LifeExtender+

Group #16
Report #2

github.com/SE2017/LifeExtenderPlus

Trirmadura J. Ariyawansa
John Eng
Daniel Huang
Chris Kim
Kevin Lee
Kyungsuk Lee

Submission Date: 3/12/2017

Individual Contribution Breakdown

Task	Trirmadura Ariyawansa	John Eng	Daniel Huang	Chris Kim	Kevin Lee	Kyungsuk Lee
UML Diagrams (10)	33%	33%				33%
Prose Description Of Diagram (10)			33%	33%	33%	
Class Diagram (5)	50%		50%			
Data Types and Operation Signatures (5)	45%		45%			10%
Traceability Matrix				100%		
Architectural Styles (5)		33%		33%	33%	
Identifying Subsystems (2)		50%		50%		
Mapping Subsystems to Hardware (2)		50%		25%	25%	
Persistent Data Storage (3)						100%
Network Protocol (1)					100%	
Global Control Flow (1)					100%	
Hardware Requirements (1)						100%
Alg's & Data Structure (4)		40%	30%		30%	
Appearance (6)		100%				
Prose Description Of UI (5)	33%		33%		33%	

Testing Design (12)				50%	50%	
Document Merge (11)	30%		20%			40%
Project Coordination/ Progress (5)						100%
Plan of Work (2)	100%					
References (-5)					100%	

Point Distribution:

Trirmadura Ariyawansa:

10(0.33) + 5(0.50) + 5(0.45) + 5(0.33) + 2(1.00) + 11(0.30) = 15.00

John Eng:

10(0.33) + 5(0.33) + 2(0.50) + 2(0.50) + 4(0.40) + 6(1.00) = 14.55

Daniel Huang:

10(0.33) + 5(0.50) + 5(0.45) + 4(0.30) + 5(0.33) + 11(0.20) = 13.10

Chris Kim:

10(0.33) + 5(0.33) + 2(0.50) + 2(0.25) + 12(0.50) = 12.45

Kevin Lee:

10(0.33) + 5(0.33) + 2(0.25) + 1(1.00) + 1(1.00) + 4(0.30) + 5(0.33) + 12(0.50) = 16.30

Kyungsuk Lee:

10(0.33) + 5(0.10) + 3(1.00) + 1(1.00) + 11(0.40) + 5(1.00) = 17.20

Table of Contents

Summary of Changes

Based on several factors, we saw the impracticalities of the functions that we were trying to implement. After viewing the feedback of the first report and working more in depth with the project, we decided to change the formula for calculating the health index. User will be receiving only for when to sleep, eat, active periods, and events (in their schedule). The app will track only active periods (time at the gym and time walking/running). The sharing of the health index is now a setting to public and private to be seen by friends. Other changes include the changes to the Use-Cases and the choice for the Fully-Dressed ones.

1. Interaction Diagrams

UC-1: Get Health Index

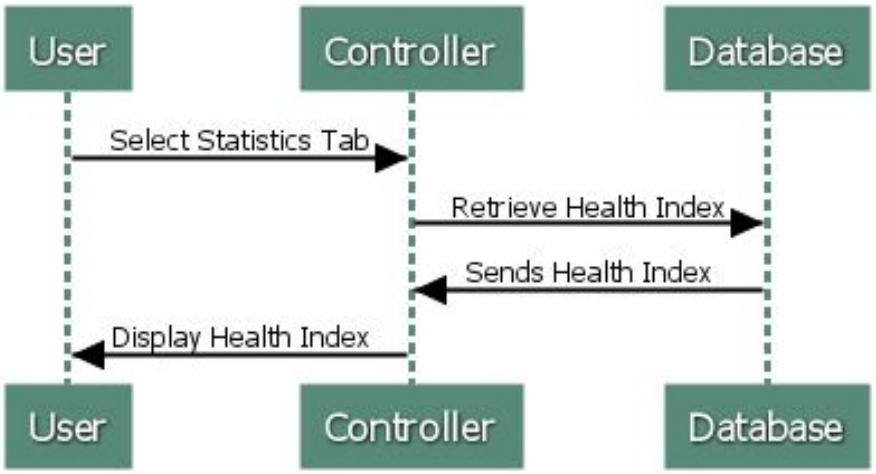


Figure 1-1

The UC-1 diagram for “Get Health Index” shows how the user receives the Health Index Number from the application. When the user selects the statistics tab in the app, the controller will send a request to the database to retrieve the user’s health index. Afterwards, the database will send the specific user’s health index back to the controller. The controller will then display the health index to the user.

UC-7: Change Setting

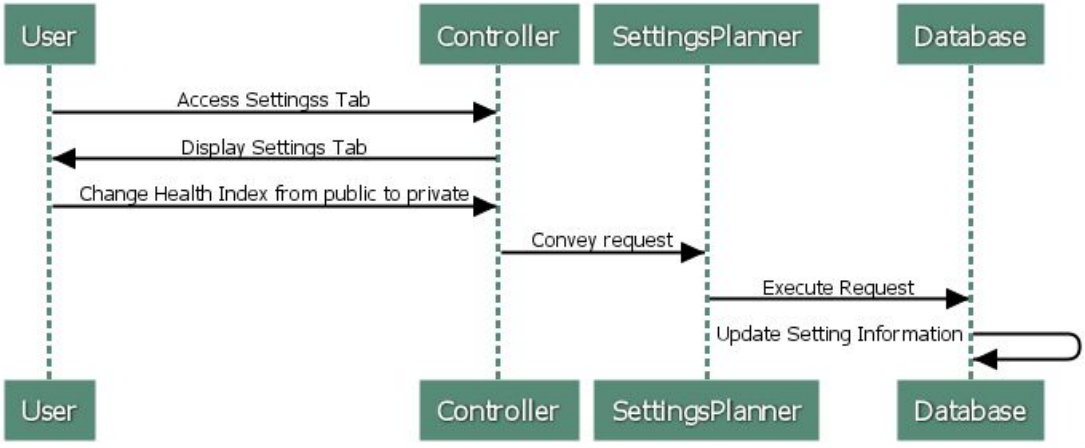


Figure 1-2

The UC-7 diagram for “Change Setting” shows how the user is able to manage his or her personal settings, such as allowing other users to see the user’s health index. When the user selects the settings tab, the controller will display the settings page which will include various features of the application that the user may change that fits his or her preference. The controller will send a request to SettingsPlanner after the user makes any changes, and the database and app functionality will be updated accordingly.

UC-8:Sets Goal

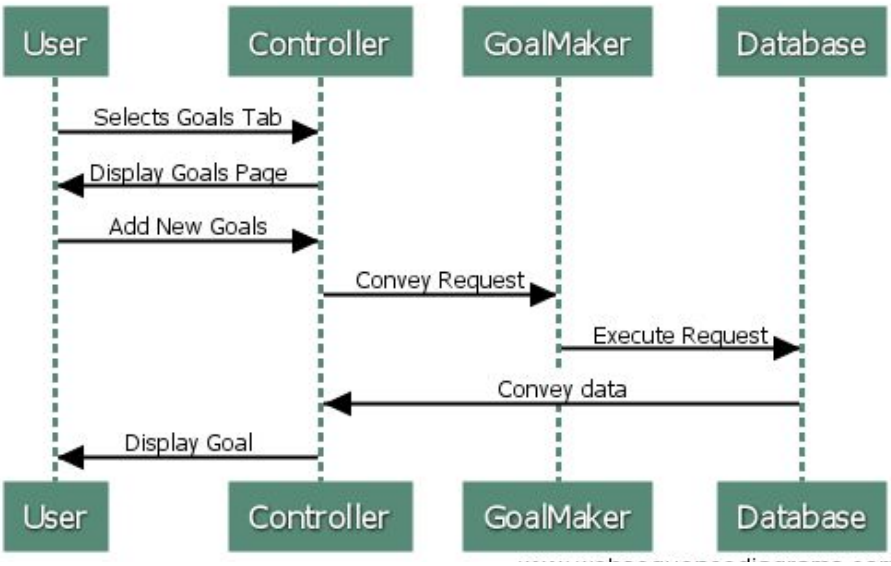


Figure 1-3

The UC-8 diagram for “Sets Goal” shows how the user may go about setting goals with the LifeExtender+ app. To start, the user will select the goals tab to bring up the goals page, which brings up the user’s goals and their progress towards those goals. From the goals page, the user will be able to add new goals via the app’s Controller. After entering all the information through the Controller, the Controller will send the information to the GoalMaker, which will create the goal through the Database. From there, the Database will return the goal through the Controller, which will display it to the user. ``

UC-9:Quick Change

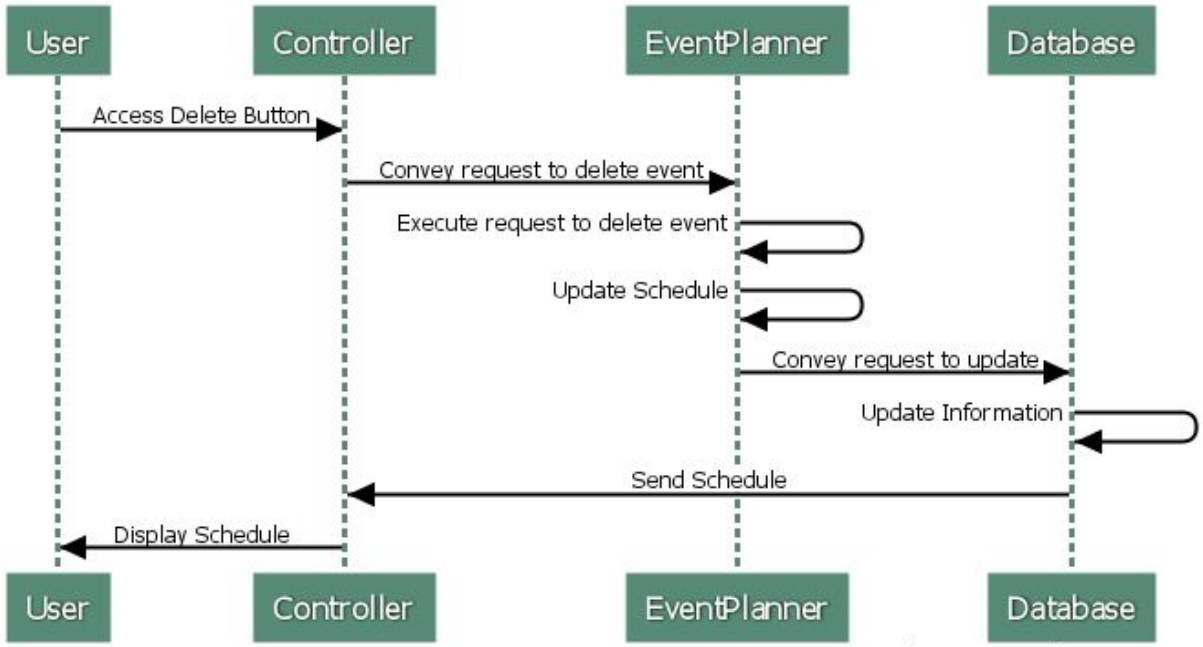


Figure 1-4

The UC-9 diagram for “Quick Change” shows the process that the user goes through in order to change a part of their set schedule. First, the user hits the delete button, which prompts the controller to request the event planner to delete the user’s selected event. The event planner will then execute that request, and update the particular user’s schedule. The event planner will then send the updated schedule to the Database. The database will then update the user’s information and send the updated schedule to the controller. Finally, the controller displays the updated schedule to the user.

2. Class Diagram and Interface Specification

a. Class Diagram

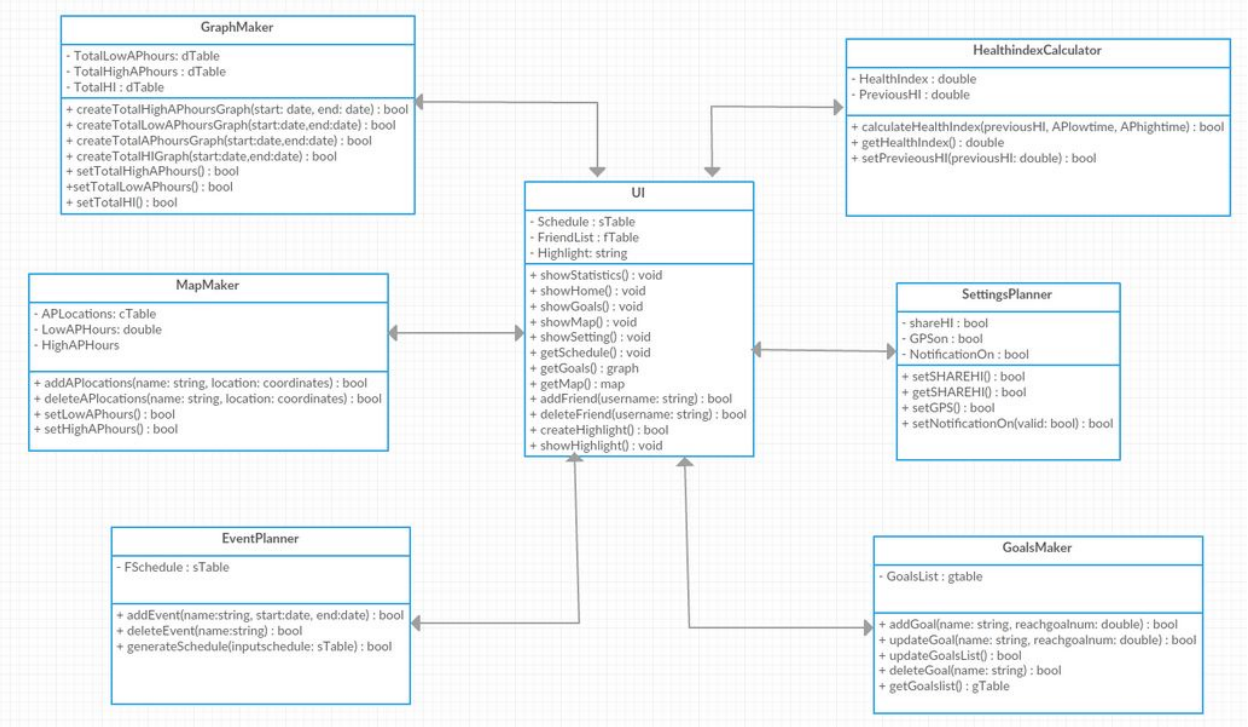


Figure 2-1

b. Data Types and Operation Signatures

UI
Variables:

- Schedule: sTable. Stores the schedule into database. Necessary to display schedule on homepage
- FriendList:fTable. User's friend list which is synchronized with database.
- Highlight:string. User's highlight of the day

Functions:

- showStatistics(): void. Shows user's statistics under statistics tab
- showHome(): void. Shows user's schedule and homepage under home tab
- showGoals(): void. Shows user's goals under goal tab
- showMap(): void. Shows user's map under location tab
- showSetting(): void. Shows user's settings under settings tab
- getSchedule(): void. Retrieves schedule from controller
- getGoals(): graph. Retrieves goals from controller
- getMap(): map. Retrieves map from controller
- addFriend(username:string): bool. Adds a friend to User's FriendList according to username. Return true if successful (if friends accepts request).
- deleteFriend(username:string): bool. Deletes a friend from User's Friendlist according to username. Return true if successful.
- createHighlight():bool.Creates daily highlight using User's fitness data from the server. Return true if successful .
- showHighlight():void. Show's User's highlight of the day

HealthIndexCalculator

Variables:

- HealthIndex:double. User's Current health index
- PreviousHI:double. User's previous day health index

Functions:

- calculateHealthIndex(previousHI,APlowtime,APhightime): bool. Calculates the user's health index given the previous health index, and the Active period of the user. Synchronized with database.
- getHealthIndex():double. Gets the health index from the server
- setPreviousHI(previousHI:double):bool. Gets the previous day's health index from database.

EventPlanner

Variables:

- fSchedule: sTable. User's schedule with Active Period. Synchronized with database

Functions:

- addEvent(name:string,start:date,end:date): bool - Adds an event to the schedule at given date, returns true or false if the event is able to be added to the schedule or not
- deleteEvent(name:string): bool - Removes the event with the given name, returns true or false dependent on success.
- generateSchedule(inputschedule:stable):bool - Creates a schedule with Active Period given a user input of multiple events and returns a boolean value on success. Returns true if successful

GoalsMaker
Variables: ● GoalsList: gTable. User's goals which is put into a goal table data type, which is synchronized with database
Function ● addGoal(name:string,reachgoalnum:double):bool. Adds a goal to goalsList with the name,and personal record value to reach). Returns true if successful ● updateGoal(name:string,reachgoalnum:double):bool. Update a goal's PR value in the GoalsList. Returns true if successful ● updateGoalsList():bool. Update GoalList from database. Returns true if successful ● deleteGoal(name:string):bool. Delete a certain goal by name. Returns true if successful ● getGoalslist():gtable. Gets GoalsList.

GraphMaker
Variables ● TotalLowAPhours: dTable. All of User's low intensity Active Period times according to date in a date table data type. Synchronized with database ● TotalHighAPhours: dTable. All of User's high intensity Active Period times according to date in a date table data type. Synchronized with database ● TotalHI: dTable. All of User's Health Indexes according to date in a date table data type. Synchronized with database
Function ● createTotalHighAPhoursGraph(start:date,end:date): bool. Shows a Graph of User's total time of High intensity Active Periods between a certain time period. Returns true if successful ● createTotalLowAPhoursGraph(start:date,end:date): bool. Shows a Graph of

User's total time of Low intensity Active Periods between a certain time period. Returns true if successful
- createTotalAPhoursGraph(start:date,end:date): bool. Shows a Graph of User's total time of Low and High intensity Active Periods between a certain time period. Returns true if successful
- createTotalHIGraph(start:date,end:date): bool. Shows a Graph of User's Health Indexes between a certain time period. Returns true if successful
- setTotalLowAPhours(): bool. Fetches TotalLowAPhours dtable from the database.
- setTotalHighAPhours(): bool. Fetches TotalLowAPhours dtable from the database.
- setTotalHI(): bool. Fetches User's Health indexes dtable from the database.

MapMaker
Variables ● APlocations: cTable. User's Active Periods' location coordinates in a coordinate table data type. Synchronized with database ● LowAPHours: double. User's daily low intensity Active Period time ● HighAPHours: double. User's daily high intensity Active Period time
Function ● addAPlocations(name:string,location:coordinates):bool. Add an Active Period location according to name and coordinates into APlocations variable which is synchronized with database. ● deleteAPlocations(name:string,location:coordinates):bool. Delete an Active Period location according to name and coordinates from APlocations variable which is synchronized with database. ● setLowAPhours() : bool. Sets User's daily low intensity Active Period Time using GPS pinging which is synchronized with database. Return true if successful. ● setHighAPhours() : bool. Sets User's daily High intensity Active Period Time using GPS pinging which is synchronized with database. Return true if successful.

SettingsPlanner
Variables ● shareHI: bool. If true the User's Health index is shared to friends, otherwise its stay private. Synchronized with database

• GPSOn: bool. If true, GPS can be used for the app , otherwise its can't be used. Synchronized with database • NotificationOn: bool. If true, push notifications will be on, otherwise it won't be on.
Function • setSHAREHI(valid:bool):bool. Set the shareHI boolean value. Returns true if successful. • getSHAREHI():bool. Get the value of shareHI boolean variable • setGPSOn(valid:bool):bool. Set the GPSOn boolean value. Returns true if successful. • setNotificationOn(valid:bool): bool. Set the NotificationOn boolean value. Returns true if successful.

c. Traceability Matrix

Domain Concepts / Classes	HealthIndexCalc	EventPlanner	GraphMaker	GoalsMaker	MapMaker	SettingsPlanner
Interface		**x**	**x**	**x**	**x**	
HealthIndex Calculator	**x**					
HealthIndex Storage	**x**					**x**
GraphMaker			**x**			
AnalyticData Goals				**x**		
ScheduleMaker		**x**				
Controller		**x**	**x**	**x**	**x**	**x**
ScheduleStored		**x**				
LocationStorage			**x**		**x**	
HighlightMaker				**x**		

In our traceability matrix, we show how multiple domain concepts that are important in performing a similar function are grouped into a single class. For example, the HealthIndexCalculator and the HealthIndexStorage domain concepts have been combined into one class called HealthIndexCalc which is responsible for calculating, setting, and getting the user's health index. Certain domain concepts, however, were not used in any classes. Instead, domain concepts that dealt with login entry and friend requests are handled by the UI subsystem.

3. System Architecture and System Design

a. Architectural Styles

Our system uses a 3-layer architecture consisting of a presentation layer, an application layer, and a data layer. This is analogous to the frontend/backend in website development, where the presentation layer is what is seen by the user, and the application and data layers work behind the scenes.

This scheme is the most logical as the layers are abstracted from each other and will parallel into the code. The differences in UI design, application code, and database calls are apparent. Our project is constructed through Android Studio, where the UI is designed through the program's graphical interface. The application will be "connected" where different parts of UI are matched with the application code. Within the code, database calls will be made with using an API. This structure is easy to follow and implement.

b. Identifying Subsystems

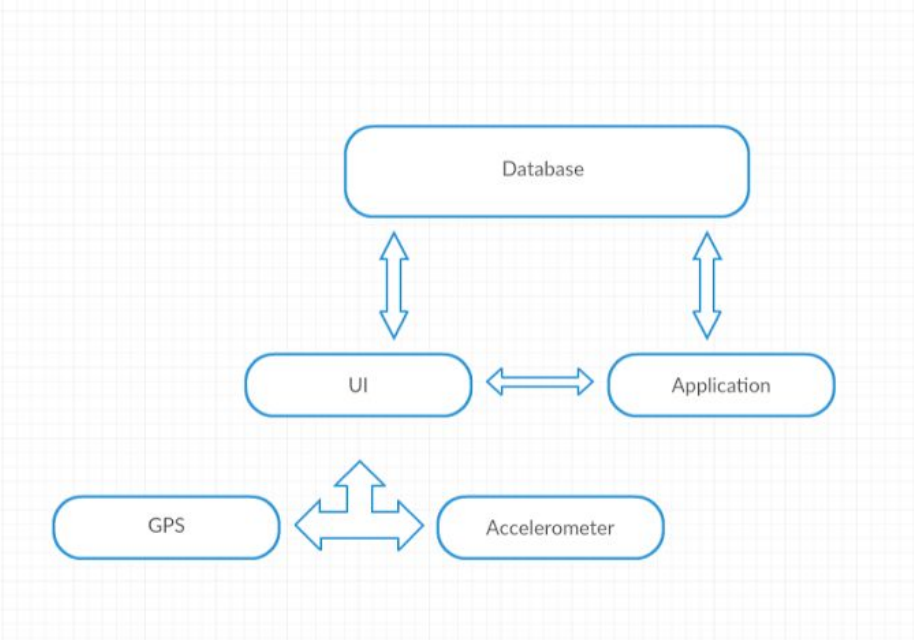


Figure 3-1

The UI subsystem will interact with the GPS and accelerometer subsystems. The Database subsystem saves the profile data of each user and will interact with both the application and database subsystems. The Application layer will handle inputs from the user and interact with the UI subsystem and Database subsystem. This includes the scheduling algorithm, settings modification, the generation of graphs, and calculation of the health index.

c. Mapping Subsystems to Hardware

Our application works with a native device (Android) and a database server. Our application will utilize a database with a public API and will run as long as it is active. The Database subsystem will naturally run the database with the Application subsystem interacting with it.

d. Persistent Data Storage

Because our software will be run on Android, SQLite will be the database used for storage. It implements a self-contained and transactional SQL database engine without the need for any configuration. It is the optimal database for applications

because it is a single cross-platform disk file that stores data onto the phone for simple retrieval of data. SQLite's small code footprint and efficient use of memory and disk space makes it highly advantageous

Classes that primarily interact with the database include SettingsPlanner, GoalsMaker, and EventPlanner. Some cases of interactions between classes and database are changing personal settings through SettingsPlanner, creating user goals through GoalsMaker, and updating user schedule with EventPlanner. Between each exchange of events with the database, the controller will act between the user and the system.

e. Network Protocol

As our project does not require real-time streaming of any kind, our application will be implemented through HTTP connecting from our application to the database server.

f. Global Control Flow

Execution Orderness

This project will be an event-driven system. The user will have complete control over how they use the application. There is no linear procedure for the user to take, and they do not even have to use all of the features that the app provides. The user can use the app's interface to use any function at any time, such as creating goals or events for their schedule. The GPS, however, is persistent upon opening the app, so that walking/exercise time and distance may be tracked.

Time Dependency

This application features timers for many of its functions. One of the application's main functions is that it will track not only how far the user walks, but for how long the user walks for, in real time. It also attempts to track how long the user uses gym facilities for. Based on how long the user exercises for, the app will increase or decrease the user's health index. Because another main component of this application is the schedule, time is also used to track the nearing or passing of events for the user to try to help keep them on track and on time. One last feature that utilizes the clock is the alarm, which is used to track the user's sleep schedule. The application overall has a dependency on time.

Concurrency

This system will run on multiple threads. The main thread will be the user interface, where the user can access and change the app's functionality, and the subthreads will be all of the smaller tasks the app does, the GPS and its associated features, and the networking. All subthreads will have to communicate with the user interface so that the all the information they collect can be viewed by the user, including position, time and distance walked, etc...

g. Hardware Requirements

This software will be run on mobile smartphones with touch screen display and network/WiFi/GPS support. The required operating system will be Android starting from version 5.0, Lollipop. This platform is accessible and the requirements are met through most Android phones.

4. Algorithms and Data Structures

a. Algorithms

Our algorithm will be based primarily off of the user's Health Index. With a default Health Index saved into a database, users can see their progress after it is updated every night. Specific classes are able to access the database to either retrieve, input, or update data. The mathematical model to calculate individual Health Indices incorporates a low-intensity and high-intensity modifier depending on how the user is exercising. For example, an activity like walking would be marked as low-intensity while an activity such as going to the gym would be marked as high-intensity. The intensity of the activity will be determined by the GPS; constant slow movement would be recorded as walking, constant fast movement would be recorded as running, a period in an exercise facility would be recorded as going to the gym. These factors would all contribute to increasing the Health Index while ignoring exercise would slowly degrade the Health Index.

$H_n = L(x_1) + H(x_2) + H_{n-1}$

- H_n = Health Index today
- H_{n-1} = Health Index yesterday
- H_0 = 200 (Default Health Index)
- x_1 = Time in minutes for low intensity activity
- x_2 = Time in minutes for high intensity activity
- L = 1

- H = 3
- If $x_1 < 30$ AND $x_2 = 0$, then 100 is deducted from the total Health Index

The optimization of the user's schedule will be based off the user's event inputs and settings. For example, if t > 40 between the end of event1 and the start of event2, the system may potentially insert an event to prompt the user to eat a meal. The algorithm will analyze the schedule, filter the time slots, and linearly look at the quantified time slot through a series of checks (if-statements) to determine when to notify the user of the time to eat/workout/sleep.

b. Data Structures

The data structure that our system plans to use is the database, more specifically, the SQLite database, which is where all the information pertinent to our system will be stored. Our system will use the following data types in order to accurately monitor the user's active periods and calculate his/her health index:

HealthIndex: double - the user's health index for the current day which will be calculated daily in order to keep progress of the user's performance.

previousHI: double - the user's health index for the previous day which is required to caluculate the user's health index for the current day.

fSchedule: sTable - the user's schedule with Active Period that is synchronized with the database. This table will be used to generate a schedule for the user that shows the chronological breakdown of the user's Active Periods.

TotalLowAPhours: dTable - All of user's low intensity Active Period times according to date in a date table data type which will be synchronized with the database. The amount of time the user spends during low intensity Active Periods is required to calculate the user's health index.

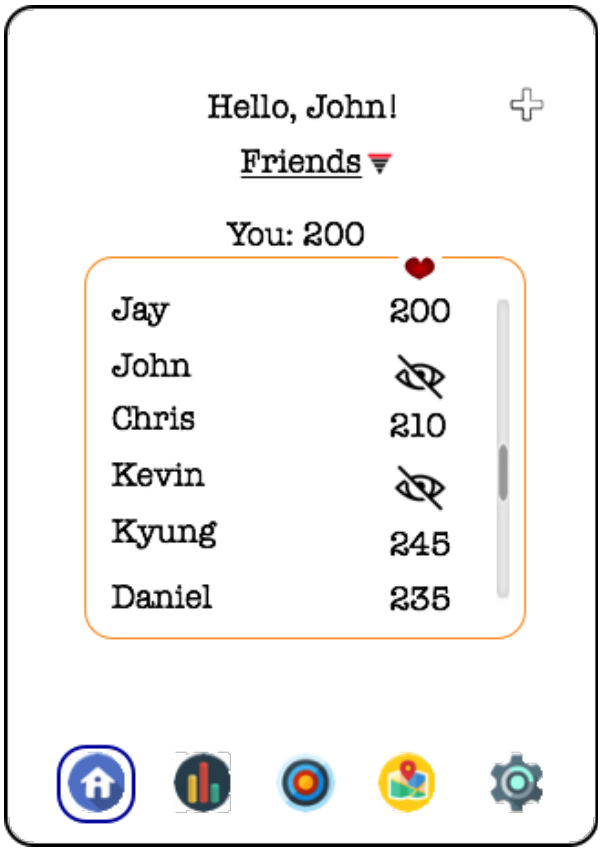
TotalHighAPhours: dTable - All of User's high intensity Active Period times according to date in a date table data type which will be synchronized with the database. The amount of time the user spends during high intensity Active Periods is also required to calculate the user's health index.

The interaction between the database and the two classes, HealthIndexCalc and GraphMaker, will ensure a successful operation of our system. The HealthIndexCalc

class is responsible for generating/calculating and storing the user's health index within the database. This is essential for tracking the user's performance on a daily basis. The GraphMaker class is responsible for creating graphs of the user's Active Periods over a certain amount of time. In addition, this class interacts with the database by storing the user's low and high Active Period times which is essential for calculating the user's health index.

5. User Interface and Design and Implementation

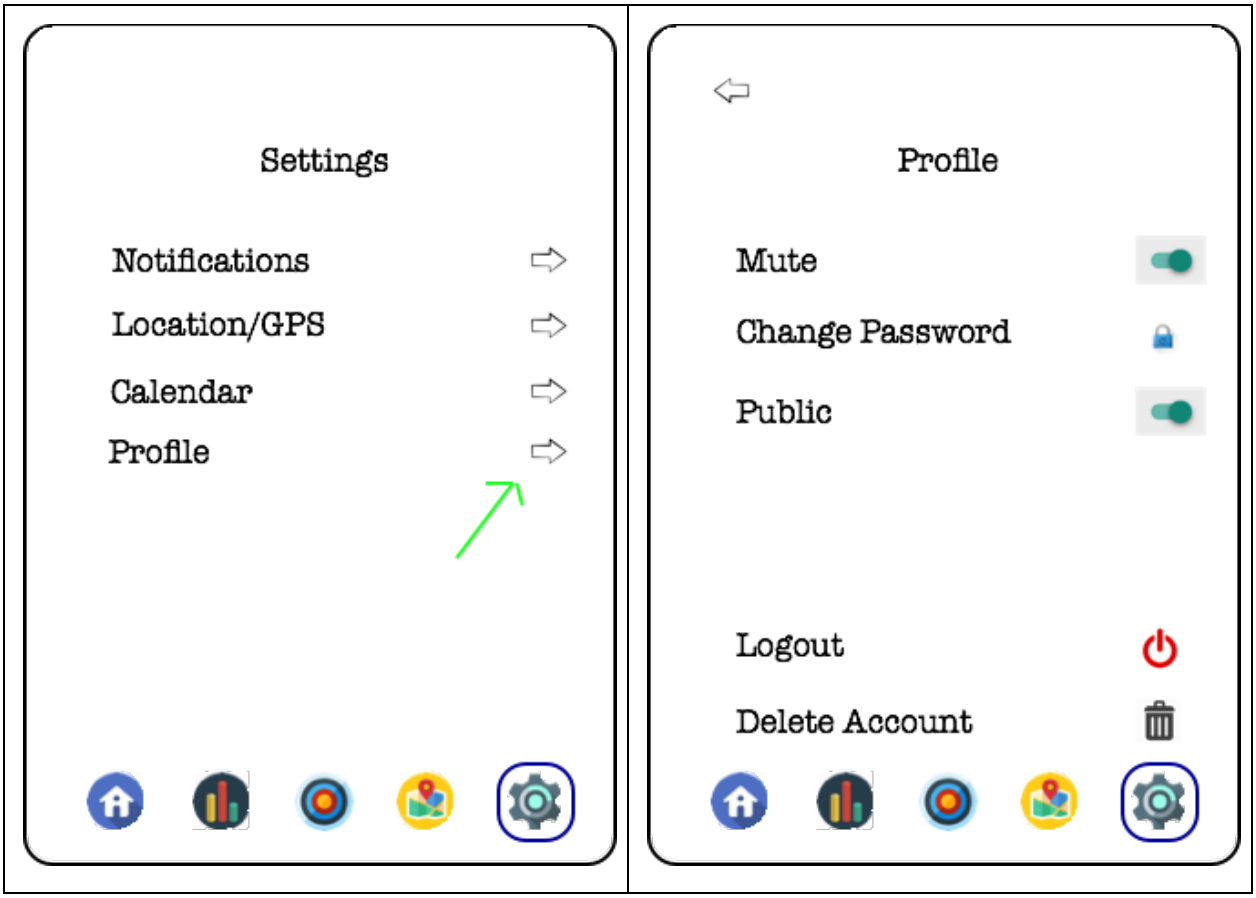
The new changes to the UI are based on overall design changes elaborated in the "Summary of Changes" section. The initial mock-ups were designed to very intuitive or suggestive (having icons to signal an action). The overall design has remained the same with minor changes that simplifies our project. User effort estimation has remained the same.



In the home tab, after tapping on "Schedule" with a new added icon (upside down triangle) indicating that the Schedule could be toggled to view a list of Friends. This mockup also reflects the public/private setting

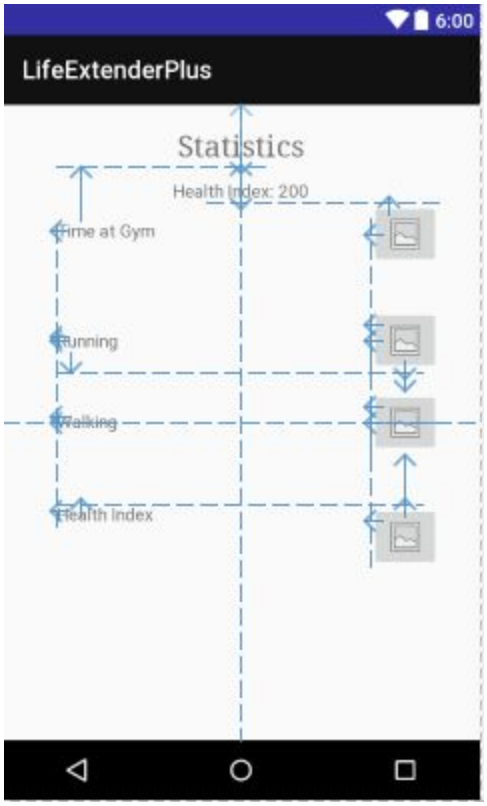
In the statistics tab, it has been update to show only what is being tracked. The sharing function for HI has been changed and is a setting. This is reflects on a minor change to UC-5 (ViewStatistics).

of the health index. The friends without a number have a “cannot see” symbol instead. This is a modification to UC-2 (ShareHealthIndex).	



The public setting to share the health index is now reflected within the settings tab. (Off means private). Again, this is reflects on UC-2 (ShareHealthIndex).

Code implementation is in progress with a very similar setup. Android Studio provides a GUI to design the UI of our app, abstracting away from the application code. It is very similar to the way the mock-ups were created (Paint). Below is the starting design of the Statistics Tab.



6. Design of Tests

a. Test Cases

Add Events in Scheduler
Test Coverage: EventPlanner **Assumption:** The user selects the option to add an event from their schedule **Steps:** 1. User selects Add New Event from the application menu 2. User enters a name for the event and a start and ending time for the event (as a minimum). 3. User submits the event.

Expected: The schedule will now display the event added by the user

Fails if: The user confirms the addition of the new event, but the event does not appear on the schedule

Add Goal

Test Coverage: EventPlanner, GoalsMaker

Assumption: The user has selected to add a new goal through the application's interface

Steps:

1. User access the "Add Goal" tab from the goals page
2. User enters information relevant to their goal
3. User confirms the information they added and submits the goal

Expected: The goals page now displays the goal that the user has added

Fails if: The user confirms the addition of their new goal, but the goal does not appear on the goals tab.

Change Settings

Test Covers: SettingsPlanner

Assumptions: User attempts to change settings in the application to meet their needs and preferences

Steps:

1. User selects the Settings options menu
2. User adjusts one setting.
3. User saves their preferences and closes the settings menu
4. User tests to see if the changes to the settings are reflected in the functionality of the application.
5. User repeats until all settings have been tested.

Expected: App functionality changes in accordance to each setting as it was changed.

Fails if: The user changes a setting, but the change is not reflected in the application.

Notifications
Test Covers: EventPlanner, UI, SettingsPlanner **Assumptions:** User has notifications enabled and an event that they scheduled is approaching. **Steps:** 1. User adds an event to the schedule. 2. When the time for the event arrives, the user's phone should notify the user via push notification. **Expected:** The push notification rings the user's phone and alerts them of the upcoming event. **Fails if:** user is not notified of the event.

Time Recommendations
Test Covers: EventPlanner **Assumptions:** The user has submitted a series of events representing their classes and other responsibilities. The application will then attempt to find times for the user to visit a gym to engage in physical activity with the user's schedule at consideration. **Steps:** 1. User enters a series of events into the calendar. 2. User confirms their schedule and checks their calendar for the application's recommendations. **Expected:** New events are added by the application suggesting times for the user to engage in physical activity, sleep, etc… **Fails if:** No new events or suggestions are added to the schedule or suggestions are added that illegally overlap with existing events.

Getting Health Indexes
Test Covers: UI **Assumption:** User accesses the statistics page of the application. **Steps:**

1. User accesses the statistics tab in the application's interface.
2. Health index is calculated using the model described in the other relevant sections.
3. Health index is displayed for the user

Expected: Health index is displayed

Fails if: Health index fails to display or an erroneous number is generated instead.

Share Health Index

Test Covers: UI

Assumptions: User has added friends and would like to share/view their friend's health indexes through the statistics page.

Steps:
1. User accesses the statistics page and taps on the share icon next to "Health Index"
2. User selects friends who the user wants to allow their health index to display for and taps the confirm/send icon at the top corner.

Expected: Friends that the user has selected should be able to view the user's Health Index

Fails if: Friends are unable to view the user's health index, or are able to view the user's health index when the user disallowed them from viewing it. Also fails if an erroneous figure is displayed

View Statistics

Test Covers: UI

Assumptions: User accessed the statistics page and would like to view their statistics for time/distance walked, sleeping times, health index, or their time spent at the gym.

Steps:
1. User accesses the statistics tab from the application's interface.
2. User selects which statistic they would like to view a graph for.
3. A graph is displayed that communicates to the user the information they are looking for.

Expected: a graph is displayed showing the user the trend of their physical activity over time.

Fails if: The graph fails to display, or the graph displays any erroneous information.

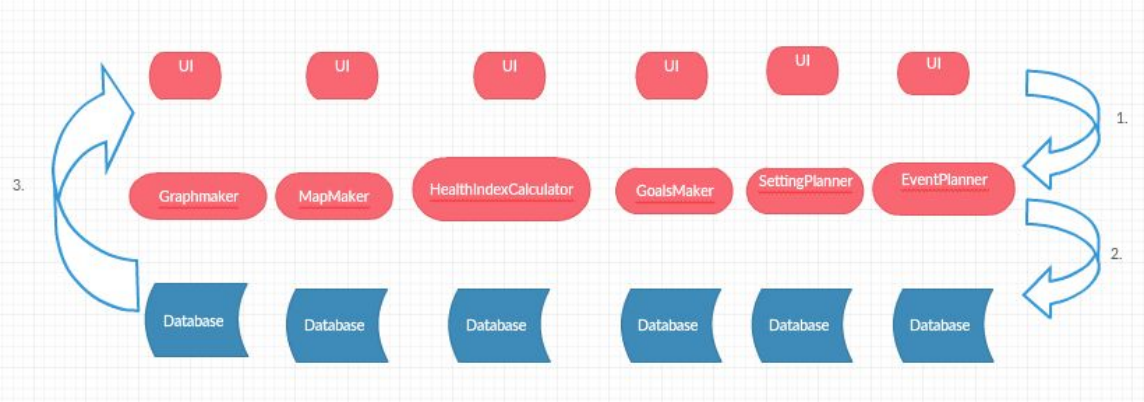
b. Test Coverage

Our test cases serve to ensure that the interactions between the user and the user interface hold up to our standards of how the application should run. Most of our testing revolves around checking to see if user inputs register accordingly and are accurately portrayed to the user in the application's interface. The first area we plan to test is the adding of events and goals to the system. This requires the complete and successful functionality of the EventPlanner and GoalsMaker classes, respectively. Without the function to personalize the app to the user's preference, such as adding his/her own events and goals, this application would be nearly ineffective. Adding on to the importance of personalizing the application to the user's preference, we make sure that the user is able to change any of the available settings in the application such as the general and notification settings which require the cooperation of the UI with the EventPlanner and SettingsPlanner classes.

c. Integration Testing

In order to figure out whether all components of the applications works together, we will go with the vertical integration testing model. This will yield working deliverable code more quickly and since most of the subsystems are working independently of each other (except for the UI and database which are working with all the other classes), it will be easy to make user stories that are parallel to each other.

We will test the appropriate part of UI that corresponding to the class in the middle and see whether the data was transferred to the database correctly. For example,when adding a goal, we will test the add goals button of the Goals Tab of the UI and see whether the goal was added to the database and displayed correctly in the UI.



1.Test the input of the UI 2.See whether correct information was stored into the database
3. See whether the data is outputted correctly in the UI

7. Project Management and Plan of Work

a. Merging the Contributions from Individual Team Members

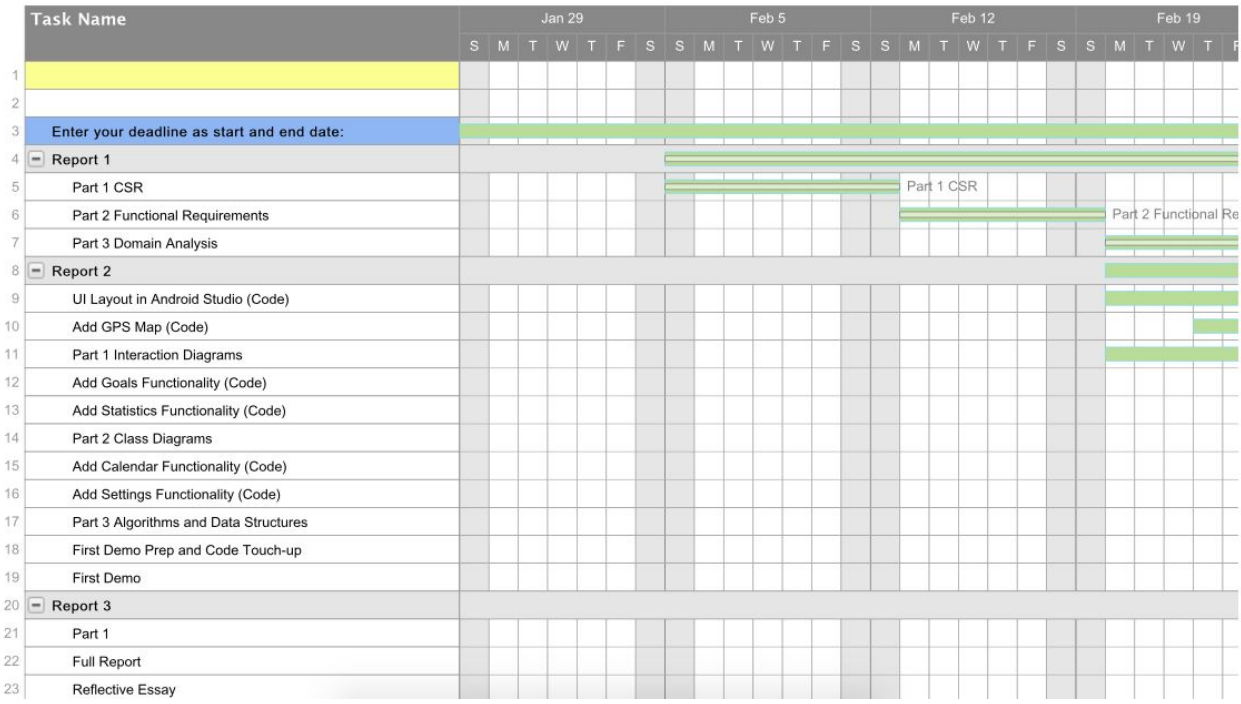
Many problems arose from the beginning as members developed individual ideas of what the software's functions and operations should be. As members started meeting more often, we realized many functions were either overambitious or superfluous. A lot of classes we thought would be necessary turned out to be useless or able to be combined with another class. Each iteration of the design made the application more and more utilitarian as we were able to condense and update the essential use cases. As the software became more practical, it also slowly became more feasible. By cross-referencing each teammate's knowledge and past projects, we were able to find many resources necessary to begin drafting the application.

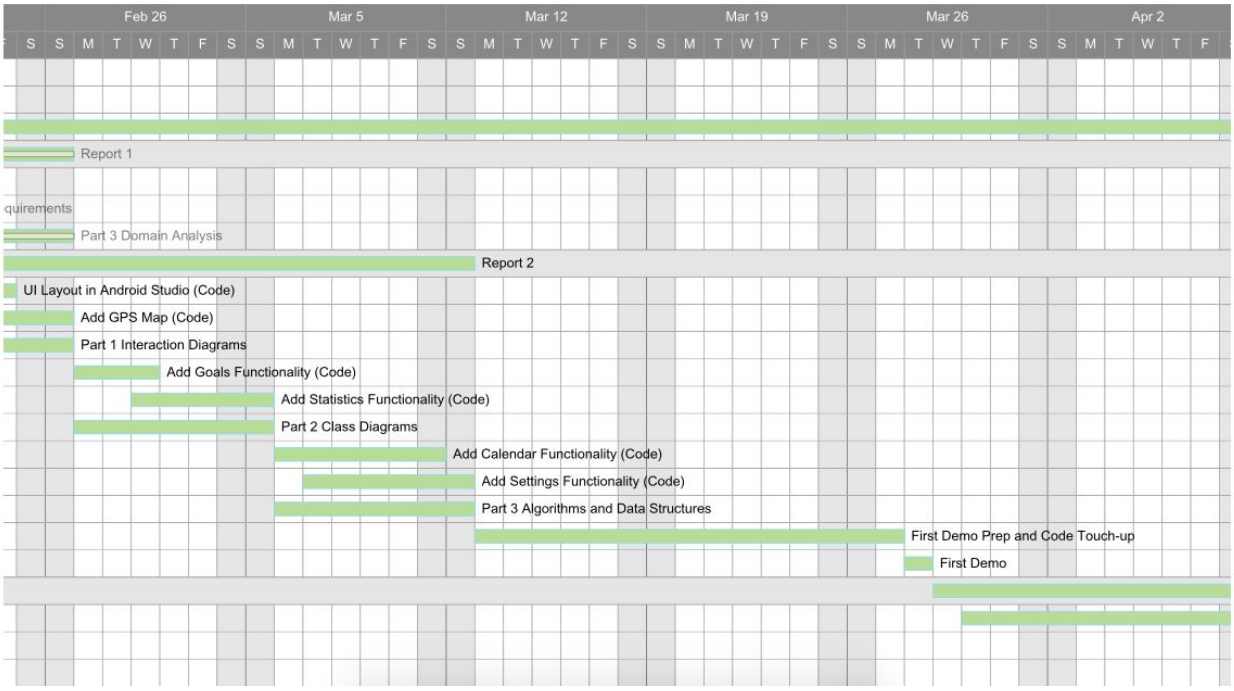
Initial complications for the project included the type of language, platform, database, etc. After team discussions, we concluded that using C, a general purpose language often used in software and software applications, was the optimal language to use because of its efficiency to map constructs to machine instructions. Later on, we will also integrate the database, SQLite, to operate on all data due to its previously described capabilities. Also, we decided to have the application run on Android starting from one of its most commonly used Operating Systems, Lollipop. Consequently, we will design the program on the IDE, Android Studios, for the easiest Android application development

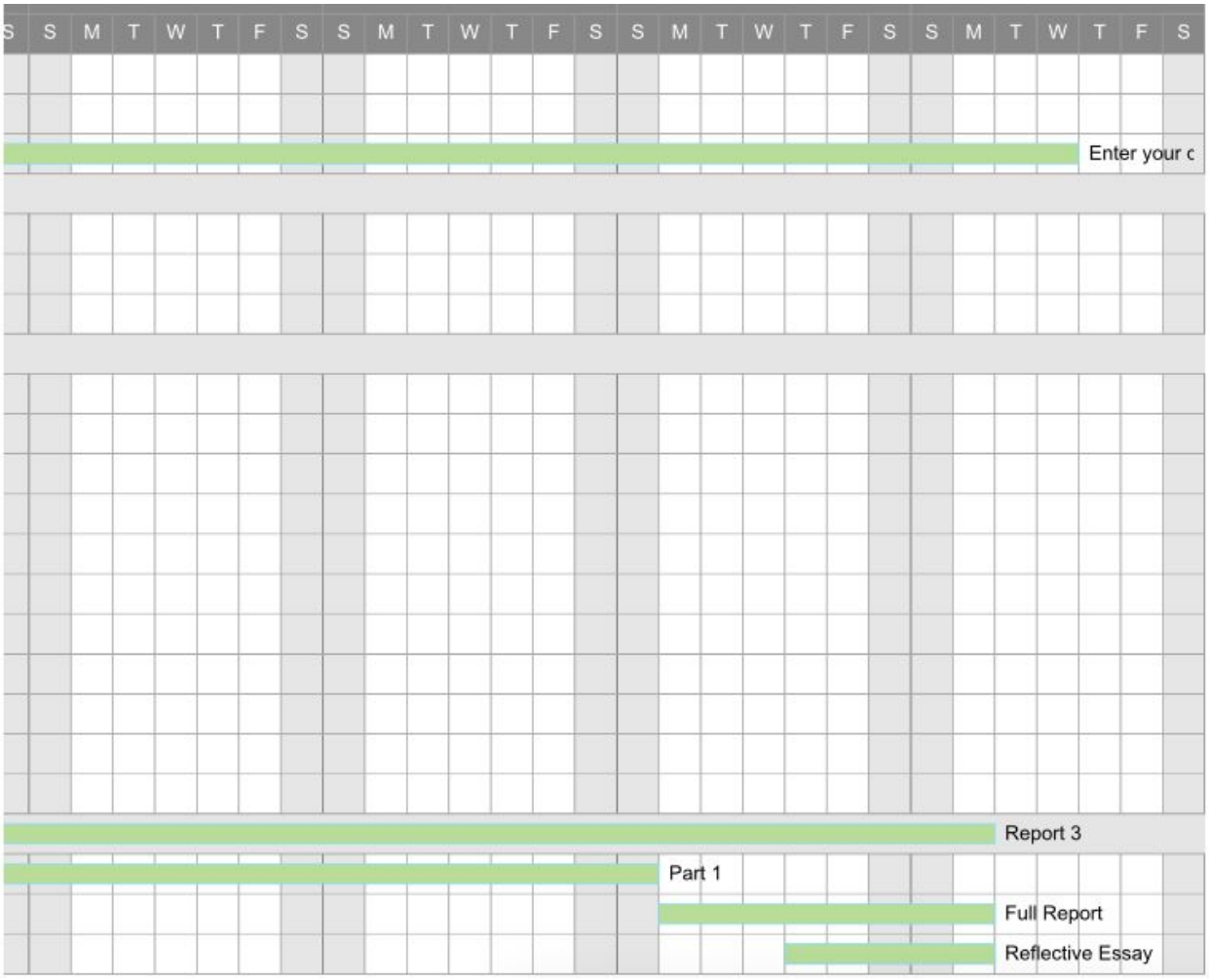
b. Project Coordination and Progress Report

As of now, many of the functions, operations, front-end/back-end logistics, and use cases of the software are concrete but not actualized. We have most of the necessary resources for the software ready to use and implement. Over the period of spring break, members will coordinate with each other to prepare the project demo and begin actual software development for functionality by communicating through GroupMe and Facebook for document sharing, revision, and collation. Furthermore, we will attempt to integrate different parts of the software into one unified system.

c. Plan of Work




Feb 26
Mar 5
Mar 12
Mar 19
Mar 26
Apr 2
S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F S S M T W T F
Report 1
quirements
Part 3 Domain Analysis
Report 2
UI Layout in Android Studio (Code)
Add GPS Map (Code)
Part 1 Interaction Diagrams
Add Goals Functionality (Code)
Add Statistics Functionality (Code)
Part 2 Class Diagrams
Add Calendar Functionality (Code)
Add Settings Functionality (Code)
Part 3 Algorithms and Data Structures
First Demo Prep and Code Touch-up
First Demo



Task Name	Start Date	End Date	Assigned To	Status	Duration	% Complete
Enter your deadline as start and end date:	**01/29/17**	**05/03/17**			**69d**	
Report 1	**02/05/17**	**02/26/17**	**ALL**		**21d**	**100%**
Part 1 CSR	02/05/17	02/12/17	ALL	Complete	7d	100%
Part 2 Functional Requirements	02/13/17	02/19/17	ALL	Complete	7d	100%
Part 3 Domain Analysis	02/20/17	02/26/17	ALL	Complete	7d	100%
Report 2	**02/20/17**	**03/12/17**	**ALL**	**Not Started**	**21d**	**0%**
UI Layout in Android Studio (Code)	02/20/17	02/24/17	John	Not Started	4d	0%
Add GPS Map (Code)	02/23/17	02/26/17	Trirmadura	Not Started	3d	0%
Part 1 Interaction Diagrams	02/20/17	02/26/17	ALL	Not Started	7d	0%
Add Goals Functionality (Code)	02/27/17	03/01/17	Dan	Not Started	3d	0%
Add Statistics Functionality (Code)	03/01/17	03/05/17	Kyung, Chris	Not Started	4d	0%
Part 2 Class Diagrams	02/27/17	03/05/17	ALL	Not Started	7d	0%
Add Calendar Functionality (Code)	03/06/17	03/11/17	Chris, Kevin, Dan	Not Started	5d	0%
Add Settings Functionality (Code)	03/07/17	03/12/17	John, Kyung, Trirmadura	Not Started	4d	0%
Part 3 Algorithms and Data Structures	03/06/17	03/12/17	ALL	Not Started	7d	0%
First Demo Prep and Code Touch-up	03/13/17	03/27/17	ALL	Not Started		0%
First Demo	03/28/17	03/28/17	ALL	Not Started	1d	0%
Report 3	**03/29/17**	**05/01/17**	**ALL**	**Not Started**	**7d**	**0%**
Part 1	03/30/17	04/23/17	ALL	Not Started	7d	0%
Full Report	04/24/17	05/01/17	ALL	Not Started	7d	0%
Reflective Essay	04/27/17	05/01/17	ALL	Not Started	4d	0%

Excel Format of Gantt Diagram with Work Distribution

d. Breakdown of Responsibilities

Responsibilities:
John: UI, MapMaker, Graphmaker
Kyung: Eventplanner, MapMaker, Graphmaker
Dan: SettingsPlanner, HealthIndexCalculator, EventPlanner
Trirmadura: EventPlanner, MapMaker, Graphmaker
Kevin: HealthIndexCalculator, GoalsMaker
Chris: GoalsMaker, HealthIndexCalculator

The person who is responsible of the integration of the classes to form a cohesive product will be John. Since he is mostly in charge of the UI and has the most experience with programming, he would be the best candidate to integrate all the classes together. While each person is responsible for the unit testing for each class, Dan will test the system after it is completely integrated in order to get a more complete test.

8. References

1. Exercise and Psychological Health. 2017. Exercise and Psychological Health. [ONLINE] Available at: https://www.unm.edu/~lkravitz/Article%20folder/exandpsychological2.html [Accessed 04 February 2017].
2. CDC.gov. 2017. How much physical activity do adults need? | Physical Activity | CDC . [ONLINE] Available at: https://www.cdc.gov/physicalactivity/basics/adults/. [Accessed 04 February 2017].
3. How Much Sleep Do We Really Need? 3 - National Sleep Foundation. 2017. How Much Sleep Do We Really Need? 3 - National Sleep Foundation. [ONLINE] Available at: https://sleepfoundation.org/how-sleep-works/how-much-sleep-do-we-really-need/page/0/2. [Accessed 04 February 2017].
4. Consequences of Insufficient Sleep | Healthy Sleep. 2017. Consequences of Insufficient Sleep | Healthy Sleep. [ONLINE] Available at: http://healthysleep.med.harvard.edu/healthy/matters/consequences. [Accessed 04 February 2017].
5. Cherry, Kendra. "6 Key Theories of Motivation." *Verywell*. N.p., 14 June 2016. Web. 04 Feb. 2017. <https://www.verywell.com/theories-of-motivation-2795720>.