Process MeNtOR 3.0

Uni-SEP

FitHealth
Design Document

object
oriented pty. ltd.

Document Change Control

Version	Date	Authors	Summary of Changes
1.0	oct	Bjorn, Khalifa, Joshua, Marko	Filled out Template
1.1	Nov 23rd 2023	Bjorn, Khalifa, Joshua, Marko	updated uml diagrams and testcases
2.0	Jan 11th 2023	Joshua Little	Reviewed and polished document.

Document Sign-Off

Name (Position)	Signature	Date
Joshua Little	Jlittle	22/10/23
Marko Stojsic	MStojsic	22/10/23
Bjorn Cipi	BC	22/10/23
Khalifa	KK	22/10/23

Contents

1 Introduction

1.1 Purpose

The purpose of FitHealth is to develop a software application to learn and apply software design concept. The application aims to help the user monitor and improve their health by tracking their calorie intake and expenditure, as well as providing feedback and recommendations based on their goals and habits. The application will also allow the user to customize their profile and settings, view their progress and history through graphs and charts, and compare their diet with the Canadian Food Guide.

The app must:

1. *allow users to put in age, weight height data, allow some customization.*
2. *allow users to input exercise Data and Meal Data*
3. *this data is used to calculate calorie burn, caloric intake.*
4. *These logs will enable us to create graph visualisations of exercise and diet habits for the client.*
5. *The data will be used to predict weight loss overtime with the user's current diet habits*
6. *the data user will be able to compare theyre diet with the Canadian Food guide, which will allow the user to make better diet choices.*

1.2 Overview

1.3 Link to github:[jelittle/3311FinalProject: Final project for eecs 3311 F23 at YorkU (github.com)]

Section 2 contains sequence diagrams of the use cases, as well as a short description of the diagrams.
Section 3 is an explanation of the major design decisions made during the development of FitHealth.
Section 4 is the architecture of FitHealth. It includes charts detailing the interaction between the modules and interfaces.

Section 5 contains UML diagrams for FitHealth. They were updated after the October 18 lecture to contain improved modularity, a change which the code does not yet reflect.

Section 6 explains the design patterns used in the project. It also contains a gantt chart of the timeline followed for Deliverable 1.

Section 7 contains the activities and implementations planned for all three deliverables.

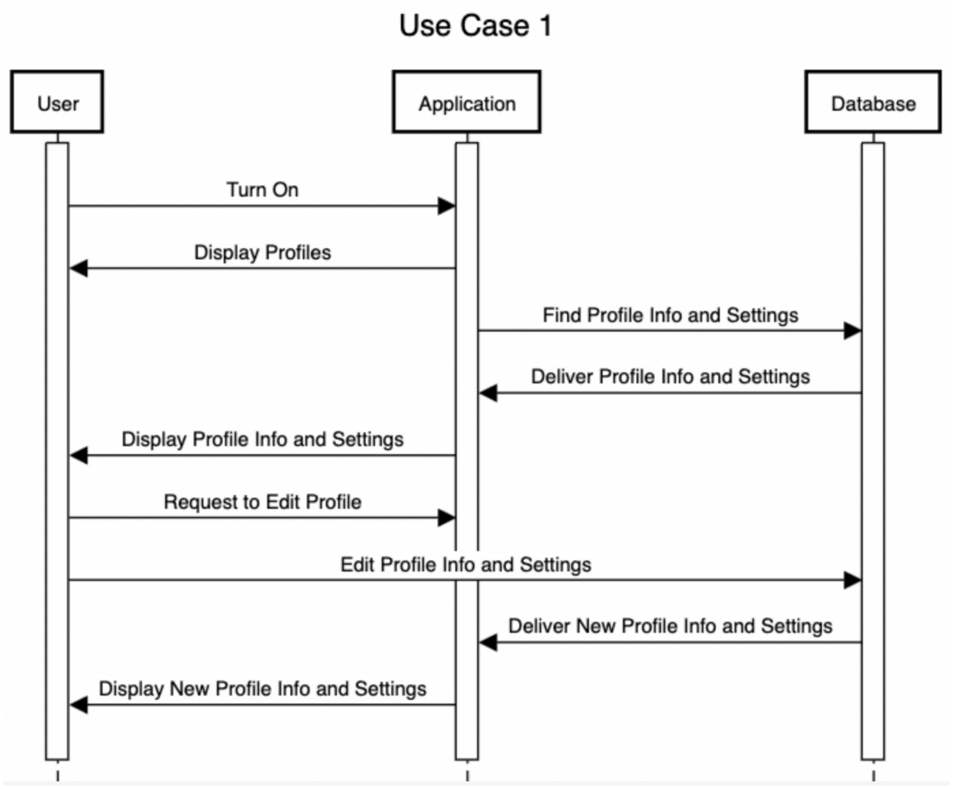
Section8 has test cases in accordance with the concept of test driven development.

1.4 Resources - References

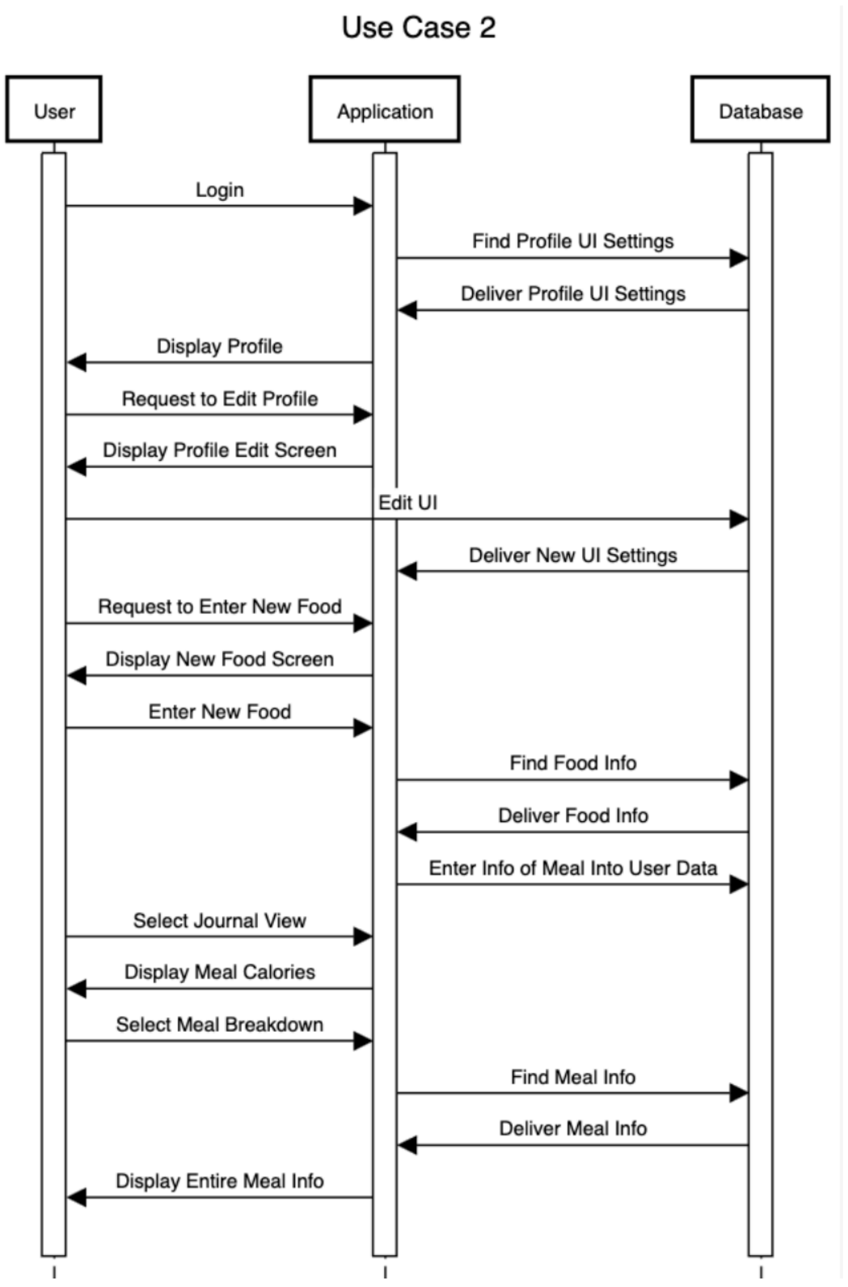
How Do You Calculate Calories Burned During Exercise? (medicinenet.com)

Compendium of Physical Activities - 12 - Running (google.com)

2 Sequence Diagrams


Use Case 1
User
Application
Database
Turn On
Display Profiles
Find Profile Info and Settings
Deliver Profile Info and Settings
Display Profile Info and Settings
Request to Edit Profile
Edit Profile Info and Settings
Deliver New Profile Info and Settings
Display New Profile Info and Settings

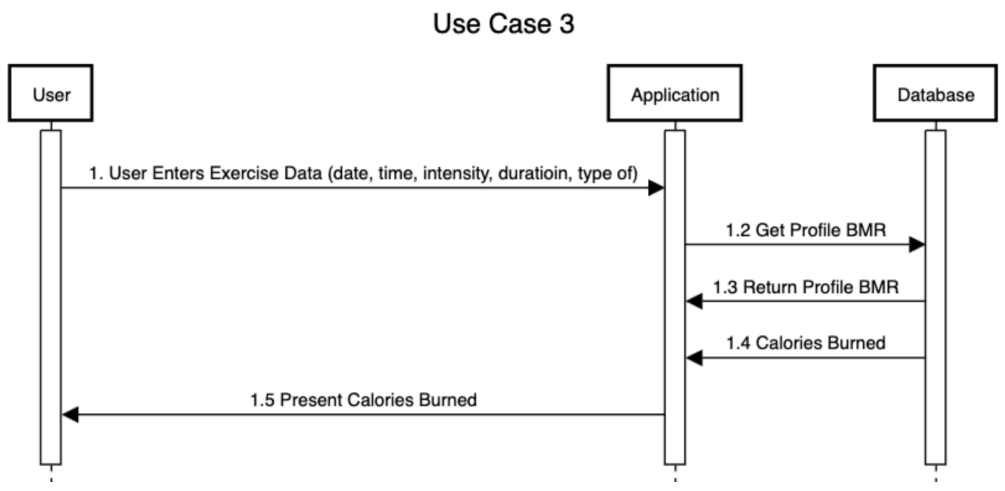
The user first turns on the application after which the profile screen is presented to the the user by the application. The application then fetches the profile info and settings from the database which are displayed to the user after the application receives the profile info and settings from the database. At that point, the user selects the "Edit Profile" button which enables them to directly edit profile info and settings within the database. The database delivers the new profile info and settings to the application which is then displayed to the user.



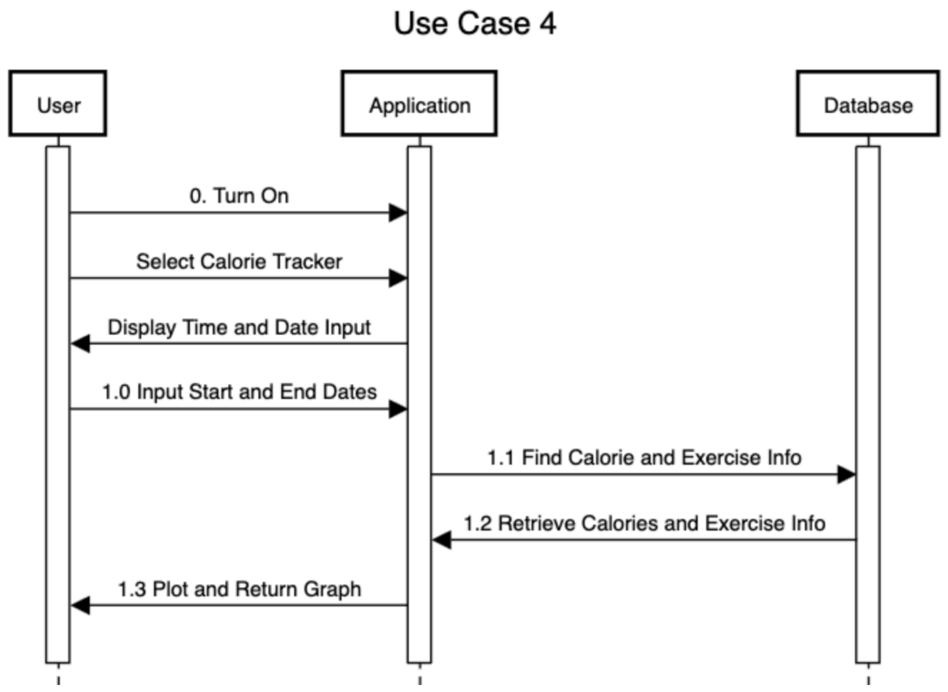
The user logs into the application which prompts the application to search the database for the users profile UI settings. The profile UI settings are then delivered to the application from the database and are then displayed to the user. At this point the user selects the option to edit their profile. This prompts the application to display the profile edit screen. The user is then enabled to edit the UI directly through the database which stores the UI settings. The database delivers the new UI settings to the application.

The user can now request to enter new food to the application. The app displays the new food screen which allows the user to enter new food to the app. Now, the app finds the entered foods info from the database which returns it to the application. The app then enters info of the users meal into the user data which is stored in the database.

The user may at this point select the "journal view" to the app which then displays the meal *calories to the user. The user selects the meal breakdown which prompts the application to find the meal info in the database. It is then retrieved from the database and displayed to the user.*

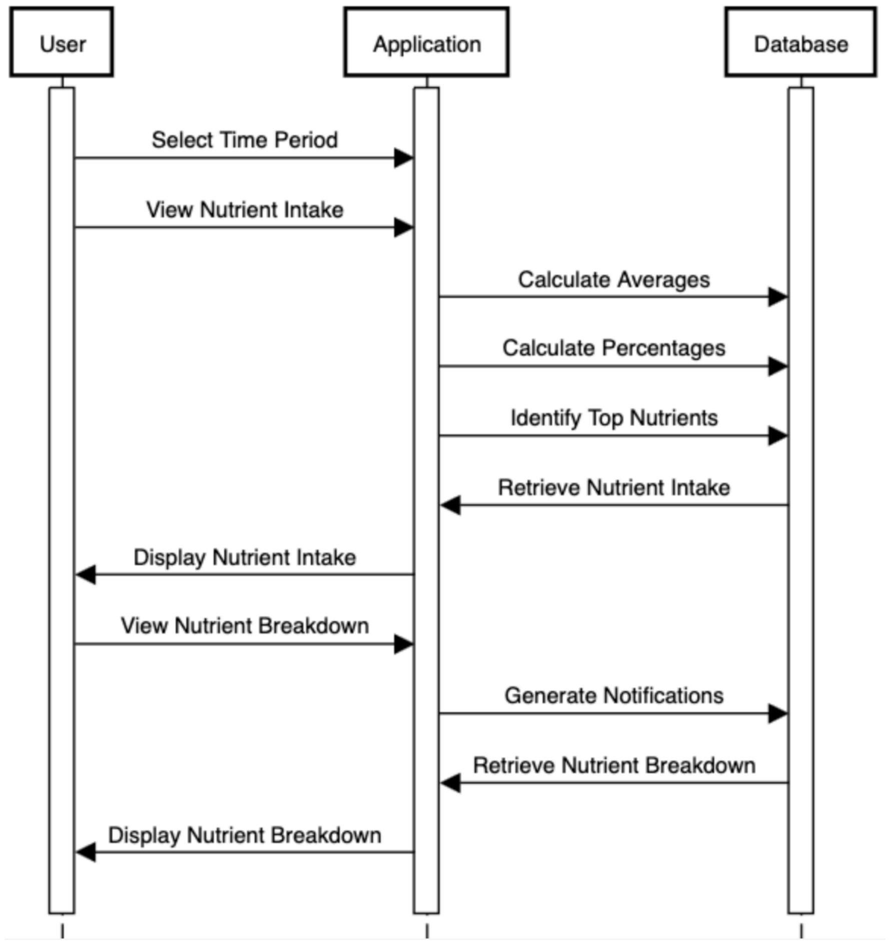

Use Case 3
User
Application
Database
1. User Enters Exercise Data (date, time, intensity, duratioin, type of)
1.2 Get Profile BMR
1.3 Return Profile BMR
1.4 Calories Burned
1.5 Present Calories Burned

First, the user enters exercise data into the app. The exercise data includes info such as the date of the exercise, time, intensity, duration, and type of. The application at this point requests the profile BMR from the database. The database successfully returns the profile BMR to the app as well as the calories burned for the exercise. At this point, the calories burned are displayed to the user from the application.

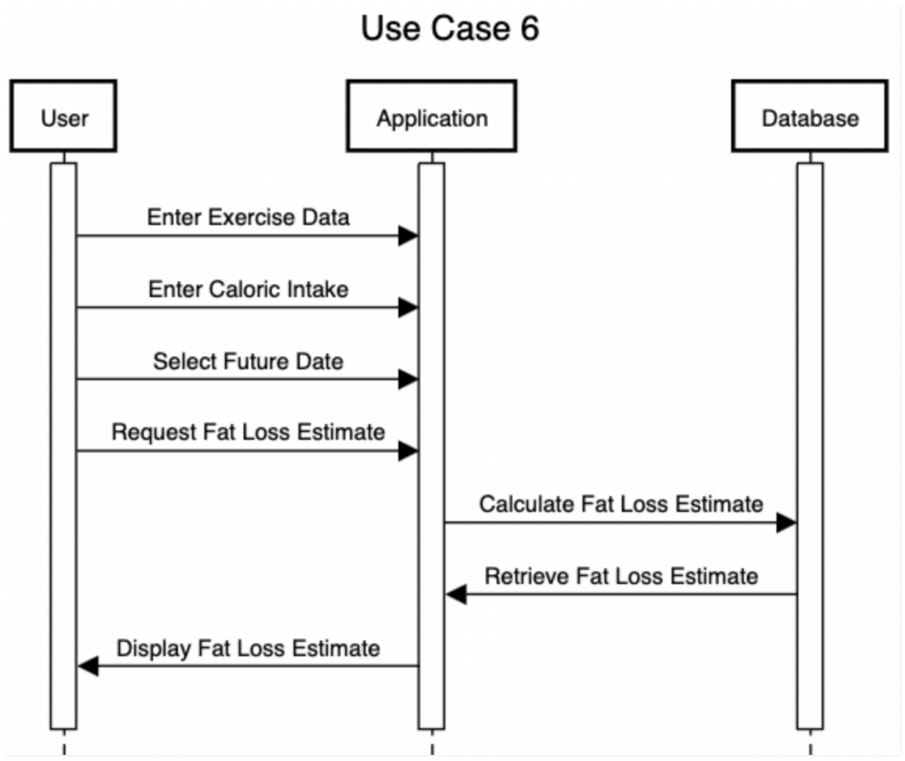

Use Case 4
User
Application
Database
0. Turn On
Select Calorie Tracker
Display Time and Date Input
1.0 Input Start and End Dates
1.1 Find Calorie and Exercise Info
1.2 Retrieve Calories and Exercise Info
1.3 Plot and Return Graph

First, the user turns on the application after which they are able to select the calorie tracker which is requested from the app. The app displays the time and date input fields which prompt the user to input the start and end dates for their calorie tracker. The app now searches the database for the calorie and exercise info which is then retrieved from the database. At this point, the app plots and displays the calorie tracked in a graph to the user

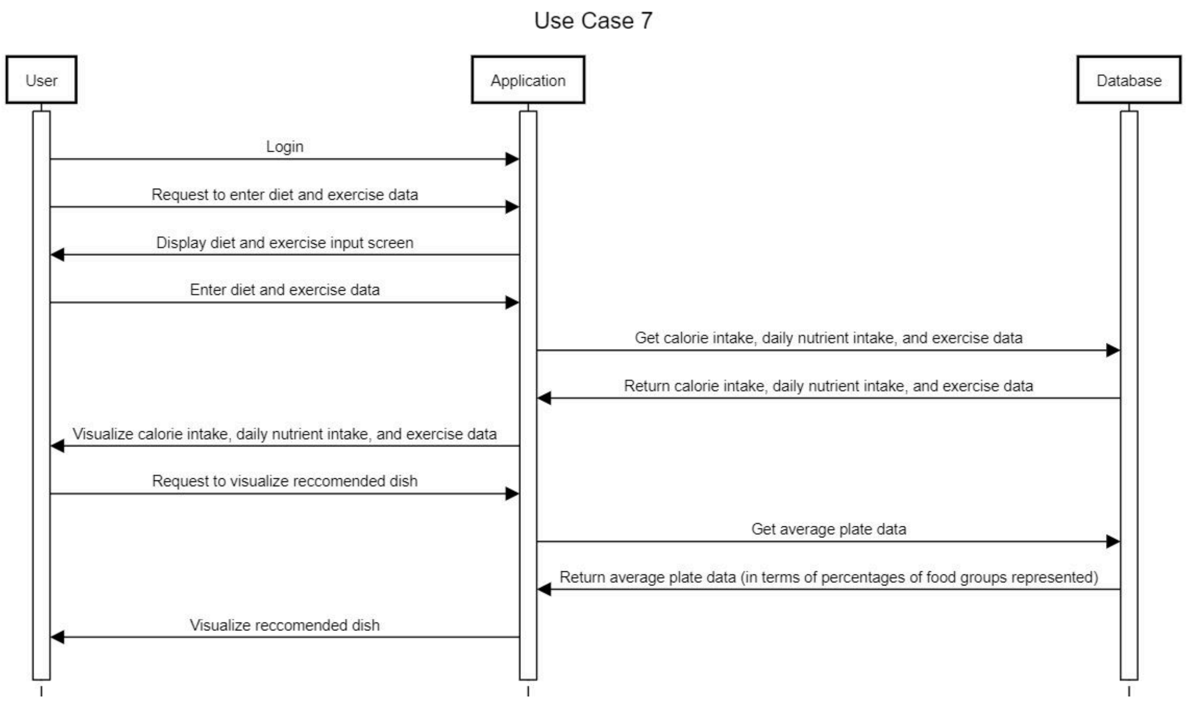
Use Case 5



The user first selects a time period and the option to view the nutrient intake. The app requests the averages, percentages, and identifies the top nutrients from the database. The database sends the nutrient information to the app which then displays it to the user. At which point, the user requests to view the nutrient breakdown from the app. The app generates the notification and the database retrieves the nutrient breakdown and sends it to the app. The app then displays the nutrient breakdown to the user.


Use Case 6
User
Application
Database
Enter Exercise Data
Enter Caloric Intake
Select Future Date
Request Fat Loss Estimate
Calculate Fat Loss Estimate
Retrieve Fat Loss Estimate
Display Fat Loss Estimate

The user first enters the exercise data, caloric intake, and a future date in the app. They request the fat loss estimate from the app which then requests it from the database. The fat loss estimate is then returned from the database to the app which is then displayed to the user


Use Case 7
User
Application
Database
Login
Request to enter diet and exercise data
Display diet and exercise input screen
Enter diet and exercise data
Get calorie intake, daily nutrient intake, and exercise data
Return calorie intake, daily nutrient intake, and exercise data
Visualize calorie intake, daily nutrient intake, and exercise data
Request to visualize reccomended dish
Get average plate data
Return average plate data (in terms of percentages of food groups represented)
Visualize reccomended dish

The user first logs into the application after which they request to enter diet and exercise data. The app displays the diet and exercise input screen which prompts the user to enter diet and exercise data to the app. The app then requests the calorie intake, daily nutrient intake, and exercise data from the database. The calorie intake, daily nutrient intake, and exercise data is retrieved from the database after which the app visualizes the calorie intake, daily nutrient *intake, and exercise data via jfreechart to the user. The user can now request to visualize the recommended dish from the app. The app requests the average plate data from the database. The database returns the average plate data (in terms of percentages of food groups represented) to the application. Finally, the visualized (via jfreechart) recommended dish is displayed to the user.*

3 Major Design Decisions

"Initially, our project was grounded in an MVC silo-based architecture. However, early in the development process, we strategically pivoted to a more modular, layered architecture. This transition facilitated a broader exploration of design concepts among our team members and enabled a more equitable distribution of workload.

4 Architecture

Modules			
Module Name	**Description**	**Exposed Interface Names**	**Interface Description**
Database	The Database Module manages data flow between the application and the database.	Database:Idatabase	Database:Idatabase calls a factory that sends either the hardcoded or live version of our database
Navigation	module connects and manages front end modules, serves as a link between them	Navigation:INavigationInterface	Navigation:INavigationInterface allows navigation between the 3 front end modules
ExerciseLogic	handles the logic for the exercise Based windows	ExerciseLogic:IExerciseLogic	ExerciseLogic:IExerciseLogic Exposes a command design pattern that allows the front end to get needed data for fat loss, graph management, ExerciseLogs within a range etc,
ExerciseFrontEnd	Builds and holds all base windows pertaining to exercise, handles user input andoutput	ExerciseFrontEnd:IExerciseWindow	ExerciseFrontEnd:IExerciseWindow interface allows navigation pane to build window data
ExerciseLog	models data and gets data from database class, builds data	ExerciseLog:IExerciseLog	ExerciseLog:IExerciseLog Exposes exerciseLog to exerciseLogic so Exerciselogic get data.,
UserFrontEnd	switches between the start, login, home, and setting screen	UserFrontEnd:IUserFrontEnd	Allows navigation between the 4 front end windows
UserLogic	handles the logic for user sign in and changing settings	UserLogic: IUserLogic	UserLogic: IUserLogic uses an adapter design pattern to translate the front end methods into the UserClient methods
User	contains the User type, the variables relating to it, and tools for searching, editing, entering, and removing a user from the database	User:IUser User:IUserClient	User:IUser exposes User ti userLogic to retrieve data User:IUserClient handles communication with the database
DietLogic	handles the logic for the Diet Based windows	User:IUser User:IUserClient	DietLogic:IDietLogic Exposes a command design pattern that allows the front end to get needed

DietFrontEnd	Builds and holds all base windows pertaining to Diet	DietFrontEnd:IDietFrontEnd	DietLogic:IDietLogic Exposes a command design pattern that allows the front end to get needed

DietLog	provides a structured representation of diet log entries and encapsulates the logic for calculating nutrient values.	Diet:IDiet Diet:IDietClient	DietLogic:IDietLogic Exposes DietLog to DietLogic so

Interfaces		
Interface Name	**Operations**	**Operation Descriptions**
Database:IDatabase	connects to exerciselog, meallog and user modules -- #getRecord(Table:string,return:string,Conditions:string): #getRecords(Table:string, Table:string,return:string,Conditions:string): #insertRecord(Table:string, Table:string,return:string,Conditions:string): #<boolean>updateRecord() #<boolean>getRecordsql()	gets records from sql simplifies building sql queries to simple table calls with conditions and wanted attributes rather than full sql
Navigation:INavigationInferface	<Boolean> changePage()	navigates to a different Page
ExerciseLogLogic:IExerciseLogic	+Execute(Component:String,Data:Array<?>):ArrayList<String> used by ExerciseFrontEnd	Build and route Commands sent by front end

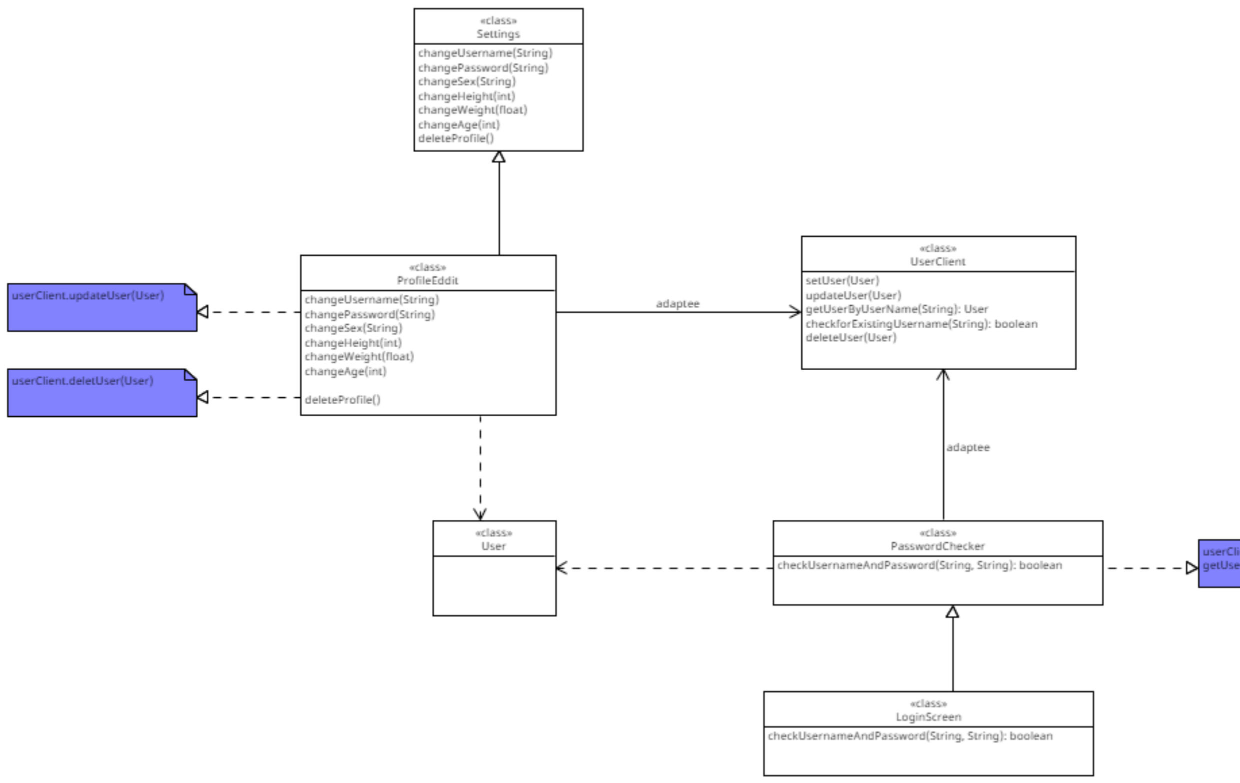
ExerciseWindow:IExerciseWindow	<boolean> buildWindow <boolean> delta window used by navigation	exposes Exercise front end to the navigation class to allow navigation of different portions of the project.
ExerciseLog:IExerciseLog	<ArrayList<Client>> getClient() <ArrayList<ExerciseLog>> getClient() <ArrayList<Met>> getClient() used by ExerciseLogLogic	gets needed Data from ExerciseLog for Exercise Logic
UserFrontEnd:IUserFrontEnd	<Boolean> changePage()	Allows the navigation between windows
UserLogic:IUserLogic	+Execute(Component:String,Data:Array<?>):ArrayList<String> used by ExerciseFrontEnd	Build and route commands sent by front end
User:IUser	calculateBMR() setName(String) getName(): String setPassword(String) getPassword(): String setSex(String) getSex(): String setHeight(int) getHeight(): int setWeight(float) getWeight(): float setAge(int) getAge(): int setID(int) getID(): int getBMR(): int	User builder

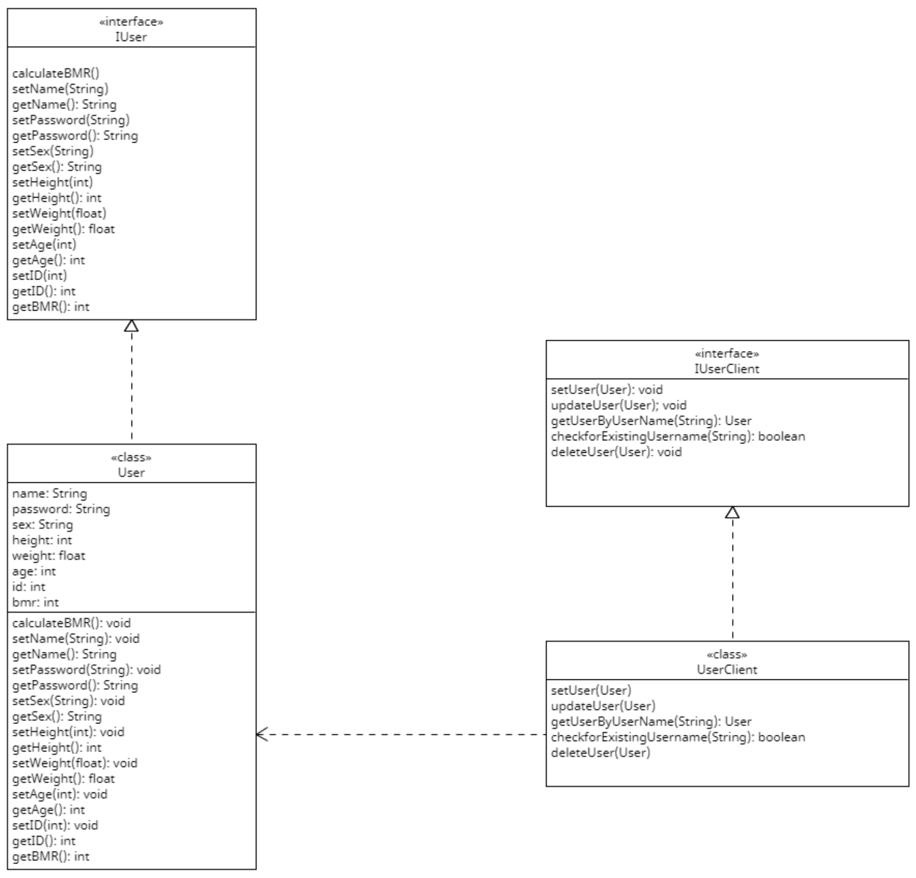
User:IUserClient	setUser(User)
	
	updateUser(User) getUserByUserName(String): User checkforExistingUsername(String): boolean deleteUser(User)
		Get needed data from the database

DietLog: IDietLog	getDietLogById(DietLogEntry dietLog): int setDietLog(DietLogEntry dietLog): boolean updateDietLog(DietLogEntry dietLog)(DietLogEntry dietLog): void deleteDietLog(DietLogEntry dietLog): void	gets needed Data from Diet for Diet Logic
DietWindow: IDietWindow	<boolean> buildWindow	exposes Diet front end to the navigation class to allow navigation of different portions of the project.
IDietLogic	+Execute(Component:String,Data:Array<?>):ArrayList<String>	Build and route commands sent by front end

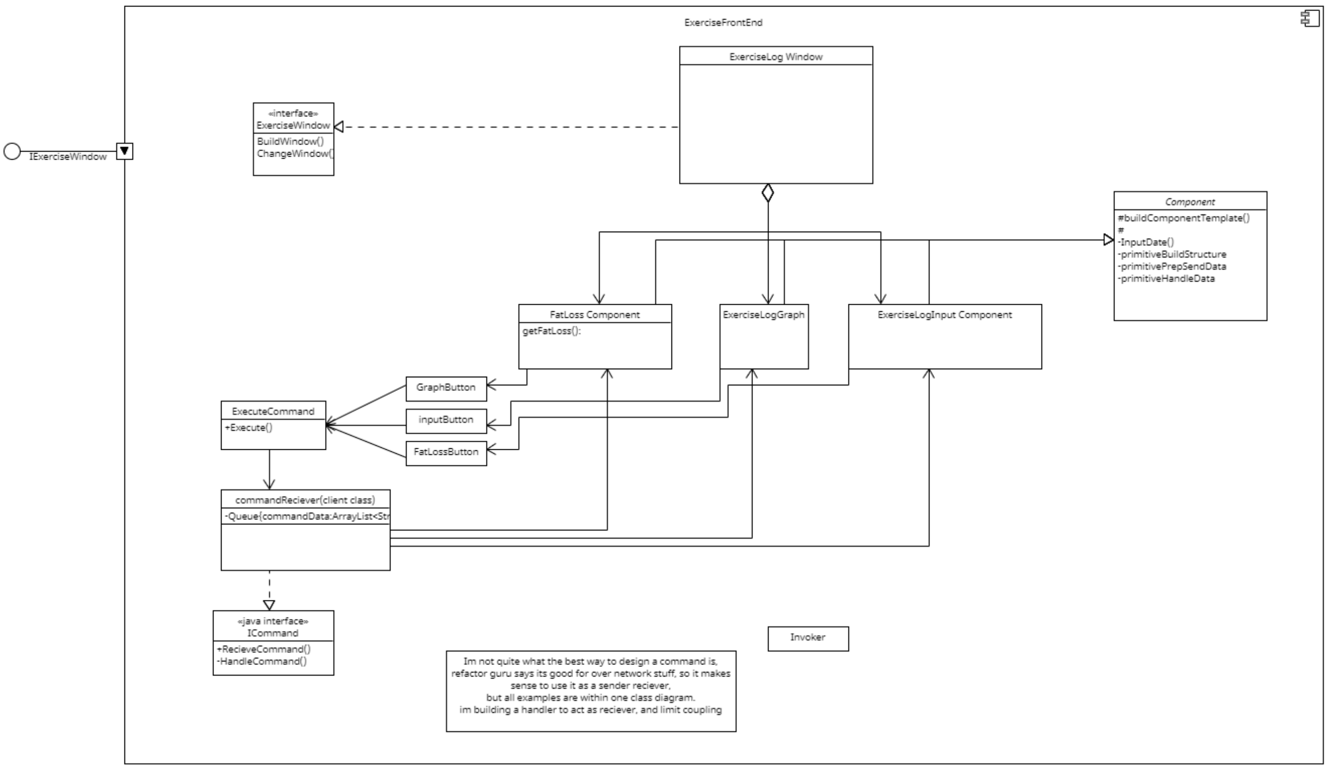
5 Detailed Class Diagrams

5.1 UML Class Diagrams.

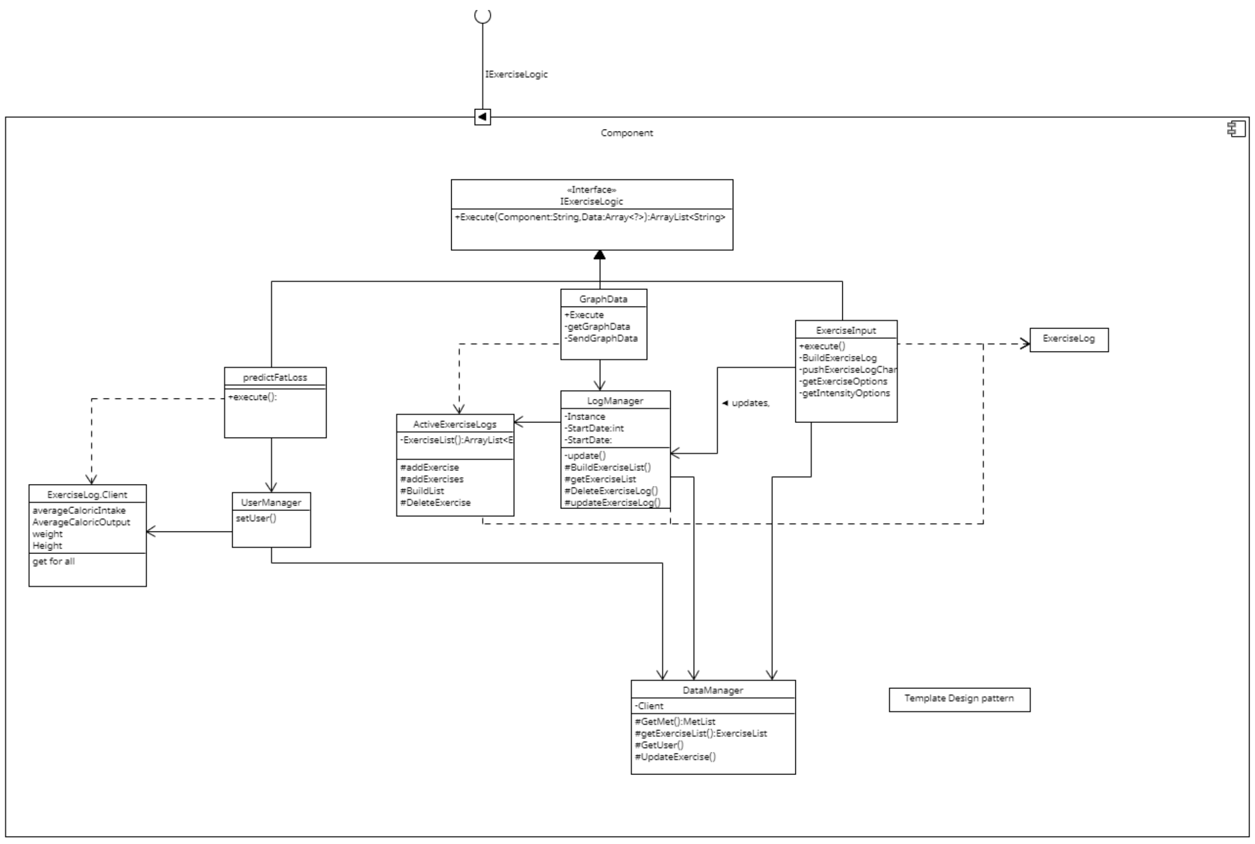




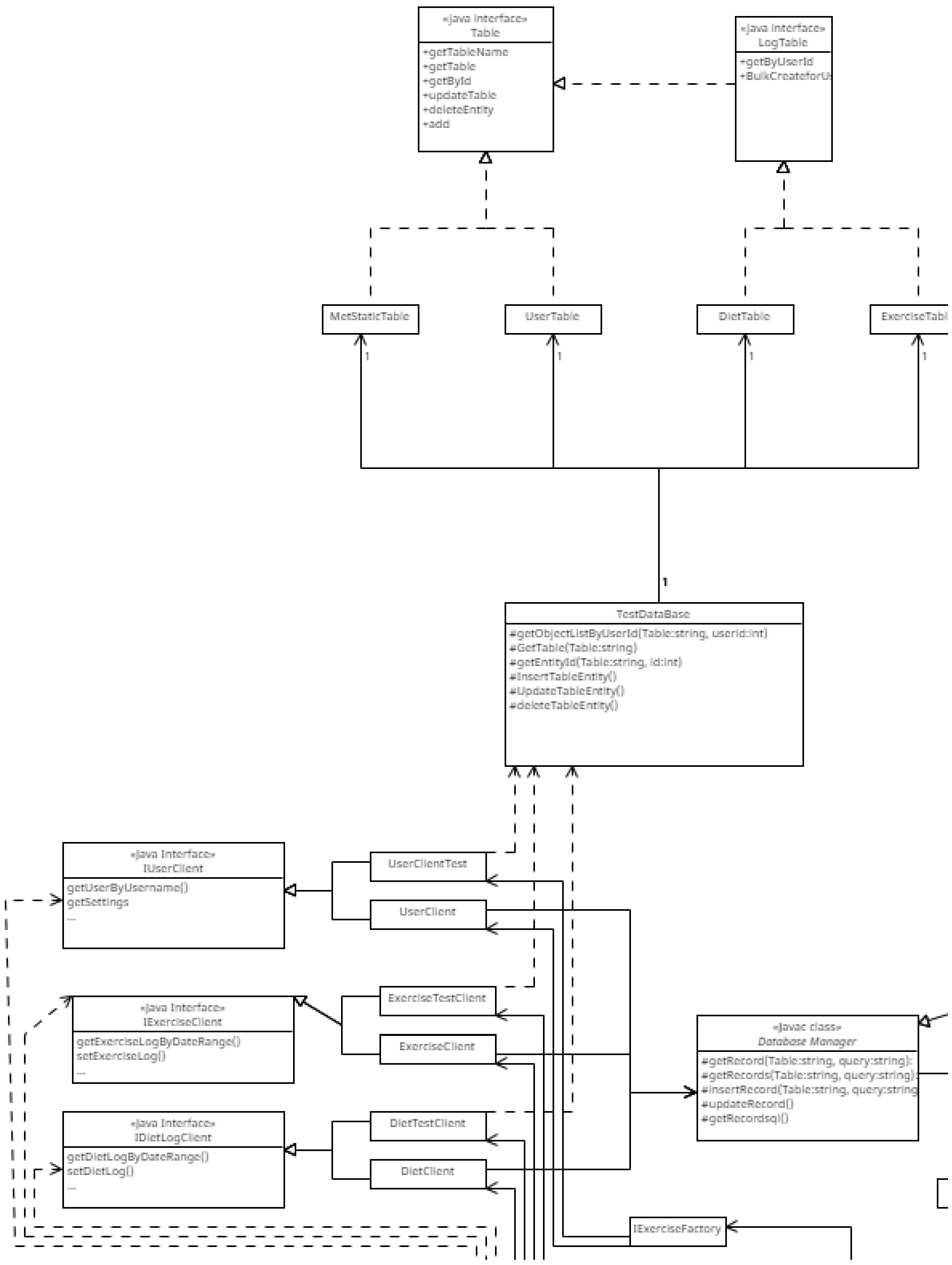
Exercise Branch modules:

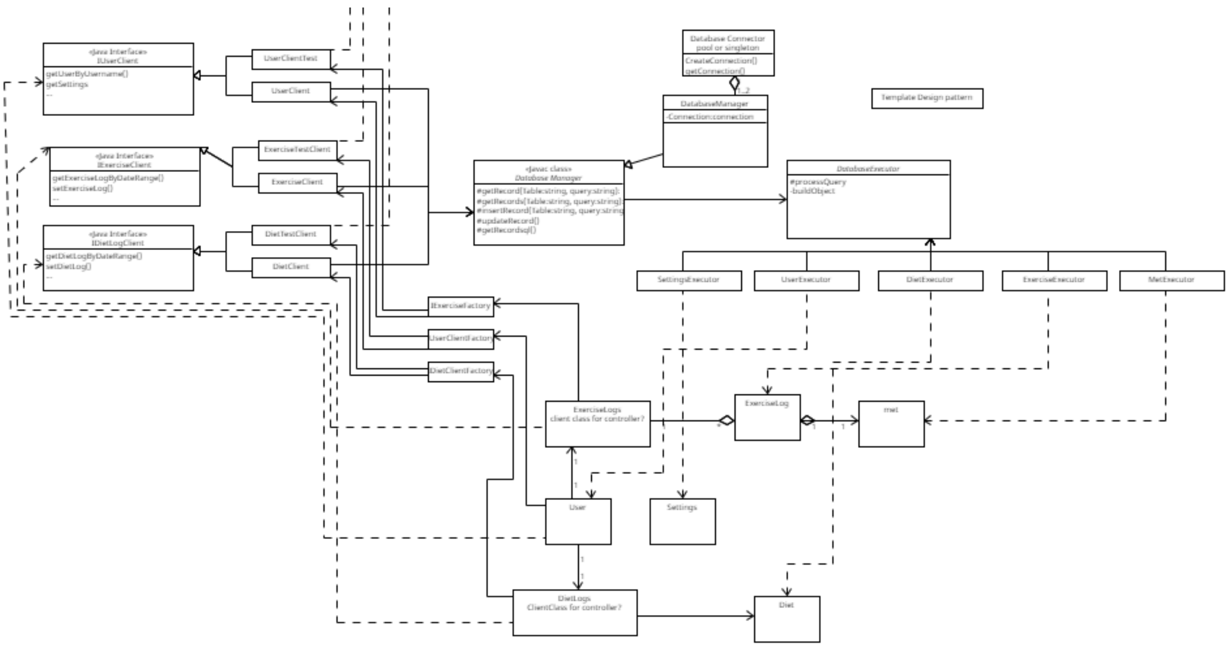


Im not quite what the best way to design a command is,
refactor guru says its good for over network stuff, so it makes
sense to use it as a sender reciever,
but all examples are within one class diagram.
im building a handler to act as reciever, and limit coupling



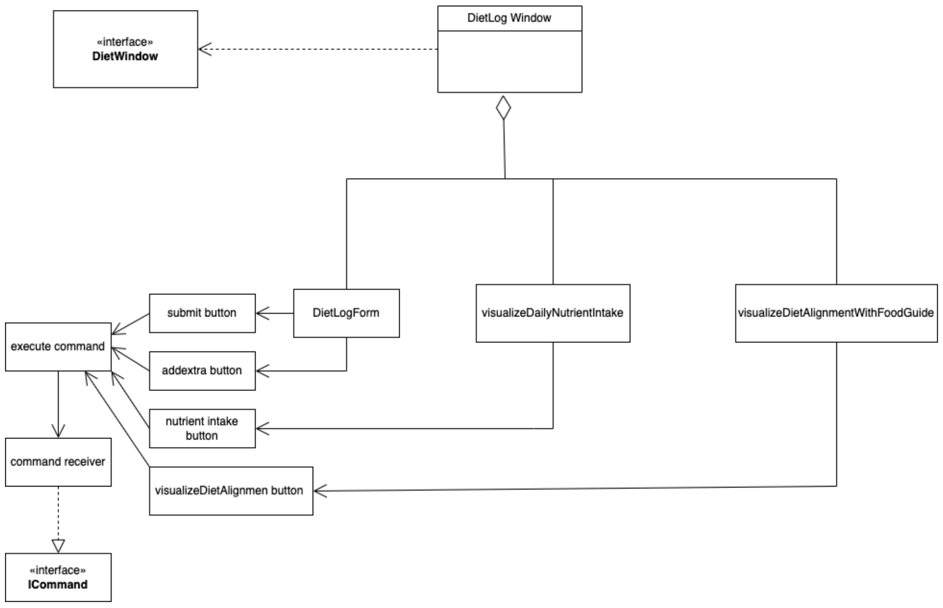
DatabaseModule:

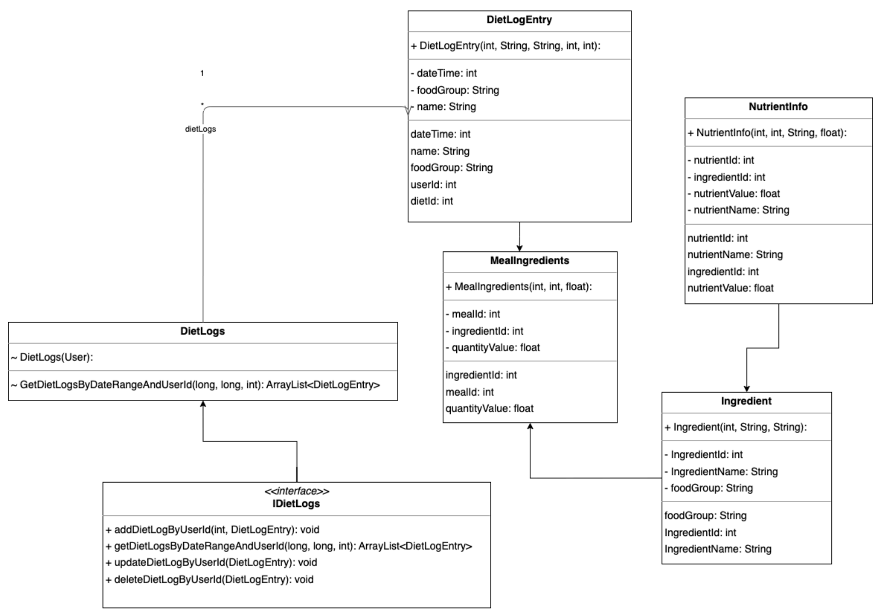


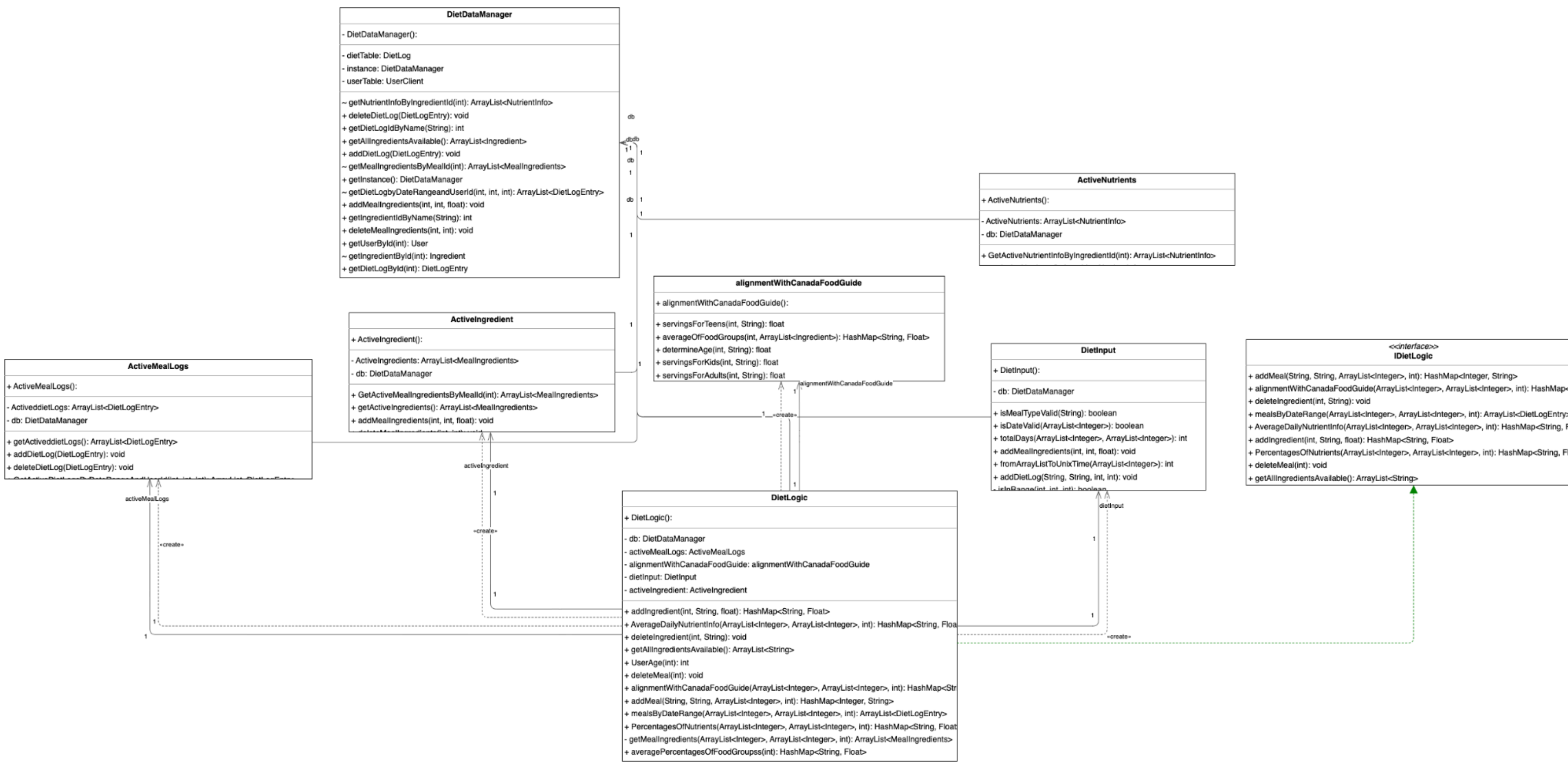


includes

Meal branch MOdules






DietDataManager
- DietDataManager():
- dietTable: DietLog
- instance: DietDataManager
- userTable: UserClient
~ getNutrientInfoByIngredientId(int): ArrayList<NutrientInfo>
+ deleteDietLog(DietLogEntry): void
+ getDietLogIdByName(String): int
+ getAllIngredientsAvailable(): ArrayList<Ingredient>
+ addDietLog(DietLogEntry): void
~ getMealIngredientsByMealId(int): ArrayList<MealIngredients>
+ getInstance(): DietDataManager
~ getDietLogbyDateRangeandUserId(int, int, int): ArrayList<DietLogEntry>
+ addMealIngredients(int, int, float): void
+ getIngredientIdByName(String): int
+ deleteMealIngredients(int, int): void
+ getUserById(int): User
~ getIngredientById(int): Ingredient
+ getDietLogById(int): DietLogEntry
db
dbdb
db
ActiveNutrients
+ ActiveNutrients():
- ActiveNutrients: ArrayList<NutrientInfo>
- db: DietDataManager
+ GetActiveNutrientInfoByIngredientId(int): ArrayList<NutrientInfo>
alignmentWithCanadaFoodGuide
+ alignmentWithCanadaFoodGuide():
+ servingsForTeens(int, String): float
+ averageOfFoodGroups(int, ArrayList<Ingredient>): HashMap<String, Float>
+ determineAge(int, String): float
+ servingsForKids(int, String): float
+ servingsForAdults(int, String): float
alignmentWithCanadaFoodGuide
ActiveIngredient
+ ActiveIngredient():
- ActiveIngredients: ArrayList<MealIngredients>
- db: DietDataManager
+ GetActiveMealIngredientsByMealId(int): ArrayList<MealIngredients>
+ getActiveIngredients(): ArrayList<MealIngredients>
+ addMealIngredients(int, int, float): void
ActiveMealLogs
+ ActiveMealLogs():
- ActiveddietLogs: ArrayList<DietLogEntry>
- db: DietDataManager
+ getActiveddietLogs(): ArrayList<DietLogEntry>
+ addDietLog(DietLogEntry): void
+ deleteDietLog(DietLogEntry): void
activeMealLogs
«create»
activeIngredient
«create»
«create»
DietInput
+ DietInput():
- db: DietDataManager
+ isMealTypeValid(String): boolean
+ isDateValid(ArrayList<Integer>): boolean
+ totalDays(ArrayList<Integer>, ArrayList<Integer>): int
+ addMealIngredients(int, int, float): void
+ fromArrayListToUnixTime(ArrayList<Integer>): int
+ addDietLog(String, String, int, int): void
dietInput
«create»
<<interface>>
IDietLogic
+ addMeal(String, String, ArrayList<Integer>, int): HashMap<Integer, String>
+ alignmentWithCanadaFoodGuide(ArrayList<Integer>, ArrayList<Integer>, int): HashMap
+ deleteIngredient(int, String): void
+ mealsByDateRange(ArrayList<Integer>, ArrayList<Integer>, int): ArrayList<DietLogEntry
+ AverageDailyNutrientInfo(ArrayList<Integer>, ArrayList<Integer>, int): HashMap<String,
+ addIngredient(int, String, float): HashMap<String, Float>
+ PercentagesOfNutrients(ArrayList<Integer>, ArrayList<Integer>, int): HashMap<String, F
+ deleteMeal(int): void
+ getAllIngredientsAvailable(): ArrayList<String>
DietLogic
+ DietLogic():
- db: DietDataManager
- activeMealLogs: ActiveMealLogs
- alignmentWithCanadaFoodGuide: alignmentWithCanadaFoodGuide
- dietInput: DietInput
- activeIngredient: ActiveIngredient
+ addIngredient(int, String, float): HashMap<String, Float>
+ AverageDailyNutrientInfo(ArrayList<Integer>, ArrayList<Integer>, int): HashMap<String, Floa
+ deleteIngredient(int, String): void
+ getAllIngredientsAvailable(): ArrayList<String>
+ UserAge(int): int
+ deleteMeal(int): void
+ alignmentWithCanadaFoodGuide(ArrayList<Integer>, ArrayList<Integer>, int): HashMap<Str
+ addMeal(String, String, ArrayList<Integer>, int): HashMap<Integer, String>
+ mealsByDateRange(ArrayList<Integer>, ArrayList<Integer>, int): ArrayList<DietLogEntry>
+ PercentagesOfNutrients(ArrayList<Integer>, ArrayList<Integer>, int): HashMap<String, Float
- getMealIngredients(ArrayList<Integer>, ArrayList<Integer>, int): ArrayList<MealIngredients>
+ averagePercentagesOfFoodGroupss(int): HashMap<String, Float>

6 Use of Design Patterns

In our current code we use template design pattern for building objects in our database executor classroom which allows for simplifying building objects and let figure out the actual algorithm, while the others only needed to build one method..

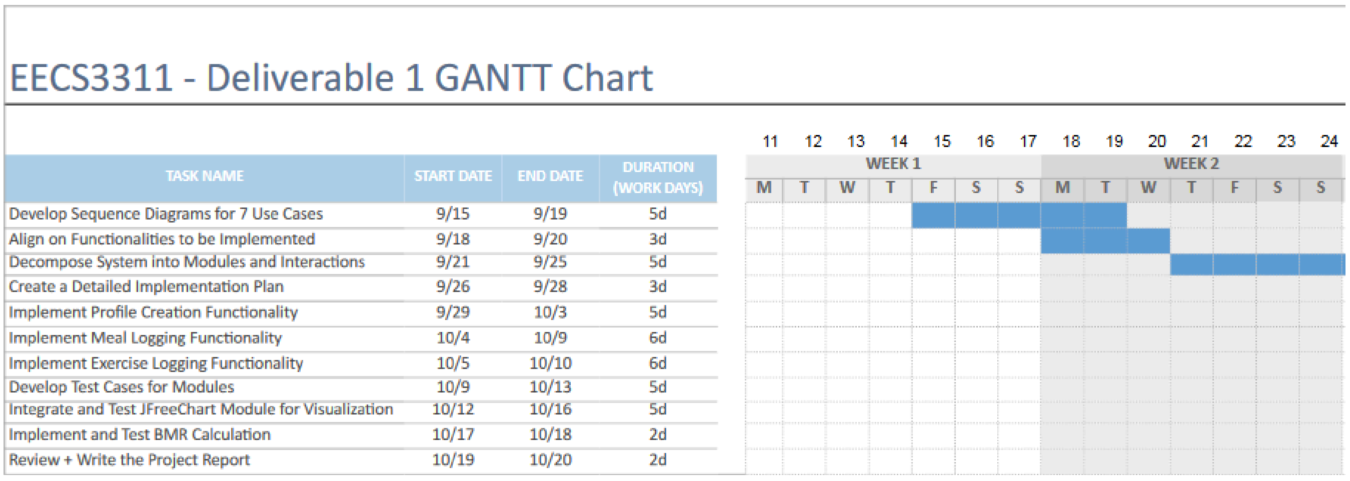
In our Database we use factory design patterns to decide which database to use,

we also use singleton for the manager, if this were to be scaled, we would build a pool of connectors connection manager and ability to have multiple database managers for multiple clients.

In the ExerciseLogic we use a facade to simplify access from the front end.

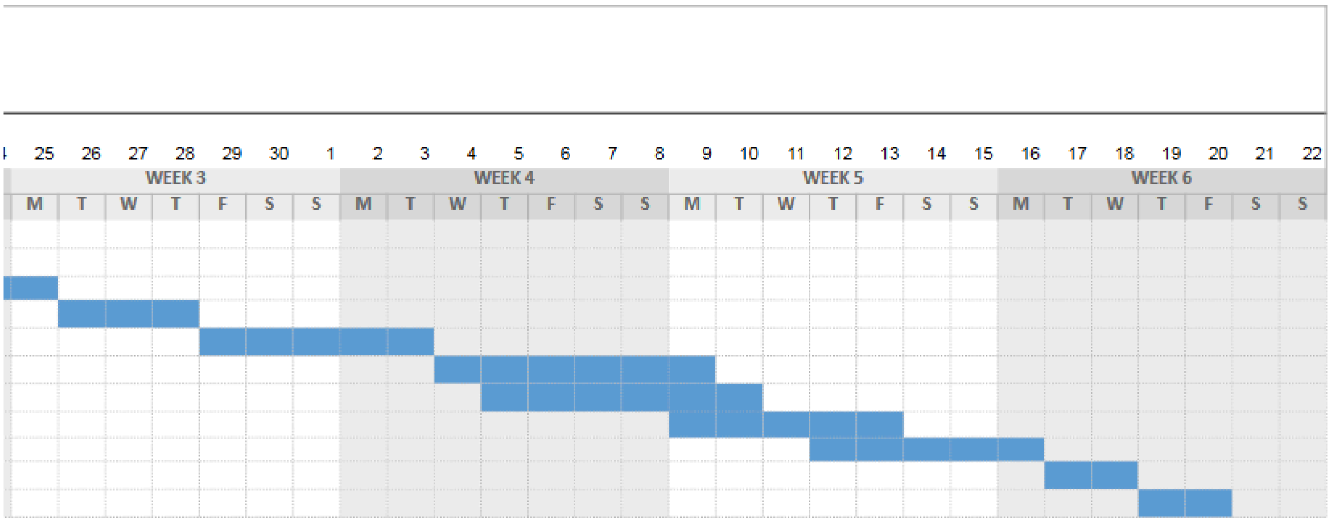
In Exercise FrontEnd we plan on using a template class that simplifies the building process for each component and has default methods for comment elements within the componets

In ExerciseLog Template pattern to simplify building objects, similar to database.



EECS3311 - Deliverable 1 GANTT Chart

TASK NAME	START DATE	END DATE	DURATION (WORK DAYS)
Develop Sequence Diagrams for 7 Use Cases	9/15	9/19	5d
Align on Functionalities to be Implemented	9/18	9/20	3d
Decompose System into Modules and Interactions	9/21	9/25	5d
Create a Detailed Implementation Plan	9/26	9/28	3d
Implement Profile Creation Functionality	9/29	10/3	5d
Implement Meal Logging Functionality	10/4	10/9	6d
Implement Exercise Logging Functionality	10/5	10/10	6d
Develop Test Cases for Modules	10/9	10/13	5d
Integrate and Test JFreeChart Module for Visualization	10/12	10/16	5d
Implement and Test BMR Calculation	10/17	10/18	2d
Review + Write the Project Report	10/19	10/20	2d



7 Activities Plan

1.1 Project Backlog and Sprint Backlog

In this Section, and assuming you follow a Scrum process model, provide a list of product backlog items so that you can select items for your Sprint backlog. Make sure the product backlog list and the tasks in each product backlog item are consistent with the Gantt Chart in Section 6.1. above.

Deliverable 1:

- *Develop sequence diagrams of the use cases*
- *Develop implementation plan*
- *Implement profile creation functionality*
- *Implement meal logging functionality*
- *Implement exercise logging functionality*
- *Develop test cases*
- *Implement and test JFreeChart visualiser*
- *Implement BMR calculator*
- *Write deliverable 1 project report*

Deliverable 2:

- *Implement daily caloric intake visualiser*
- *Implement average daily nutrient breakdown visualiser*
- *Set up the user profile section of the database*
- *Set up the meal nutritional and caloric information section of the database*
- *Set up the exercise section of the database*
- *Integrate the database with the code*
- *Update code design patterns*
- *Update and clean diagrams*
- *Update test cases*
- *Clean up the code*
- *Write deliverable 2 project report*

Deliverable 3:

- *Record demo of application*
- *Create slides about the project architecture*
- *Record a presentation about the project architecture*
- *Combine the two videos*
- *Write deliverable 3 project report*

1.2 Group Meeting Logs

In this Section you write minutes of each meeting, listing the attendance, what the topics of discussion in the meeting were, any decisions that were made, and which team members were assigned which tasks. These minutes must be submitted with the project report in each deliverable and will provide input to be used for the overall assessment of the project.

Present Group Members	**Meeting Date**	**Issues Discussed / Resolved**
Khalifa,	9/13/2023	set last date to get Sequence diagrams done

Joshua, Bjorn		
Khalifa, Joshua, Bjorn	9/29/2023	review use cases and outlined desired functionality. found minimum tables needed set out to start building use case 1,2,3 with simple csv files.(next day decided to build db)
Khalifa, Joshua, Bjorn	10/19/2023	Reviewed Unit 8 microarchitecture. uml diagram was underdeveloped, the development pattern was silo but tried to get everyone working on every portion, so shifted to microarchitecture-esque style focused on functional development, both old model class diagrams for mvc and new architecture is included in this template. looking at layered or multi tiered development. drop (currently unsure how to tie front end classes together. Not refactoring code for deliverable one, but will refactor code asap before continuing development to apply learned skills like smart endpoints, dumb pipes, and more.
Khalifa, Joshua, Bjorn, Marko	11/15/2023	seeting up revised schedule for getting work done - bjorn, khalifa, marko need to identify design patterns in theyre code and add it in the report, if you dont have any(or any new ones) use the one suggested to you. planning out tying up project for monday.

2 Test Driven Development

to open the front end window, run launcher.java in view/src

Auto tests in project are mostly deprecated.

Test ID	The unique Id of the test case
Category	Which part of the system is tested (*e.g. evaluation of user credentials stored on file or DB*)
Requirements Coverage	The unique ID of the requirement tested (*e.g. UC1-Successful-User-Login*)

Initial Condition	Initial conditions required for the test case to run (*e.g. the system has been initiated and runs*)
Procedure	The list of steps required for this test case (*e.g.* *1. The user selects login* *2. The user provides a user name* *3. The user provides a password* *4. The user logs-in into the system and is presented with the main UI window*)
Expected Outcome	The expected outcome of the test case (*e.g. the login form closes, and the user is presented with the main UI window*)
Notes	Any other notes you may want to add for this test case, which are also reflected in the requirements specification (*e.g. the user should provide only alphanumeric usernames and passwords without any special characters*)

Test ID	1.1
Category	Evaluation of parameters to store data into the database
Requirements Coverage	UC1-Successful-Profile-Creation
Initial Condition	The system has been initiated and runs.
Procedure	*1. The user is presented with the login screen* *2. The user selects the "New Profile" option* *3. The user enters a username* *4. The user enters a password* *5. The user enters their sex* *6. The user enters their age* *7. The user enters their height* *8. The user enters their weight*
Expected Outcome	A new profile is created and stored in the database
Notes	The user provides an integer for age and height, and a float for weight.

Test ID	1.2
Category	Edit data in database
Requirements Coverage	UC1-Successful-Editing-Of-Profile
Initial Condition	The system has been initiated and runs, a user profile already exists.
Procedure	*1. The user is presented with a login screen* *2. The user logs in to their profile* *3. The user selects settings* *4. The user selects edit profile* *5. The user selects weight* *6. The user enters a new weights*
Expected Outcome	The new weight of that profile is stored in the database
Notes	The weight entered is a float

Test ID	1.3
Category	Store data in database

Requirements Coverage	UC2-Successful-Storing-Of-Meal
Initial Condition	The system has been initiated and run, a user has logged in.
Procedure	1. *The user selects the log new meal option* 2. *The user enters a date and time* 3. *The user enter a type of meal* 4. *The user enters basic ingredients and quantities*
Expected Outcome	The program calculates calories, the meal is stored in the database, and nutritional information is retrieved from the database
Notes	The user enters valid types such as meal type being breakfast, lunch, dinner, or snack, or the ingredient quantities being floats. A user can have multiple snacks per day, but are limited to only one of the other meals.

Test ID	1.4
Category	Request Data from Database
Requirements Coverage	UC3-Successful-retrieval-of-exercise
Initial Condition	The system has been initiated and run, a user has logged in.
Procedure	1. *The user selects get exercises* 2. *The user enters a date and time* 3. *user confirms*
Expected Outcome	Program returns a list of all stored Exercises for the user, with start time, duration,exercise name, intensity, and calories.
Notes	The user enters valid types such as intensity being low, medium, high, or very high, or exercise start and end time being integers.

Test ID	1.5
Category	Visualization of data
Requirements Coverage	UC4-Exercise-Visualisation
Initial Condition	The system has been initiated and run, a user has logged in. The user may have entered exercise data in the past.
Procedure	1. *The user selects the exercise visualization* 2. *The user selects the start of the visualized period* 3. *The user selects the end of the visualized period*
Expected Outcome	A jfreechart line graph of calories burned from exercise during the time period
Notes	If there is no exercise data for the selected period then the graph will be empty

Test ID	1.6
Category	Store data in database
Requirements Coverage	UC3-Successful-Storing-Of-Exercise
Initial Condition	The system has been initiated and run, a user has logged in.
Procedure	5. *The user selects the log new exercise option* 6. *The user enters a date and time* 7. *The user enters a type of exercise* 8. *The user enters a duration*

	9. The user enters the intensity
Expected Outcome	exercise is logged in database, for project submission, will not be persistent
Notes	The user enters valid types such as intensity being low, medium, high, or very high, or exercise start and end time being integers.

Test ID	1.7
Category	Delete Data from Database
Requirements Coverage	UC3-Successful-Deletion of Exercise
Initial Condition	The system has been initiated and run, a user has logged in.
Procedure	*4. The user selects get exercises* *5. The user enters a date and time* *6. user confirms*
Expected Outcome	Program returns a list of all stored Exercises for the user, with start time, duration,exercise name, intensity, and calories.
Notes	The user enters valid types such as intensity being low, medium, high, or very high, or exercise start and end time being integers.

Category	Database Data Retrieval
Requirements Coverage	*UC2-Successful*-Data-Retrieval
Initial Condition	The system has been initiated and run, a user has logged in.
Procedure	Fetches a diet log entry from the database by its ID and compares it with a predefined entry
Expected Outcome	The test case verifies that the fetched data matches the expected data, and it asserts that the entries do not match
Notes	The user enters valid entries such as Ingredient or meal in string,and quantity in Gram

Category	Database Data Retrieval and Nutrient Calculation
\Requirements Coverage	*UC2-Successful*-Nutrient-Calculation
Initial Condition	The system has been initiated and runs.
Procedure	Iterates over diet log entries in the database, and if a match is found based on the name, it calculates and verifies the total nutrients
Expected Outcome	Verifies that the calculated nutrient values for fetched entries match the expected values.
Notes	The user enters valid entries such as Ingredient or meal in string,and quantity in Gram

Category	Database Data Retrieval

Requirements Coverage	*UC2-Successful*-Diet-intake
Initial Condition	The system has been initiated and runs.
Procedure	Iterates over diet log entries in the database and checks if a specific ingredient is found.
Expected Outcome	Verifies that the system handles the case when an ingredient is not found in the database.
Notes	The user enters valid entries such as Ingredient or meal in string,and quantity in Gram