**EECS3311   
Echo fitness app**Software Design Document

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Summary of Changes** |
| 1.0 | 13/10/23 | Khoa Tran | Document created, general structure outlined, project title pending. |
| 1.1 | 13/10/23 | Adam Mokdad | Added charts for modules and interfaces. |
| 1.2 | 14/10/23 | Alex Valdez | Added test cases. |
| 1.3 | 19/10/23 | Khoa Tran | Added “Major Design Decisions”, some sequence diagrams, and meeting logs. |
| 1.4 | 19/10/23 | Alex Valdez | Added some sequence diagrams and UML diagram. |
| 1.5 | 20/10/23 | Khoa Tran | Added GANTT diagram, link to GitHub, and use case 1 UML Class diagram. |
| 1.6 | 20/10/23 | Adam Mokdad | Added class diagram for use case 2 and 3. |
|  |  |  |  |

# Introduction:

**Purpose:** The goal of the project is to create an application that tracks and calculates a user’s BMR level, BMI level, calorie intake, and nutrition goals.

**Overview:** The software must be able to handle the following use cases.

1. *As a user, I want to be able to create a profile in the application.*
2. *As a user, I want to be able to log my diet data in the application.*
3. *As a user, I want to be able to log my exercise in the application.*
4. *As a user, I want to be able to visualize my calory intake and my exercise over time.*
5. *As a user, I want to be able to visualize my daily nutrient intake.*
6. *As a user, I want to see how much weight in fat I will lose under my current diet and exercise pattern.*
7. *As a user, I want to know how well my diet aligns with the Canada Food Guide.*

**Repository:** <https://github.com/adam5192/EECS3311-Echo>

**References:**

* BMR Calculator Formula: <https://www.calculator.io/bmr-calculator/#the-formula-of-katch-mcardle-3>
* BMI Calculator Formula: <https://www.calculator.io/bmi-calculator/>
* Total Daily Energy Expenditure: <https://www.verywellfit.com/what-is-energy-expenditure-3496103#toc-tdee-calculator>

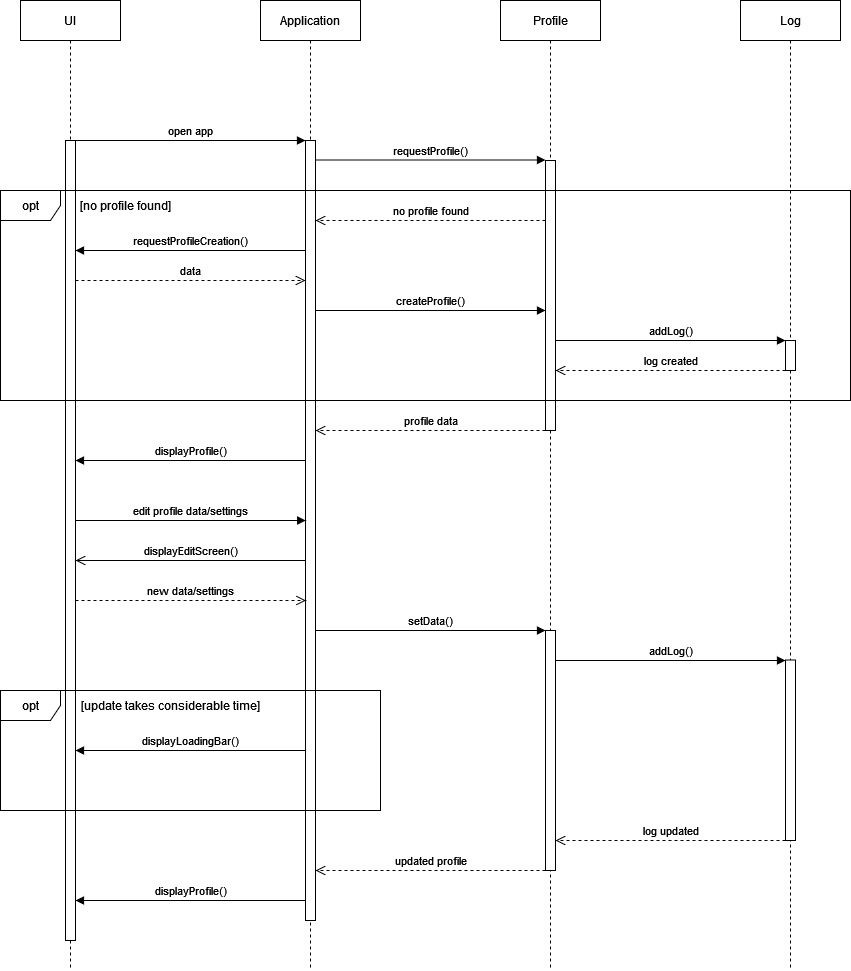
# Major Design Decisions:

The development will be done in Java, as per the recommended programming language of the course. The database management system picked is MySQL and the 2007 Canada Food Guide was chosen as the main database as it is more detailed compared to the alternative option.  
  
The architecture style of the application is MVC where the application is split into a Model (represented by the Log module), a Controller (further split into more specialized modules to handle the different use cases), and a View (which handles the communication between the user and the Controller).

The modules are mostly unaffected by the implementations of the functions of other modules by limiting their interactions through predefined methods. Furthermore, the modules are only responsible for their specialized purposes, such as Profile is only responsible for storing and managing the data related directly to the user and information unrelated (or requires further manipulation to be related) is defined and store outside of its view. In addition, the View’s requests will be pass through a façade class in order to reduce coupling with the modules.

# Sequence Diagrams:

Use case 1:



Use case 2:

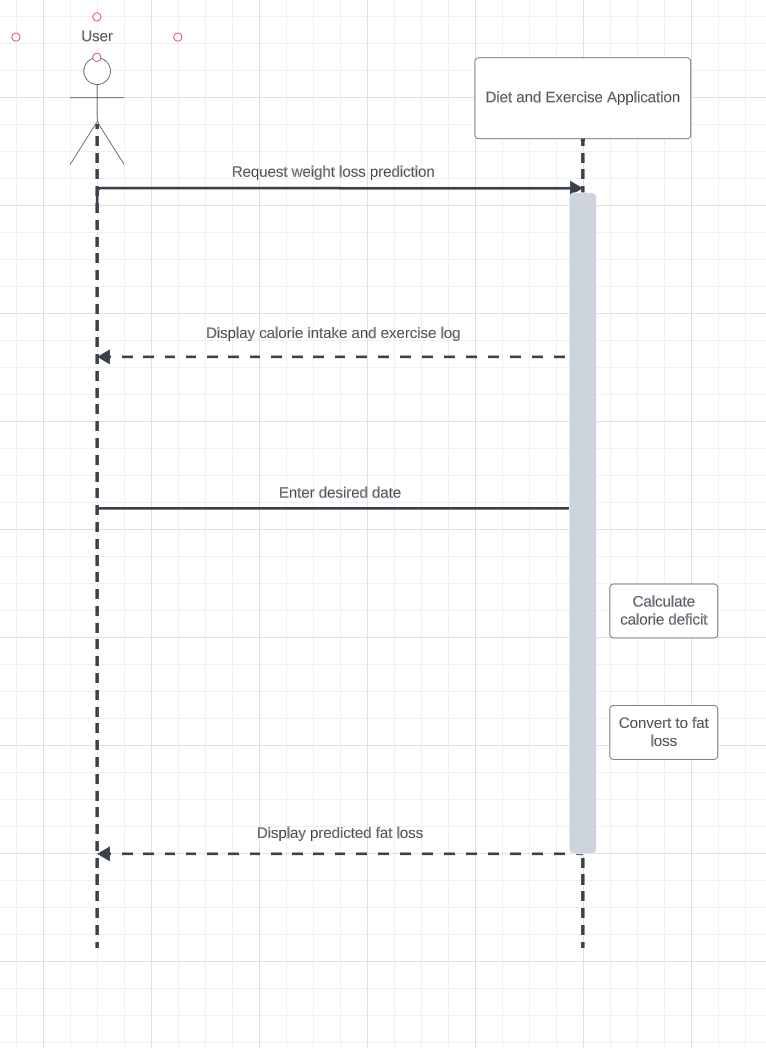
Use case 3:

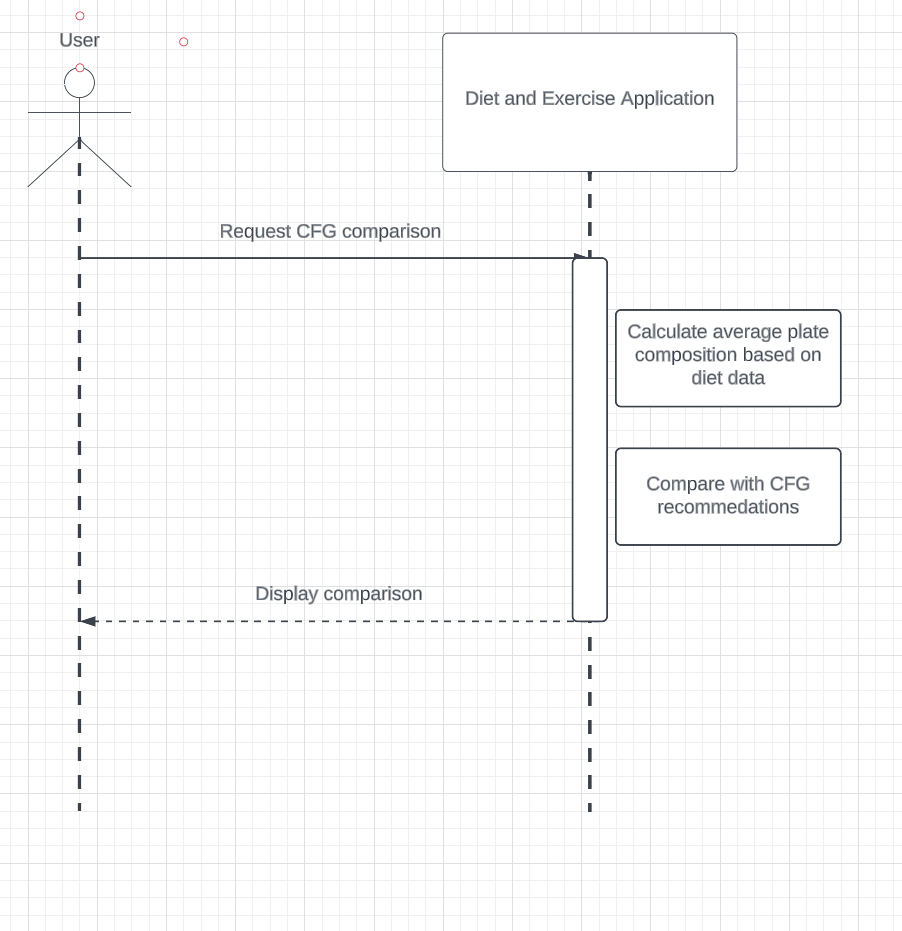
Use case 4:  
A diagram of a calorie intake and exercise

Description automatically generated

Use case 5:  
A diagram of a program

Description automatically generated

Use case 6:  


Use case 7:  


# Architecture:

**Modules**

|  |  |  |  |
| --- | --- | --- | --- |
| **Module Name** | **Description** | **Exposed Interface Names** | **Interface Description** |
| M1: User Profile | Manage user profile (creation, editing, deletion). Manage settings. | M1:I1, M1:I2 | M1:I1: Interface to handle basic user data.  M1:I2: Interface to manage user settings and preferences |
| M2: Dietary and Exercise Logging | Allows users to log dietary and exercise data, and calculate nutritional value and calories burned. | M2:I3, M2:I4 | M2: I3: Interface for data input and nutrition calculation  M2:I4: Interface for logging exercise and calculate calories burned |
| M3: Graphing Visualization | Generates visual data representations for caloric and nutritional data. | M3:I5 | M3: I5: Interface to produce various visualizations |
| M4: Weight Prediction | Uses caloric data to predict potential weight loss | M4:I6 | M4:I6: Predicts weight loss based on caloric data |
| M5: CFG Alignment | Compares dietary data with CFG recommendations | M5:I7 | M5:I7: Evaluates and visualizes alignment with CFG |
| M6: Database | Handles storage, retrieval and management of all user data | M6:I8 | M6:I8: Handles data storage and retrieval |
| M7: UI | Renders UI, handles user inputs, and manages frontend interaction. | M7:I9 | M7:I9: Interfaces for displaying UI and managing user interactions. |

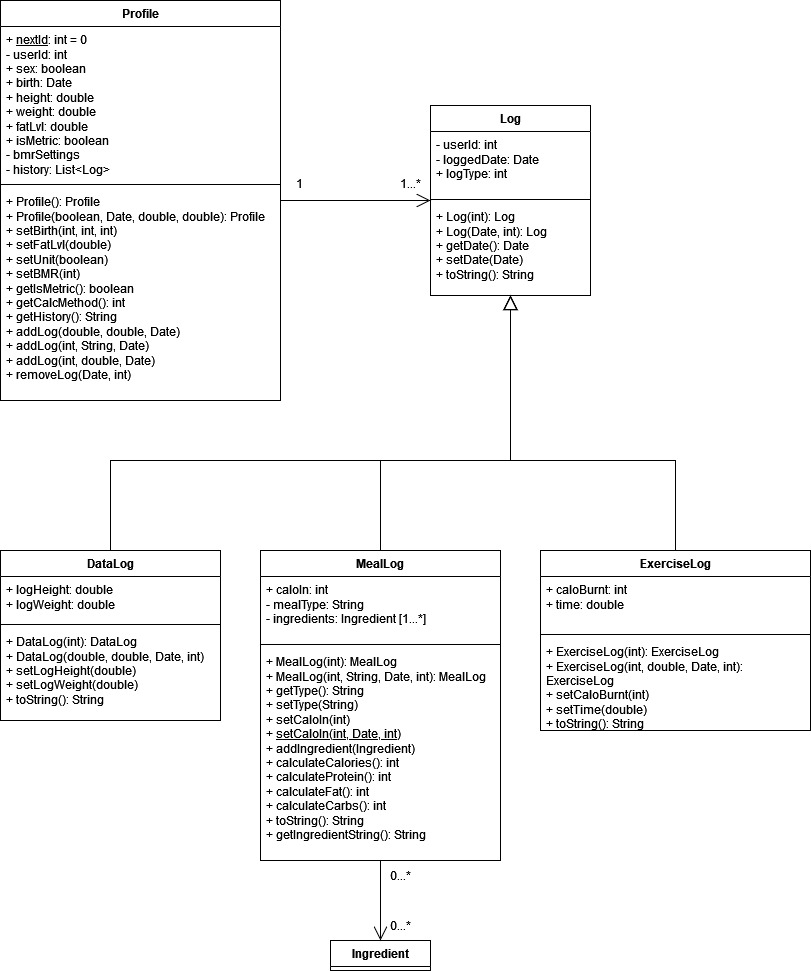
**Interfaces**

|  |  |  |
| --- | --- | --- |
| Interface Name | Operations | Operation Desc |
| M1:I1 | <void> I1:Op1() used by M7  <void> I1:Op2(int x) used by M7 | Op1(): Handle creation and selection of user profiles.  Op2(int x): Modify details in a user’s profile. |
| M1:I2 | <void> I2:Op3() used by M7 | Op3(): Adjust user settings |
| M2:I3 | <Nutrition Data> I3:Op4(String y) used by M7, M4, M5 | Op4(String y): Input dietary data and calculate nutritional values. |
| M2:I4 | <int> I4:Op5(int z) used by M7, M4 | Op5(int z): Log exercise data and calculate calories burned. |
| M3:I5 | <Chart> I5:Op6(Date a, Date b) used by M7 | Op6(Date a, Date b): Generate a visualization for a specified date range. |
| M4:I6 | <float> I6:Op7() used by M7 | Op7(): Predict weight loss using current data. |
| M5:I7 | <CFG Comparison> I7:Op8() used by M7 | Op8(): Compare and visualize comparison with CFG recommendations. |
| M6:I8 | <User Data> I7:Op8() used by M7, M2, M3, M4, M5 | Op9(): Retrieve/store data in the database. |
| M7:I9 | <void>  I9:Op10() used by M1, M2, M3, M4, M5, M6 | Op10(): Render specific UI elements based on user interaction. |

# Class Diagrams and Initial Implementation:

Use case 1:

|  |  |  |
| --- | --- | --- |
| **Class name** | **Attribute/Method name** | **Description** |
| Profile | nextId: static int | For creating unique IDs for each profile instance. |
| userId: int | Unique ID of a Profile instance. |
| sex : boolean birth: Date height: double weight: double fatLvl: double | Fields representing the basic data each profile stores. |
| isMetric: boolean  bmrSettings: int | Fields representing the basic settings of each profile. Used to determine the inputs and outputs of some methods. |
| history: List<Log> | Log instances associated with the profile. |
| Profile Profile()  Profile Profile(boolean, Date, double, double) | Constructors. Create a new profile with specific data. For the default constructor, Profile(true, null, 0.0, 0.0) is called. |
| void setSex(boolean) void setBirth(Date) void setHeight(double) void setWeight(double) void setFatLvl(double)  void setUnit(boolean)  void setBMR(int) | Setter methods. Some might get removed if not used outside of Profile’s self-calls/edits. |
| boolean getSex() Date getBirth()  double getHeight()  double getWeight()  double getFatLvl()  boolean getIsMetric() int getCalcMethod() | Basic getter methods. |
| String getHistory() | Returns all the logs associated with the profile as a String instance. |
| void addLog(double, double, Date)  void addLog(int, String, Date) void addLog(int, double, Date) | Overloaded methods for adding new logs to the profile, each is responsible for generating a specific type of log. |
| Log removeLog(Date, int) | Find the first instance of Log in history that has the same Date and type specified, remove it, and return it to the user. |
| Log | userId: int  loggedDate: Date  logType: int | Fields associated with all logs. userId is currently unused (possibly to be used to store and access logs from a database structure). logType is to differentiate between subclasses of Log instances (0 for Log, 1 for DataLog, 2 for MealLog, 3 for ExerciseLog) |
| Log Log(int) Log Log(Date, int) | Basic constructors, create the basic values common to all logs.  If no Date instance is provided, store a Date instance associated with the time the method was called. |
| setDate(Date) setLogType(int) | Basic setter methods. |
| Date getDate()  int getLogType() | Basic getter methods. |
| String toString() | Returns a String in the format “YY/MM/DD” |
| DataLog | logHeight: double logWeight: double | Fields specific to DataLog in addition fields specified by Log. |
| DataLog DataLog(int) DataLog DataLog(double, double, Date, int) | Constructors. Call the superclass constructor and store the value given (if none, stores the value 0.0 for height and weight) |
| setLogHeight(double) setLogWeight(double) | Basic setter methods.  Only takes non-negative values. If a negative value is given, throw an InvalidAttributeException. |
| double getLogHeight()  double getLogWeight() | Basic getter methods. |
| String toString() | Returns a String in the format “YY/MM/DD – Height, Weight” |
| MealLog | caloIn: int mealType: String ingredients: List<Ingredient> | Fields associated with the log to represent the calorie value, when the meal was eaten/prepared, and the ingredients used. |
| MealLog(userId) MealLog(int, String, Date, int) | Constructors. Call the superclass constructor and the store the value given (if none, stores 0 and “Breakfast” for caloIn and mealType respectively) |
| setType(String) setCaloIn(int) | Basic setter methods. mealType’s valid values are limited to “Breakfast”, “Lunch”, “Dinner”, and “Snack”. caloIn value cannot be negative. |
| String getType() int getCaloIn() | Basic getter methods. |
| addIngredient(Ingredient) | Add a new Ingredient instance to the List. |
| int calculateCalories()  int calculateProtein()  int calculateFat()  int calculateCarbs() | Calculate the total calorie, protein, fat, and carb values of the meal. |
| String toString() | Returns a String in the format “YY/MM/DD – Meal type – Calorie Intake” |
| String getIngredientString() | Returns a String that is the list of Ingredient instance stored in ingredients following their toString() method. |
| ExerciseLog | caloBurnt: int  time: double | Fields associated with an exercise log. |
| ExerciseLog(int) ExerciseLog(int, double, Date, int) | Constructors. Call the superclass constructor and further assign values to caloBurnt and time (default is 0 and 0.0). |
| setCaloBurnt(int) setTime(double) | Basic setter methods. The value of caloBurn and time cannot be negative. |
| int getCaloBurnt()  double getTime() | Basic getter methods. |
| String toString() | Returns a String in the format “YY/MM/DD – Time – Calorie burnt” |



Use Case 2:  
  
A screenshot of a computer program

Description automatically generated

Use Case 3:

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Class name** | **Attribute/Method name** | **Description** |
| CEGraph | private static CEGraph instance  private String startDate  private String endDate  private JLabel inputDate  private JTextField start  private JLabel to  private JTextField end  private JLabel example  private JButton graph  Public static getInstance()  Public createTimeSeries(JPanel, startDate, endDate) | This class creates the GUI that asks the user the time period and uses another method to create the graph about their daily calory intake and daily exercise |

A screenshot of a computer

Description automatically generatedUse case 5:

|  |  |  |
| --- | --- | --- |
| **Class name** | **Attribute/Method name** | **Description** |
| DNGraph | private static DNGraph instance  private String startDate  private String endDate  private JLabel inputDate  private JTextField start  private JLabel to  private JTextField end  private JLabel example  private JButton graph  Public static getInstance()  Public createPie(JPanel, startDate, endDate) | This class creates the GUI that asks the user the time period and uses another method to create the graph based on nutritional intake |

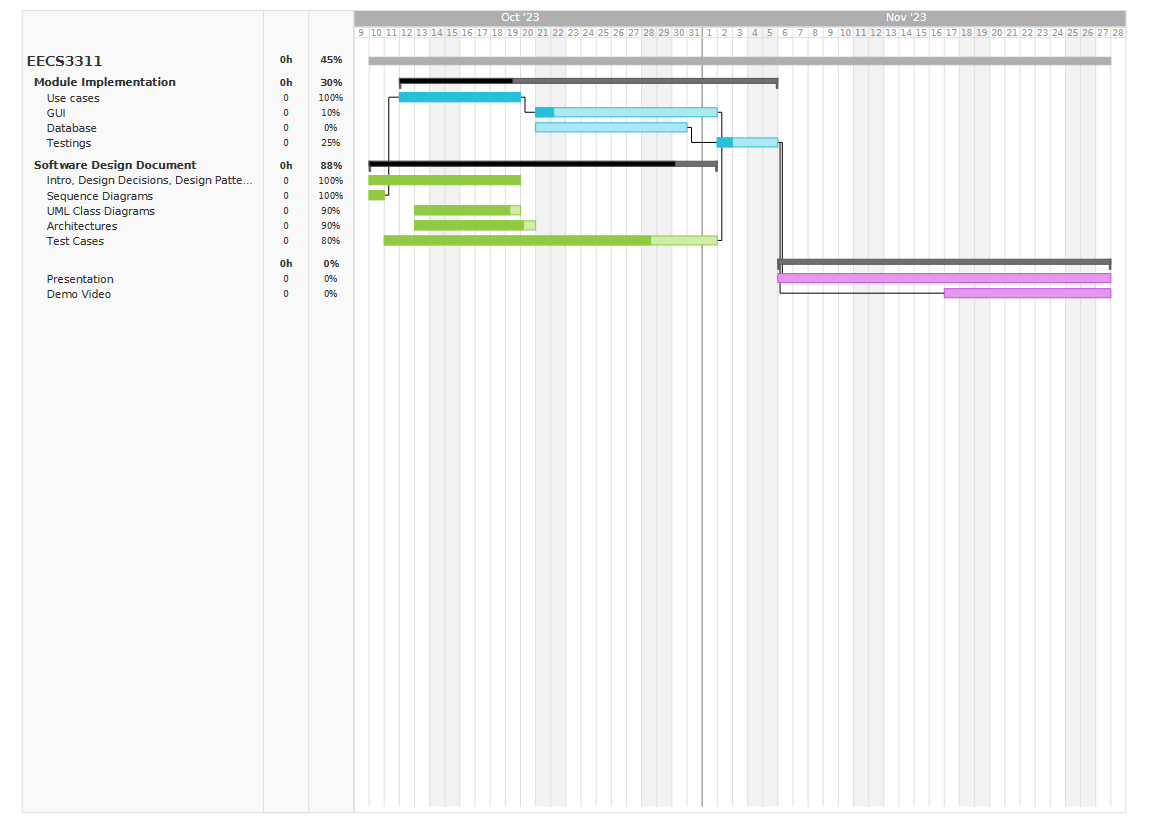
Use case 6:  
A screenshot of a computer

Description automatically generated

# Design Patterns:

* Structural Patterns:
  + Façade: To decouple the logic modules from the view.
* Behavioral Patterns:
  + Chain of Responsibility: To deal with the possible need of requiring multiple modules to handle a request.
  + Strategy: Alternative to chain of responsibility.
* Creational Patterns:
  + Builder: To reduce the complexity of the Log constructors (to be implemented)

# Activities Plan, Product Backlog, and Sprint Backlog:

**GANTT Diagrams:**  


**Group Meeting Logs:**

* Oct 10th, 6:30pm:
  + Time: 30min
  + Attendance: All
  + Discussed how to split use cases to design sequence diagrams.
  + Decisions:
    - Assignment:
      * Use case 1: Khoa Tran
      * Use case 2-3: Omer Omer
      * Use case 4-5: Alex Valdez
      * Use case 6-7: Adam Mokdad
    - Finish first draft of sequence diagrams by Oct 11th.
* Oct 11th, 6:30pm:
  + Time: 1h
  + Attendance: All
  + Showing of first draft of sequence diagrams.
  + Established the modules required for the application and their basic classes.
  + Decisions:
    - Further assignment of work:
      * Architecture Summary Table: Adam Mokdad
      * Component Diagram for architecture: Omer Omer
      * SDD – Introduction and basic structures: Khoa Tran
      * Database accessor: Khoa Tran
      * Basic test cases: Alex Valdez
    - Next meeting/progress report on Oct 13th.
* Oct 13th, 2:00pm:
  + Time: 45min
  + Attendance: All
  + Decision:
    - Major Design Decision:
      * Coding: Java
      * Database: MySQL
      * Canada Food Guide: 2007 version (more detailed)
    - Discussed the design patterns being used: Builder, Façade, Bridge
    - Next meeting/progress report on Oct 16th.
* Oct 16th, 9:00pm:
  + Time: 30min
  + Attendance: All
  + Discussed the goal for the deadline of the Deliverable 1.
  + Decision:
    - Implementations of main methods for all modules (for testing).
    - Major Design Decision:
      * MVC (to be changed)
    - Implementations of basic MET values.
    - Change in implementation of nutrient intake to better meet use case’s need.
    - Final meeting before Deliverable 1 planned.
* Oct 19th, 10:00pm:
  + Time:
  + Attendance: All
  + Quick discussion final necessary components of Deliverable 1.
  + Decisions:
    - Creations of UML class diagrams specific to each module.
    - Completion of the SDD.

# Test Driven Development:

|  |  |
| --- | --- |
| Test ID | 1 |
| Category | Evaluation of parameters to create a profile |
| Requirements Coverage | UC1-Successful-profile-creation |
| Initial Condition | System has been initiated and runs |
| Procedure | 1. User provides name 2. User provides date of birth 3. User provides height 4. User provides weight 5. User clicks create profile |
| **Expected Outcome** | The users profile should be added and will be displayed in the splash screen |
| **Notes** | User should provide valid input |

|  |  |
| --- | --- |
| Test ID | 2 |
| Category | Evaluation of parameters to log diet data |
| Requirements Coverage | UC2-Successful-data-load |
| Initial Condition | System has been initiated and runs,user is able to access their profile and its data |
| Procedure | 1. User inputs a meal 2. User inputs a date 3. User inputs an amount 4. User clicks log data |
| **Expected Outcome** | Data about the meals calories, food group and nutrients is correctly fetched from the database |
| **Notes** | User should provide valid input |

|  |  |
| --- | --- |
| Test ID | 3 |
| Category | Evaluation of parameters to catch when a meal is not known |
| Requirements Coverage | UC3-Successful-error-catch |
| Initial Condition | System has been initiated and runs,user is able to access their profile and its data |
| Procedure | 1. User inputs a meal (\*that is not in the database) 2. User inputs a date 3. User inputs an amount 4. User clicks log data |
| **Expected Outcome** | The data is not fetched and an message is displayed |
| **Notes** | User should provide valid input |

|  |  |
| --- | --- |
| Test ID | 4 |
| Category | Evaluation of parameters to catch when input is not provided |
| Requirements Coverage | UC4-Successful-data-load |
| Initial Condition | System has been initiated and runs,user is able to access their profile and its data |
| Procedure | 1. User inputs a meal 2. User inputs a date 3. User forgets to input amount or inputs an amount of zero 4. User clicks log data   \*user could forget to input meal, date and/or amount |
| **Expected Outcome** | The data is not fetched and a message is displayed |
| **Notes** |  |

|  |  |
| --- | --- |
| Test ID | 5 |
| Category | Evaluation of parameters to exercise data |
| Requirements Coverage | UC5-Successful-data-load |
| Initial Condition | System has been initiated and runs,user is able to access their profile and its data |
| Procedure | 1. User inputs a date and time 2. User inputs the type of exercise 3. User inputs duration 4. User inputs the intensity 5. User clicks log exercise data |
| **Expected Outcome** | Data about the exercise is correctly fetched form the database |
| **Notes** | User should provide valid input |

|  |  |
| --- | --- |
| Test ID | 6 |
| Category | Evaluation of parameters to catch when an exercise is unknown |
| Requirements Coverage | UC6-Successful-error-catch |
| Initial Condition | System has been initiated and runs,user is able to access their profile and its data |
| Procedure | 1. User inputs a date and time 2. User inputs the type of exercise (\*that is not in the database) 3. User inputs duration 4. User inputs the intensity 5. User clicks log exercise data |
| **Expected Outcome** | Data is not fetched and a message is displayed |
| **Notes** | User should provide valid input |

|  |  |
| --- | --- |
| Test ID | 7 |
| Category | Evaluation of parameters to catch when an exercise is unknown |
| Requirements Coverage | UC7-Successful-error-catch |
| Initial Condition | System has been initiated and runs,user is able to access their profile and its data |
| Procedure | 1. User inputs a date and time 2. User inputs the type of exercise 3. User inputs duration 4. User forgets to input the intensity 5. User clicks log exercise data   \*user could forget to input date and time, type of exercise, duration and/or intensity |
| **Expected Outcome** | Data is not fetched and a message is displayed |
| **Notes** | User should provide valid input |

|  |  |
| --- | --- |
| Test ID | 8 |
| Category | Evaluation of parameters to calculate calories burnt |
| Requirements Coverage | UC8-Successful-calcualtion |
| Initial Condition | System has been initiated and runs, exercise data is accurately logged |
| Procedure | 1. System access exercise data 2. Calculates calories burnt |
| **Expected Outcome** | Amount of calories burnt is calculated accurately and saved in the database |
| **Notes** |  |

|  |  |
| --- | --- |
| Test ID | 9 |
| Category | Evaluation of parameters to calculate BMR |
| Requirements Coverage | UC9-Successful-BMR-calcualtion |
| Initial Condition | System has been initiated and runs, exercise data is accurately logged |
| Procedure | 1. System access exercise data 2. Calculates BMR |
| **Expected Outcome** | BMR is calculated accurately and saved in the database |
| **Notes** |  |

|  |  |
| --- | --- |
| Test ID | 10 |
| Category | Evaluation of parameters to input a time period |
| Requirements Coverage | UC10-Successful-data-load |
| Initial Condition | System has been initiated and runs |
| Procedure | 1. User inputs start date 2. User inputs an end date 3. User clicks confirm |
| **Expected Outcome** | Time period is logged and used in use case four,five,six or seven |
| **Notes** | User should provide valid input |