- calorie_per_gram: Double
- KindList: ArrayList<String>
+ getName(): String
+ get Kind(): String
+ getCalorie_per_gram(): Double
Account
- password: String
- username: String
- date: Date
- birthday: Date
- gender: String
- weight: double
- date_record: ArratList<String>
- food_record: Arraylist<Arraylist<String>>
- weight_record: ArrayList<Arraylist<String>>
+ Account (String password, String username, Date date, Date birthday, String gender, double weight)
+ getWeight(): double
+ getName_for_user(): String
+ getPassword(): String
+ getBirthday(): Date
+ getGender(): String
+ getDate(): Date
+ getFood_record(): ArrayList<ArrayList<String>>
+ getDate_record: ArrayList<String>
+ getWeight_record(): ArrayList<ArrayList<String>>
+ setWeight(double weight): void
+ setPassword(String password): void
+ setUsername(String username): void

Citation: Using Lucidchart app to create the class diagram.
URL: www.lucidchart.com

3. Major Design Decisions

【1】Throughout the process, our group has faced a few major problems on this idea design. At first, our topic was not the fitness app. We were choosing between the movie rating website, music rating website, and the fitness app. Our group had a deep discussion on these topics and we asked the TA for his opinion. TA's advice was very useful and clever. He told us that the movie rating website and music rating website are basically the same program but function in different areas. Moreover, for things like these websites, we need to input all the information about the movie or the music. However, it may take quite a long time to accomplish this. On the other hand, the fitness app is easy to start and develop other functions. After absorbing TA's feedback, we had further discussions on this and finally we all agreed on the fitness app idea. We had a blueprint for this project. This fitness app can function as a workout progress report that records down the exercise time for the user. In order to gain the background information and understand how we can design this app to the best degree, we asked our friends who workout everyday for their opinions. They want something that is convenient for their workout and not just an exercise record. To extend this idea, we decided to bring our app to a new level by designing some new functions that are unique compared to others.

【2】We researched on the internet and took surveys for other useful functions that can be added in our app. After doing all this effort, we were basically finished with the base of our fitness app. We decided to improve our app by giving it more unique functions. First, our app can act like a workout record by helping the user to record down his/her exercise data. This is the function basically every workout app has. However, extending from this, we are adding the food calories checker in our app. Food calories checker works by searching up the food's name and the system will return the amount of calories in the standard unit. This benefits the user by providing them a convenient meal calories researcher. With these two functions, the users would gain a sense of their daily calories burnt and calories taken-in.

【3】Since there are unlimited workout types and foods, it is hard to record all of them by just a seven-member team, then we decided that users can record their own personal workout type and food thus we added more functions in Manage Classes.

【4】As we mentioned above, it is also a massive workload if we create one parent class Exercise Class and Food Class then for each type of exercise and food we create subclasses. Not even the workload is massive, the files will be plenty and unable to be well-organized. After discussions, we tried to use the Template Method which perfected worked for our situation but our subclasses are still overwhelmed. We realized that for each workout type and food, it only contains two to three variables which are their name, calories, and MET if it is a workout type. Then we created one txt file for all of them.

4. Clean Architecture

The different classes play different roles. For example, we would like to make our system an app. The (GUI) External Interface can help us to achieve. The users can use their phones to turn on the app. In the meantime, the whole system will through the external interface to the controllers. The Controller Class Menu will offer several choices to the users which include the Account, Exercise, Food. The users can press the Exercise button. And the new pages will appear. The users can choose time during their exercise. The use case class ManageExercise will help the users to achieve this function. Once they start the exercise, they can press the start, and after finishing the exercise, they press the stop. At the same time, the use case class can calculate the total exercise time.And exercise time data will be saved into the user's account through the Entity class Account.Therefore, we can notice that the whole system will always access the class from outer to inner, and it follows the dependency rule. In addition, not only the structure but also the code part we also follow the Clean Architecture. This means that the name of the functions or classes declared in the outer layer does not be mentioned by the code in an inner layer.

5. SOLID Design Principles

In our code, each class only has a single responsibility. And the code is easy to extend. For example, In the exercise code, there are many types of exercise, and some have similar functions. But different burn calories. It is easy to use to extend the new exercise based on the original one.
Moreover, we follow the interface segregation principle. To avoid the high coupling, we have several entity classes (Exercise & Food & Account). Once we want to make some changes to the data, we need to use an interface. The interface offers a simple but important function that we will use. And the interface connects to the use case class. Thus it can help us to avoid coupling.

6. Packaging Strategies

We used a component packing strategy since we are placing all the databases together, other than this, we do not really use other special packaging strategies.

7. Summary of design patterns

Command is one of the behavioural design patterns which can transform a request from users into a standalone object with all the request's details. Since our group created a user interface component such as a context menu in several places as a menu bar on top of each page and display of list of exercises and foods. Our group planned for users to be able to interact with menu items that do actions when the user clicks them. Then it would be more convenient if we can include Command design patterns instead of repeatedly creating subclasses for each sport and food. In order to complete the task, we created one abstract class which is defined for executing the command, then there are two subclasses which are ManageExercise and ManageFood which are used to indicate parameters necessary to run a method on a receiving object.

The Template Method pattern involves breaking down an algorithm into a sequence of stages, turning these steps into methods, and calling these methods from inside a single "template method." We have a super class Exercise and decided to create subclasses for each kind of sport. Using a template method will reduce our workload and reduce the amount of subclasses.

8. Progress Report

Since we are trying to do an application where users want to search the calories they burn or the calories they eat every day, we have prepared a detailed database for account, food and exercise. And since we want to create an application where users can use it on their phone, we are making a great process on GUI, which means users will be able to download this application in the app store eventually. We have created an interface that can provide login or sign up for users, therefore their data will be saved account wise, even though they change their devices, they will still see their old history as long as they login to their accounts. and we are close to finishing the list or interface to analyse. And for further updates, we would love to make recommendations and analyse better. For recommendation, we want to provide recommendations to users according to what they eat in the past 7 days. For analysis, we currently only have average, but for further updates, we are planning to create a bar chart or pie chart for users in order to show them the percentage of a certain food users have eaten or the frequency of a certain food users have eaten. Seems like everything is doing well, but at the same time, we meet some hard core problems as well. Since we created 3 databases ourselves, it is hard to read a certain line of the databases if we want to use it for the coding part, and this is the main problem we are dealing with and trying to solve. At the same time, since our application requires users to login and signup, whenever there is a new user that has never used our application before, he needs to creates a new account for the app, so at the back end of the app, we need to create a brand new txt document to store all the information of this new user such as the user name, the password, the gender, the weight, and also what has this user done or what has this user eaten on a certain date, we would all memory these information in one txt document. It is hard for us to put data or retrieve data into this txt document through Java right now.

Compared with Phase 0, we have added up a lot of functions and we even created the database for the program.

Group Attribution Sheet

	Yumeng He	Zixuan Zhou	Xinxue Guo	Yicao Song	Yancheng Zhou	Ray Xu	Hao Yin
CODING							
Account							✓
ManageAccount							✓
Menu		✓					
GUI				✓			
Exercise							✓
Food		✓					
Analyses		✓					
ManageExercise							✓
ManageFood		✓					
Recommendation					✓	✓	
DOCUMENTATION							
Updated Specification	✓					✓	
Class Diagram		✓					
Major Design Decisions	✓					✓	
Clean Architecture			✓				
SOLID Design Principle			✓				
Packaging Strategy					✓		
Design Patterns	✓						
Progress Report					✓		
OTHERS							
Attribution Sheet	✓		✓	✓		✓	
PowerPoint	✓		✓		✓	✓	