Lean Analytics for a Leaner Person

Report #3

Fall 2015, Group #8 – Information & Members

Project Repository @ GitHub – Health Analytics

Project Journal – http://blog.willkara.com

Project Website- http://willkara.com/projects/HealthAnalytics/

Manuel Maldonado – manuel.maldonado@rutgers.edu

Ming Tai Ha – ming.tai.ha@rutgers.edu

Nick Taylor – nitaylor@rci.rutgers.edu

Tongpeng Zhang – tongpeng.zhang@rutgers.edu

William Karavites – willkara@rutgers.edu

1 Table of Contents

Report #3 Contributions

Everyone contributed the same amount for writing Report #3.

2 Summary of Work

Our project didn't evolve much as we went through the semester. The biggest change was in our domain model, mathematical model and algorithms since we needed to generate users for our health model to be valid.

3 Customer Statement of Work

3.1 Problem Statement

Nowadays, more and more people are concerned about their health. It has been a challenge for people to become healthier and yet there exists no simple way to visualize how and what they and their communities are doing. Applications available today concentrate mainly on an individual's health, i.e. counting steps and burnt calories. Why not bring all of these use cases together into a single place? The less you have to travel or remember, the more you can concentrate on learning from the data than actually worrying about the data itself.

People tend to focus mainly on their individual health and not their health with respect to their communities. Since the environment heavily influences the decisions an individual makes, the health of an individual is affected by the health decisions made by the people around. The ability to see the lifestyle trends surrounding an individual empowers the individual to make better decisions about his or her health; being aware of an "unhealthy" surrounding can motivate a person to improve his or her health, while being aware of a "healthy" surrounding can motivate a person to exercise in group for support. Additionally, allowing individuals to track their progress, motivate them and provide them with a comparison with average and/or ideal models of health would push them towards a clear healthy goal.

The dramatic growth of open health-related data sources and social media in recent years has not gone unnoticed in the health sector. Social media sites, such as Twitter and Facebook, are increasingly being used to disseminate information among health professional and patients, and are seen as a source of data for surveillance and research. These sites allow people to track public health concerns or to capture discourse taking place beyond traditional media outlets. Additionally, more and more researchers are exposing (also known as open sourcing) their data so other researchers or institutions can validate their findings or utilize it in novel ways.

If we aspire to make an impact in the health landscape of the United States we should focus our efforts on one of the biggest health problems, cardiovascular health. In 2013 about 1 in 3 deaths in the United States were the result of major cardiovascular problems (http://www.cdc.gov/nchs/fastats/leading-causes-of-death.htm). A variety of institutions (i.e. Johns Hopkins Center to Eliminate Cardiovascular Health Disparities, Cardiovascular Health Research Unit of the University of Washington, The Cardiovascular Institute of New Jersey at Robert Wood Johnson Medical School, etc.) have already started to focus more and more on researching ways to lower the morbidity and mortality rates of these types of problems as they not only present a problem to our society's health but also its ability to finance the health care of its citizens.

Thus, people need a large-scale, community-focused health analytics system to better track and manage their health, to be more aware of their own lifestyle choices, and specifically tailor it to also help them increase and maintain their cardiovascular health. This system should allow users to create accounts and profiles, and input information about their workout sessions, and possibly feeding habits, utilizing a web and mobile application. It should take advantage of open data sources and social media to generate health statistics and build health models. We should present such models to users on easy-to-understand visualizations, as well as allowing them to compare themselves, see who is doing great within their community, track their healthy (or unhealthy) habits, follow and live a healthy life, and by focusing on the individual's (and community's) cardiovascular health the system would also aim at lowering the outrageous deaths counts caused by cardiovascular problems.

3.2 The Vision

Though there are systems where users can post their personal accomplishments to social media and can allow them to view various relevant statistics, the data presented to these users is usually constrained to the user community of that particular system; and though these other systems have given their users a glimpse into the everyday health decisions made by the people in their system’s community, the sample may not necessarily be representative, especially since users of a health-related system may be inclined to be more health-conscientious than the general population. Our proposed health analytics system must include a data-derived model generated from various health-related open source data sets backed by public health research papers, this is done in order to simulate a representative sample of different communities across the United States. The reason for basing our model on statistics gathered from governmental and private public health data repositories such as the CDC, Census Bureau, etc..., is because they have a well-document procedure by which they collect various health information about an individual. This ensures that we have a source of information where its quality can be guaranteed by statisticians and epidemiologists. Social

media sources, however, are unable to provide us with the type of information that we need. While people do post information on their exercise and eating habits, it is unlikely people will tweet their cholesterol, blood pressure, and or other useful health statistics. While mining data from social media sources will yield a high quantity of data, there is no guarantee to the quality and utility of the data we may mine. So, we choose to use statistical data from reputable repositories in order to build our model.

By using data from reputable sources, we can build a model and test the model without having to worry about the quality and utility of our base dataset. This health model will be later enhanced with health-related information shared in social media, such as Twitter, in order to make the simulated population more realistic and representative of real-life communities. Moreover, our model can and should learn by utilizing user provided information entered using our system's interfaces and grow as the user-base of our system increases.

Additionally, our application will predict useful health indicators (such as heart rate, blood pressure levels, cholesterol levels) to users based on, among other things, their current state of fitness. Though some health analytics applications plot various statistics about the user community and predict a possible weight gain/loss trajectory, they do not necessarily make a prediction about the health of an individual, and therefore about the health of the group.

By making predictions about the health our users, they can become more aware of what they could do to maintain or increase their personal health. By utilizing our health models, we can present them with their current health levels, as calculated by our system given certain criteria, they can see how their community is doing as a whole, and we can show them how they stand among their local communities. Moreover, by making these predictions about the health of individuals and storing it in our database, we can also collect aggregate health statistics about populations and allow the download of this collected data, thus reducing the need for a testing/data-gathering process that would otherwise be time-consuming for public-health researchers and invasive to the individual's privacy.

Overtime, our system may be able to build a model for individual users based on their current health situation, this individualized model approach may aid them in:

1. Workout planning
2. Tracking the healthy eating and workout habits
3. Personal Health Statistics
4. Exportation of progress for athletic trainer/doctor

Finally, although the main goal of a system like this is to help users and communities focus on their health, specifically cardiovascular health, and in turn reduce these negative death statistics associated with poor cardiovascular health and empower our users to make better health-related decisions we cannot do this without a sound business model which we could allow us fund this operation. Given the potential this collected and anonymized health data presents, it can be provided to other companies (i.e. insurance, health-related department

stores or pharmacies, etc.), at a price, which could use this information to conduct studies on certain populations and communities of interest by identifying certain health trends among them. But as we stated before, we will help public institutions that are focused on the betterment of cardiovascular health by providing them with this data free of charge.

4 Glossary

- **Server –** A server is both a running instance of some software that is capable of accepting requests from clients and the computer that executes such software. They operate within a client-server architecture, in which "servers" are computer programs running to serve the requests of other programs, the "clients". This may be to share data, information or hardware and software resources. Typical computing servers are database server, file server, mail server, print server, web server, gaming server, and application server. The clients may run on the same computer, but typically connect to the server through a network. In the hardware sense, a computer primarily designed as a server is generally specialized in some way for its task. Sometimes more powerful and reliable than standard desktop computers, they may conversely be simpler and more disposable if clustered in large numbers.(https://en.wikipedia.org/wiki/Server_(computing))

Figure 4-1 – Servers in Rack

- **Website –** Websites have many functions and can be used in various fashions; a website can be a personal website, a commercial website, a government website or a non-profit organization website. Websites can be the work of an individual, a business or other organization, and are typically dedicated to a particular topic or purpose. Any website can contain a hyperlink to any other website, so the distinction between individual sites, as perceived by the user, can be blurred. Especially with modern adaptive web design, you can sometimes even create a website that functions well-enough to be used as the main focal point for your company. You may not even need a native application. (https://en.wikipedia.org/wiki/Website)

Figure 4-2 - Rugter's Website

- **Data –** Data is a set of values of quantitative or quantitative variables; restated, pieces of data are individual pieces of information. Data is measured, collected, and analyzed, whereupon it can be visualized using graphs or images. Data as a general concept refers to the fact that some existing information or knowledge is *represented* or *coded* in some form suitable for better usage or processing. (https://en.wikipedia.org/wiki/Data)

- **Database –** A database is an organized collection of data. It is the collection of schemes, tables, queries, reports, views and other objects. The data is typically organized to model aspects of reality in a way that supports processes requiring information, such as modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies. (https://en.wikipedia.org/wiki/Database). In the context of our system, a database will have a slightly different meaning and multiple responsibilities. These are:
    - Long term backend data storage
    - Faster caching service for live data feeds
    - Local data storage on phone for phone-specific functions and cached data

- **Guest –** Is an anonymous user of our system that does not have an account and thus cannot add or view personal information of himself or herself nor can he view the information of others.

- **User –** The subset of guests that have registered accounts with the system and can utilize the system's features available for registered persons only.

- **Mobile Application –** Software designed to run natively on smartphones and other mobile devices.

Figure 4-3 - Mobile Application Visualization

- **Cardiovascular System –** The circulatory system, also called the cardiovascular system, is an organ system that permits blood to circulate and transport nutrients, oxygen, carbon dioxide, hormones, and blood cells to and from the cells in the body to provide nourishment and help in fighting diseases, stabilize temperature and pH, and maintain homeostasis. The study of the blood flow is called hemodynamics. The study of the properties of the blood flow is called Hemorheology. (https://en.wikipedia.org/wiki/Circulatory_system)

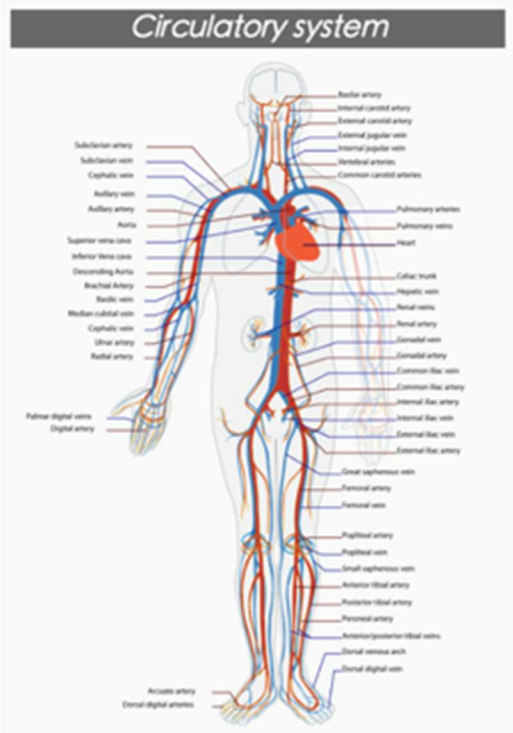


Figure 4-4 - Cardiovascular System

- **Heart Rate –** Heart rate is the speed of the heartbeat measured by the number of contractions of the heart per unit of time — typically beats per minute (bpm). The heart rate can vary according to the body's physical needs, including the need to absorb oxygen and excrete carbon dioxide. Activities that can provoke change include physical exercise, sleep, anxiety, stress, illness, ingesting, and drugs.

- **Blood Pressure –** Blood pressure (BP) is the pressure exerted by circulating blood upon the walls of blood vessels. When used without further specification, "blood pressure" usually refers to the arterial pressure in the systemic circulation. It is usually measured at a person's upper arm. Blood pressure is usually expressed in terms of the systolic (maximum) pressure over diastolic (minimum) pressure and is measured in millimeters of mercury (mm Hg). It is one of the vital signs along with respiratory rate, heart rate, oxygen saturation, and body temperature. Normal resting blood pressure in an adult is approximately 120/80 mm Hg. (https://en.wikipedia.org/wiki/Blood_pressure)

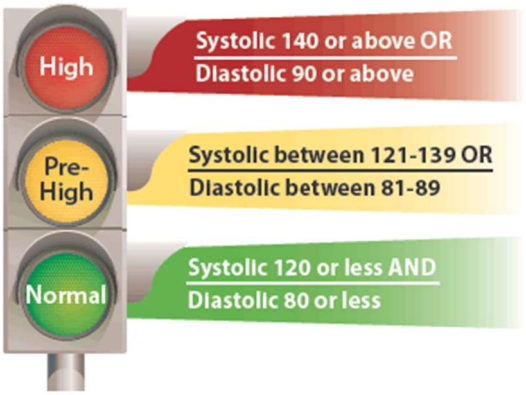


Figure 4-5 - Blood Pressure Levels

- **Cholesterol –** Cholesterol is an organic molecule. It is a sterol (or modified steroid), a lipid molecule and is biosynthesized by all animal cells because it is an essential structural component of all animal (not plant or bacterial) cell membranes that is required to maintain both membrane structural integrity and fluidity. High levels of cholesterol can lead to fatty build up in arteries which in turn can lead to a heart attack or stroke. (https://en.wikipedia.org/wiki/Cholesterol)

- **Data Model –** A data model organizes data elements and standardizes how the data elements relate to one another. Since data elements document real life people, places and things and the events between them, the data model represents reality, for example a house has many windows or a cat has two eyes. Computers are used for the accounting of these real life things and events and therefore the data model is a necessary standard to ensure exact communication between human beings.(https://en.wikipedia.org/wiki/Data_model)

5 Requirements

In the customer statement of work, we have seen what type of system-to-be the customer is expecting. In this section we are going to explore what are the requirements needed to create a system that meets these needs. Here we will not only denote the requirements that would cover the customer's wants but also those that will, behind the scenes, make our system usable and reliable.

5.1 Functional Requirements

Here we will begin by deriving the functional requirements from the statement of work (Section 3 above) which will cover the needs stated by the customer in the problem statement. Given the short time we have to implement such a system, the scope of the system which these requirements denote have been carefully considered and some of these requirements have been marked as "nice-to-have" (denoted by the word "should") and we will try to get to these as time permits.

We will also consider the non-functional on a later section. These types of requirements tend to not be visible in the problem statement but nonetheless are required for our system to do its intended functions. For example, we need to create a health model, to do this we need to gather, study and analyze data, this isn't clearly mentioned in the statement of work but are still an important to the system.

Enumerated requirements for our system-to-be are show below:

Table 5-1 - Enumerated functional requirements for the Health Analytics System.

Identifier	Requirement	Priority
REQ-1	The website should determine a scoring system through supervised learning of known data sets	5
REQ-2	The mobile application shall allow a user to create an account	4
REQ-3	The mobile application shall allow users to log in using their correct account credentials (i.e. username and password)	4
REQ-4	The mobile application shall allow logged-in users to update their account's information (i.e. name, email, age, etc.)	2
REQ-5	The mobile application shall allow users to enter a workout session's information (i.e. type of exercise, duration, intensity, etc.)	5
REQ-6	The mobile application should allow users to view their history of exercise sessions	3

REQ-7	The mobile application should allow users to enter information about food they have eaten (i.e. type of food, time of consumption, etc.)	3
REQ-8	The mobile application should allow users to view their history of food consumption	2
REQ-9	The mobile application shall allow users to view their health index (in comparison to the system's model)	5
REQ-10	The mobile application shall allow users to view information pertinent to their account, workout history and publicly shareable information	4
REQ-11	The mobile application should allow users to link their system's account with their social media account (i.e. Twitter)	2
REQ-12	The mobile application should allow users to share the exercise sessions via social media (i.e. Twitter)	2
REQ-13	The mobile application should allow users to record their exercise session utilizing their device's tracking capabilities and present them with a summary of the session; at a minimum this should work for running and/or jogging exercise types	1
REQ-14	The mobile application should allow users to invite friends from social media (i.e. Twitter) to get the application	1
REQ-15	The website shall display the popularity of different sports according to exercise frequency and the total number of people doing exercise.	2
REQ-16	The website shall display a heat map according to exercise population density of each location from database.	3
REQ-17	The website shall display users heart rate information to remind user about their heart condition.	5
REQ-18	The website shall display the users blood pressure to remind them to keep health.	5
REQ-19	The website shall display the percentage of most popular and useful sports on the location, and user can search what kind of exercises applying to them to keep health.	3
REQ-20	The website shall display a model which is a perfect healthy one making by sources from database , and user can choose useful information through the model.	5
REQ-21	The website shall display a rank list in terms of exercise population density of each state from local database.	3
REQ-22	The website shall allow a guest to create a new account.	5
REQ-23	The website shall allow user log in with correct account credentials (i.e. username and password)	4
REQ-24	The website shall allow user to update their account information.	4
REQ-25	The website should allow user share their exercise information with social media.	3

REQ-26	The website should provide health advice.	4
REQ-27	The website should provide a leaderboard	2
REQ-28	The website shall be able to export user data.	5
REQ-29	The website shall allow user to enter their exercise information.	5
REQ-30	The website should allow users to enter information about food they have eaten (i.e. type of food, time of consumption, etc.)	3

5.2 Nonfunctional Requirements

As stated in Section 5.1 above, we found that there are important non-functional requirements that our system-to-be depends on to provide the desired functionality to our customer. These requirements are listed on Table 5-2 - Enumerated non-functional requirements for the Health Analytics System seen below.

Table 5-2 - Enumerated non-functional requirements for the Health Analytics System

Identifier	Requirement	Priority
REQ-31	The website shall be platform agnostic	5
REQ-32	Internationalized for wider adoption	3
REQ-33	Allow bulk 'import' of data if user uses the app where no cell service is available. User can move the data into the system when service returns.	2
REQ-34	The mobile application should be available for iOS and Android devices	5
REQ-35	The mobile application shall communicate with the system to allow user account creation	4
REQ-36	The mobile application shall communicate with the system to allow user login	4
REQ-37	The mobile application shall submit user-entered exercise session details to the system for storage	5
REQ-38	The mobile application shall retrieve user's profile information from the system	4
REQ-39	The mobile application shall communicate with the system to update user profiles	2
REQ-40	The mobile application shall retrieve the user's health index information from the system	5
REQ-41	The mobile application should retrieve the user's workout session history from the system	3
REQ-42	The mobile application should submit user-entered food consumption details to the system for storage	3

REQ-43	The mobile application should retrieve the user's food consumption history from the system	2
REQ-44	The mobile application should retrieve the user's social media account information from the system	2
REQ-45	The mobile application should communicate (to-from) with the social media system (i.e. Twitter) to link user's account	3
REQ-46	The mobile application should implement the social media system's API (i.e. Twitter's API) to allow users to post their workout session	2
REQ-47	The mobile application should integrate with mobile device's GPS to track user's movement to allow workout session recording	1
REQ-48	The website should be able to maintain function in the event of any changes to Twitter's API.	3
REQ-49	The website shall allow only user to log in with correct account and password.	5
REQ-50	The website shall give a security questions to user of they forget their password. If the user give a correct answer they are able to change their password.	5
REQ-51	The website shall have a friendly UI. Date should be presented clear and tidy so that user would like to use our system.	4
REQ-52	The users information will be mainly stored inside the server.	5
REQ-53	A user will only be able to change their own data.	5
REQ-54	Model shall adapt to user input	5
REQ-55	Model should predict health status	4
REQ-56	Model should provide general health stats	4

6 Functional Requirements Specification

In this section we will be exploring in-depth the functional requirements. First, we will be presenting the individuals or organization with specific interest in our system-to-be. We will also show the specific individuals or groups that will directly interact with our system. Finally, we will be exploring the use cases of our system as well as mapping them to our requirements. Doing this exploration and mapping and is important since it will give us a better understanding of our overall system-to-be, establish the scope the work our team will be doing and give us a way to focus on the specific use cases we should tackle first.

6.1 Stakeholders

6.1.1 Users

Most of users are ordinary users who don't have their own specific health goals. Our system-to-be should allow these users to easily see and track health plans and see information that other users of our system have shared.

6.1.2 Fitness Enthusiasts

A subset of the users, fitness enthusiasts have a chance to be leaders in their communities' top rank lists and depending on the data and information these users provide, we can modify our health models, which is a key function in our product, that user can compare and track their health according to these models.

6.1.3 Health Companies

Meanwhile, our system-to-be would also attract some businesses, such as fitness companies, sports products companies, and health insurance companies. When these kinds of companies use our system, they will get health trends and health conditions of people who use our system, and base on this useful information, companies can improve their products to apply to customers, and make a better marketing strategy. Any and all information shared with these companies should be anonymized in order to protect our users' identities.

6.1.4 Government Official

Due to the fact that our product contains a lot of useful information regarding communities and their aggregate health conditions. Government officials can use our data to track these conditions, as they make specific policy changes, to encourage people to keep healthy, and quantify their success.

6.1.5 Health Researchers

As our system grows in popularity, health researchers will want to utilize our data for health trends and data analytics experiments. Our system will allow these types of users to select and download anonymized datasets specifically targeting health research.

6.2 Actors & Goals

6.2.1 Users – Initiating

The user's goal is to login in the system to track the exercise records and find what other people do in the same location.

6.2.2 Guest – Initiating

A guest's goal would be to either view the community data or create a new account with the system. A guest will have very minimal access and available functionality due to their unauthenticated nature.

6.2.3 Database – Participating

The database stores users' account information, activity information, and store extracted information from Twitter by searching related Tweets through keywords. It also stores the details needed to re-create our health model.

6.2.4 Social Media Platform: Twitter – Participating

Twitter shall be our social media platform and should offer all the social information about health and exercise-related activities which we want. It should also offer the ability to link a user's account and allow the system to share user's activities on their behalf.

6.2.5 Data Model – Participating

The data model shall present a standardized healthy person and shall allow comparisons between users and the model. The model should also update itself as more information is available, either real-time or by an administrator command.

6.2.6 External Companies and Researchers – Initiating

A researcher's goal would be, partly, similar to a guest's goal which is to view trends in different communities. In addition, researchers would like to download the anonymized datasets for their research needs.

6.3 Use Cases

Here are the lists of use cases for our system-to-be. Some specifications also list any potential "special case" that the system could or would do. In our initial design, we will not go into an in-depth exceptional user case design given the time we have to implement the system. Needless to say, this means we are assuming the risk in certain areas of our system.

6.3.1 Casual Description

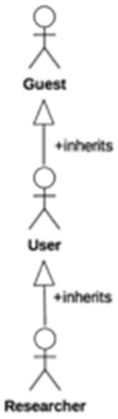
Table 6-1 - Use Cases Description

ID	Name	Description	Requirements
UC-1	Sign up	Allow the guest to create a new user profile.	REQ-2, REQ-22, REQ-35
UC-2	Log in	Allow the user to access the system.	REQ-3, REQ-23, REQ-36, REQ-49, REQ-50
UC-3	Link social media	Allow the user to link their account with their social media account.	REQ-48, REQ-45, REQ-11
UC-4	Update information	Allow user to update their user profile.	REQ-4, REQ-24, REQ-38, REQ-39, REQ-52, REQ-53
UC-5	Add activity	Allow user to enter (or record) an activity (workout or food consumption).	REQ-5, REQ-7, REQ-12, REQ-13, REQ-25, REQ-29, REQ-30, REQ-37, REQ-42, REQ-46, REQ-47, REQ-48
UC-6	Show activity history	Allow user to view his or her past activities.	REQ-8, REQ-9, REQ-10, REQ-41, REQ-43
UC-7	View health information	Allow user to view their health information, health index and recommendations.	REQ-6, REQ-7, REQ-17, REQ-18, REQ-19, REQ-20, REQ-21, REQ-38, REQ-40
UC-8	View community information	Allow user or guest to view community-related information and visualizations.	REQ-6, REQ-7, REQ-8, REQ-11, REQ-15, REQ-16, REQ-44
UC-9	Exporting Information	Allow user to export their information in a human readable format.	REQ-6, REQ-7, REQ-15, REQ-29, REQ-43, REQ-52
UC-10	Predicting Health Status	The app will predict the health status of the user based up the data input from the user	REQ-1, REQ-5, REQ-55

6.3.2 Diagrams

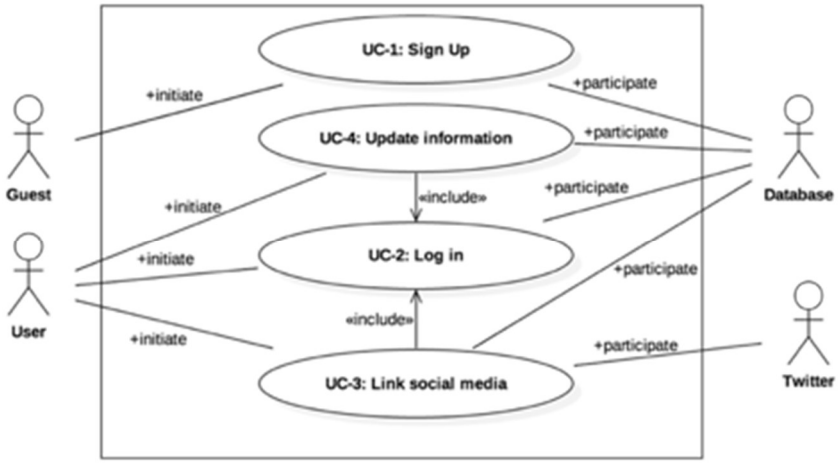
6.3.2.1 Generalizations:

Here we present the hierarchy of actors, specifically how guests, users and researchers are related. The figure below shows that guests are at the top of the hierarchy, all users inherit from the guest while adding a few extra features (like being able to log in and add activities). Finally, researchers are a special type of user where they have the ability to download anonymized data from our system.



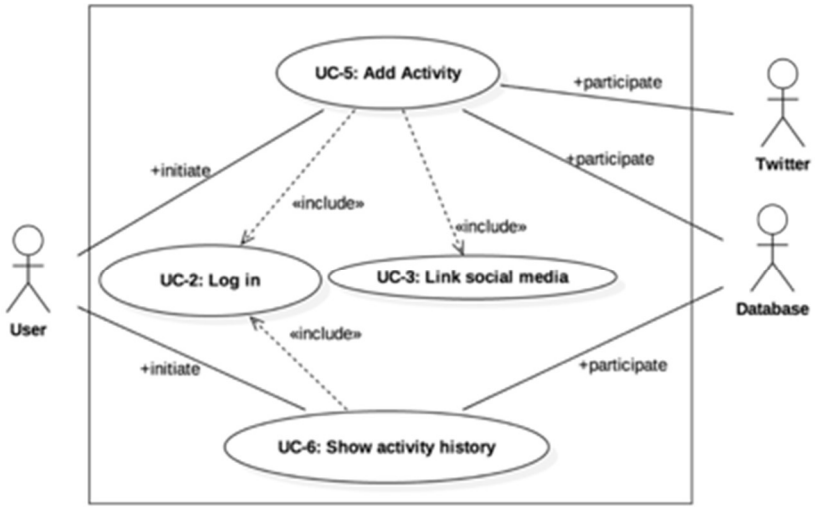
6.3.2.2 User account creation, management and log in

In this diagram we see how the guest initiates account creation, and how a user initiate logging into the system, update account information and link account with social media (Twitter). Also, we also see that for a user to initiate update information and link social media it needs to log in first. Finally, we see that Twitter participates in the link social media use case while the database participates in all of the use cases.



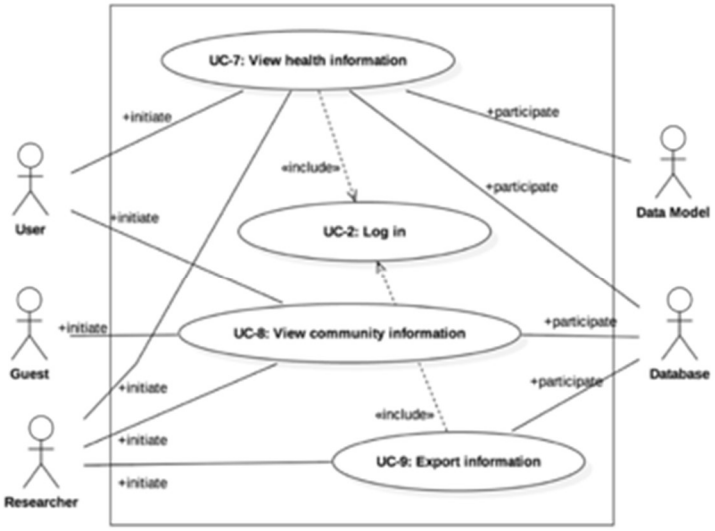
6.3.2.3 User activity viewing and recording

In the diagram below we can see that the user actor is the only one allowed to initiate the add activity and view activity history use cases and both of these depend on the log in use case. Also, we see that the user can share any given activity with their social media account given that they have linked their account with social media before. Finally, we see that Twitter participates during the add activity use case while the database participates in all of the use cases.



6.3.2.4 View health-related information

In the diagram below, we see the actors that can initiate the different health data viewing use cases. Guests are only allowed to view community data while users can view both personal and community. Also, since the researcher actor inherits from the user actor, it can do all the user can plus it can also download aggregated and anonymized health data. Finally, the data model participates during the view health information use case while the database participates in all of the use cases.



6.3.3 Traceability Matrix

REQ ID	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10
REQ-1	5										X
REQ-2	4	X									
REQ-3	4		X								
REQ-4	2				X						
REQ-5	5					X					X
REQ-6	3							X	X	X	
REQ-7	3					X		X	X	X	
REQ-8	2						X		X		
REQ-9	5						X				
REQ-10	4						X				
REQ-11	2			X					X		
REQ-12	2					X					
REQ-13	1					X					
REQ-14	1										
REQ-15	2								X	X	
REQ-16	3								X		
REQ-17	5							X			
REQ-18	5							X			
REQ-19	3							X			
REQ-20	5							X			
REQ-21	3							X			

REQ-22	5	X									
REQ-23	4		X								
REQ-24	4				X						
REQ-25	3					X					
REQ-26	4										
REQ-27	2										
REQ-28	5										
REQ-29	5					X				X	
REQ-30	3					X					
REQ-31	5										
REQ-32	3										
REQ-33	2										
REQ-34	5										
REQ-35	4	X									
REQ-36	4		X								
REQ-37	5					X					
REQ-38	4				X			X			
REQ-39	2				X						
REQ-40	5							X			
REQ-41	3						X				
REQ-42	3					X					
REQ-43	2						X			X	
REQ-44	2								X		
REQ-45	3			X							
REQ-46	2					X					
REQ-47	1					X					
REQ-48	3			X		X					
REQ-49	5		X								
REQ-50	5		X								
REQ-51	4										
REQ-52	5				X					X	
REQ-53	5				X						
REQ-54	5										
REQ-55	4										X
REQ-56	4										
Max PW		5	5	3	5	5	5	5	3	5	5
Total PW		13	22	8	22	36	16	36	15	20	14

6.3.4 Full Description

In this section, we will detail each use case and steps needed to accomplish them. A few things need to be explained prior to jumping into the details.

1. When we use the term “system” or “application” we typically mean the interface the user is utilizing (unless stated otherwise) which could be either the mobile application or the website; since these two have similar functionality we can use them interchangeably throughout this section
2. We assume that if an actor (user or otherwise) initiates a use case via one interface (website or mobile application) the actor will continue each step of the use case using the same interface, meaning that the actor cannot switch from the mobile application to the website (or vice versa) mid-use case
3. Since we are dealing with different social media accounts, we assume the user remembers their credentials to those accounts.

6.3.4.1 UC-1: Sign Up

Initiating actor: Guest.

Actor's goal: To utilize the website or mobile application to create a new user account.

Participating Actor: Database.

Pre-conditions: The guest should browse to the website or download the mobile application, we will refer to either interface as the system. The system shows a “Sign up” link or button to allow guests to create a new user account.

Post-conditions: The guest is now a user with an account. The account will be stored in the database. The guest will now be able to log into the system.

Flow of events for main success step:

1. Guest clicks the "Sign up" button
2. System will present the guest with a form allowing the user to enter their information
3. Guest will enter their name, birth date, gender, height, current weight, email, password, etc.
4. Guest will click the “Create account” button
5. System will (a) validate the information for miss-information
6. System will store the information in the database
7. System will present the guest, now a user, with a congratulatory page explaining that a new account has been created and that the user can now log in using their selected email and password.

Flow of Events for Extensions (Alternate Scenarios):

1a. Guest an email that belongs to another account:

1. System will display an error message to the guest explaining that the email entered is already in use.

6.3.4.2 UC-2: Log In

Initiating actor: User.

Actor's goal: Log into the system to be able to utilize its features.

Participating Actor: Database.

Pre-conditions: The user should already have an account created and should have valid account credentials at hand. The system should show a “Log in” button.

Post-conditions: The user is now logged in.

Flow of events for main success step:

1. User clicks the "Log in" button
2. System will present the user with email and password fields for the user to fill with correct information
3. User enters correct account credentials
4. User clicks the “Log in” button
5. System (a) validates the user’s credential then (b) shows the name of the logged in user in a portion of the system’s interface (a corner of the website or mobile application) and also present the “Log out” button should the user desire to log out.

Flow of Events for Extensions (Alternate Scenarios):

2a. User entered wrong credentials:

1. System will display an error message and allow the user to re-try.

2b. User entered wrong credentials more than the allotted tries:

1. System will display an error message to the user and will not allow that account to be logged in again for 24 hours.

6.3.4.3 UC-3: Link Social Media

Initiating actor: User

Actor's goal: To link the user’s different social media accounts to their application account. This allows the ability to post information to and use information from the different accounts

Participating Actor: Database, User

Pre-conditions: Show a list of different social media platforms with the ability to 'connect' the accounts.

Post-conditions: User will now have the ability to post information to their social media platforms through the app and use social information from the different accounts.

Flow of events for main success step:

2. User clicks "Social Media" button;
3. User is presented with a collection of different social media platforms to link to.
4. User clicks the social media account they wish to link and enters in their credentials.
5. User's account will now have functional access to the social media accounts.

6.3.4.4 UC-4: Update Profile

Initiating actor: User.

Actor's goal: Update his or her information in the system.

Participating Actor: Database.

Pre-conditions: The user should already be logged in. The system should show an “Update Profile” button.

Post-conditions: The user’s profile is updated with the newly provided information and stored in the database.

Flow of events for main success step:

1. User clicks the "Update profile" button
2. System will present the user with the current values for each item in the profile
3. User enters corrects any item in the profile as needed
4. User clicks the “Save” button
5. System (a) validates the user’s newly entered information then (b) displays a success message telling the user the changes to his or her profile have been saved

Flow of Events for Extensions (Alternate Scenarios):

4a. Guest an email that belongs to another account:

1. System will display an error message to the guest explaining that the email entered is already in use.

6.3.4.5 UC-5: Add Activity

Initiating actor: User.

Actor's goal: Add an activity (either food eaten or workout done) to their account.

Participating Actor: Database, Twitter.

Pre-conditions: The user should already be logged in. The system should show the “Add new Workout” or “Add new food” options. For sharing in social media, the user should have already linked their account with Twitter. In the mobile application, the button “Record a workout” will be shown as well.

Post-conditions: The user’s account will have a new activity stored and can be seen in the activity history.

Flow of events for main success step:

1. User clicks the (a) "Add new workout" or “Add new food” button
2. System will present the user with fields the user needs to enter in order to record the new activity, for food, it will display type of food, amount, calories, etc., and for workout it will display type of workout, time, distance, etc.
3. User enters the information pertinent to the activity
4. User clicks the “Submit” button
5. System will store the new activity’s information in the database.
6. System will show the user a (d) “Share on Twitter” button
7. Mobile application sends a formatted message to Twitter on behalf of the user containing the details of the activity

Flow of Events for Extensions (Alternate Scenarios):

5a. The user is using the mobile application and would like to record an activity:

1. User clicks on the “Record a workout” button
2. Mobile application shows several workout options for the user to choose
3. User chooses an option (b)
4. Mobile application will display a button that reads “Start workout”
5. User clicks on the “Start workout” button
6. Mobile application now displays a “Pause workout” button as well as showing the elapsed time, distance ran, average speed and average calories burned
7. User clicks on the “Pause workout” button
8. Mobile application will stop the timer and stop recording statistics as well as showing two buttons “Resume workout” and “Finish Workout”
9. User clicks on (c) “Finish Workout”

10. Mobile application displays a congratulatory message as well as showing statistics on the workout session that was recorded, it will also send this information to the database for recording
11. Mobile application will show the user a (d) “Share on Twitter” button
12. Mobile application sends a formatted message to Twitter on behalf of the user containing the details of the activity

5b. The user is using the mobile application, would like to record an activity but the activity selected cannot be tracked using GPS (for example: stationary bike)

1. Mobile application will follow 1-5 in **5a**
2. Mobile application now displays a “Pause workout” button as well as showing the elapsed time.
3. User clicks on the “Pause workout” button
4. Mobile application will stop the timer as well as showing two buttons “Resume workout” and “Finish Workout”
5. User clicks on (c) “Finish Workout”
6. Mobile application displays a congratulatory message as well as asking the user for additional information about the recorded workout like “distance traveled” and calories burned
7. User enters this information
8. System will calculate statistics based on the information recorded and provided by the user and display them to the user as well as send them to the database for storage
9. Mobile application will show the user a (d) “Share on Twitter” button
10. User clicks the “Share on Twitter” button
11. Mobile application sends a formatted message to Twitter on behalf of the user containing the details of the activity

5c. The user is using the mobile application; it is recording an activity pauses it but would like to continue instead of finishing

1. User clicks on “Resume workout”
2. Mobile application will continue the workout session as before

5d. The user decides not to share the activity

1. User does not click on the “Share on Twitter” button
2. System will not send a message to Twitter and the activity information will not be shared

6.3.4.5.1 UC-6: Show Activity History

Initiating actor: User.

Actor's goal: View the history of entered or recorded activities.

Participating Actor: Database.

Pre-conditions: The user should already be logged in. The system has a “Show history” button

Post-conditions: The user will see a list containing the history of activities

Flow of events for main success step:

1. User clicks the “Show history” button
2. System (a) displays a list of the user’s activities with buttons that filter by type (food consumption or workout)

Flow of Events for Extensions (Alternate Scenarios):

6a. The user has not entered or recorded any activity yet:

1. The system will display a message stating that the user has not entered any activity.

6.3.4.6 UC-7: View health information

Initiating actor: User

Actor's goal: To allow the website or mobile application users to view their health information, health index and recommendations.

Participating Actor: Database

Pre-conditions: Our website or mobile application shows the "View" button that users click it can view their health information.

Post-conditions: Our website or mobile application displays the user's health information, health index and recommendations stored in database.

Flow of events for main success step:

1. User clicks "View" button;
2. Our website or mobile application show buttons of “Health Information", "Health Index" and "Recommendations", basically his or her account name;
3. User clicks any of above three buttons to check the information they want;
4. Our website or mobile application displays the information that user chose before.

6.3.4.7 UC-8: View community information

Initiating Actor: User and Guest

Actor's goal: Our website or mobile application allow user or guest to view community-related information and visualizations.

Participating Actor: Database

Pre-conditions: Our website or mobile application has stored all the data and done analysis about community.

Post-conditions: Our website or mobile application displays the community-related information and visualizations.

Flow of events for main success step:

1. User or guest clicks" Show Community" button.
2. Our website or mobile application displays the community-related information and visualizations.

6.3.4.8 UC-9: Exporting Information

Initiating Actor: User

Actor's goal: Our website or mobile application allow user to export their information in a human readable format.

Participating Actor: Database

Pre-conditions: Our website or mobile application has collected the user's health information, analyzed information and stored the results to database.

Post-conditions: Our website or mobile application displays their information in a human readable format.

Flow of events for main success step:

1. User navigates to the Report screen and clicks " Export " button.
2. Our website or mobile application gathers data for database, analyzes data. After above process, system displays the result in a human readable format in the user's interface.

6.3.4.9 UC-10: Predicting Health Status

Initiating actor: Data Model.

Actor's goal: To predict the user's health status based upon new data provided by the user.

Participating Actor: Database.

Pre-conditions: The data model determines the health status of the user based upon a learning algorithm and data previously provided by the user.

Post-conditions: The user's health status will be re-evaluated based upon new data provided by the user

Flow of events for main success step:

1. User inputs new data (exercise activity, weight, blood pressure, cholesterol).
2. Data model calculates the user's current health status.

6.3.5 System Sequence Diagrams

6.3.5.1 UC-1: Sign up

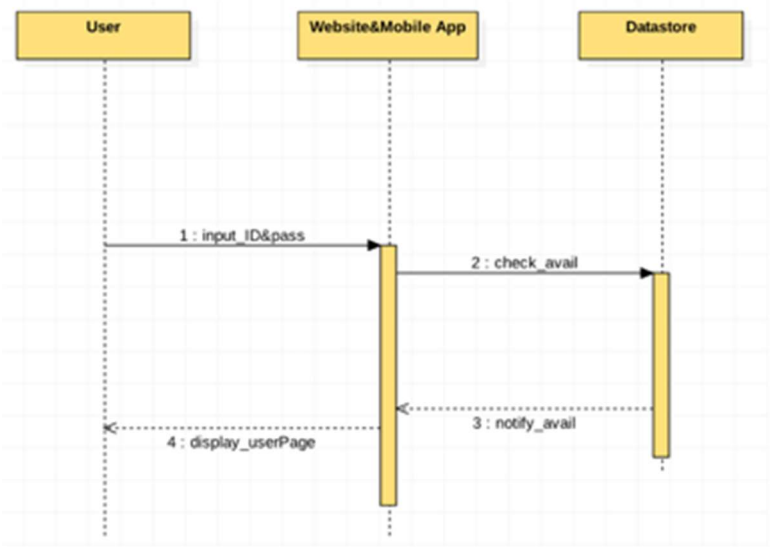


Figure 6-1 - Sign up

6.3.5.2 UC-2: Log In

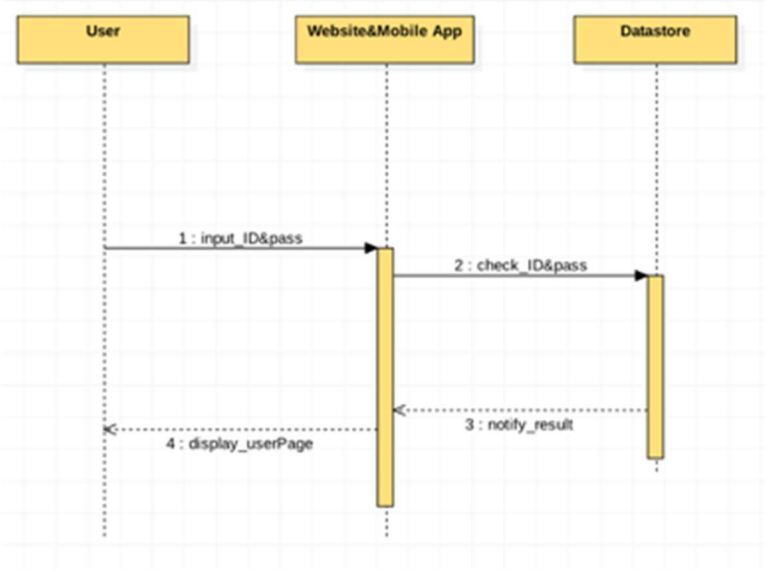


Figure 6-2 - Log in

6.3.5.3 UC-3: Link Social Media

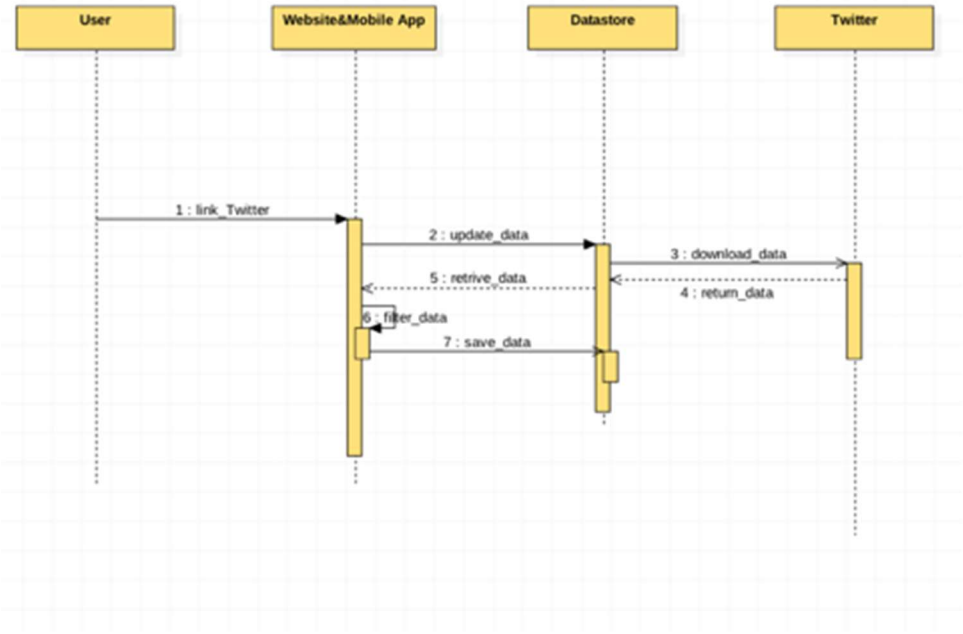


Figure 6-3 - Link to Social Media

6.3.5.4 UC-4: Update Profile

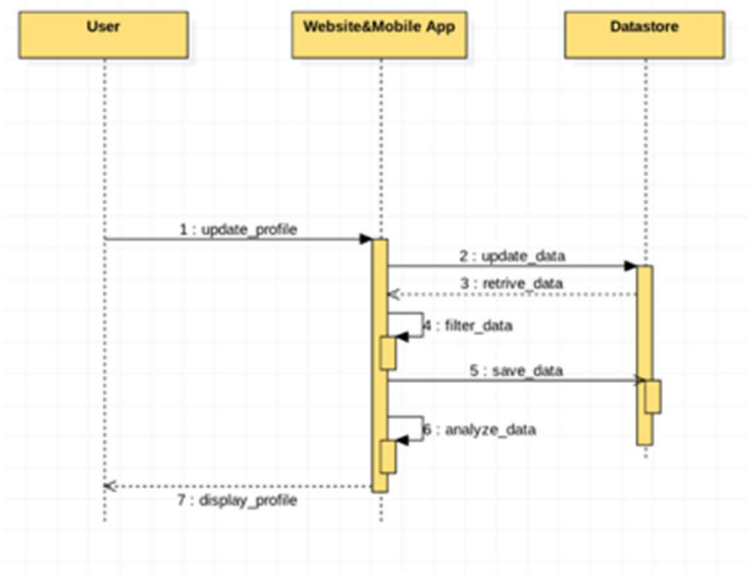


Figure 6-4 - Update Profile

UC-5: Add Activity

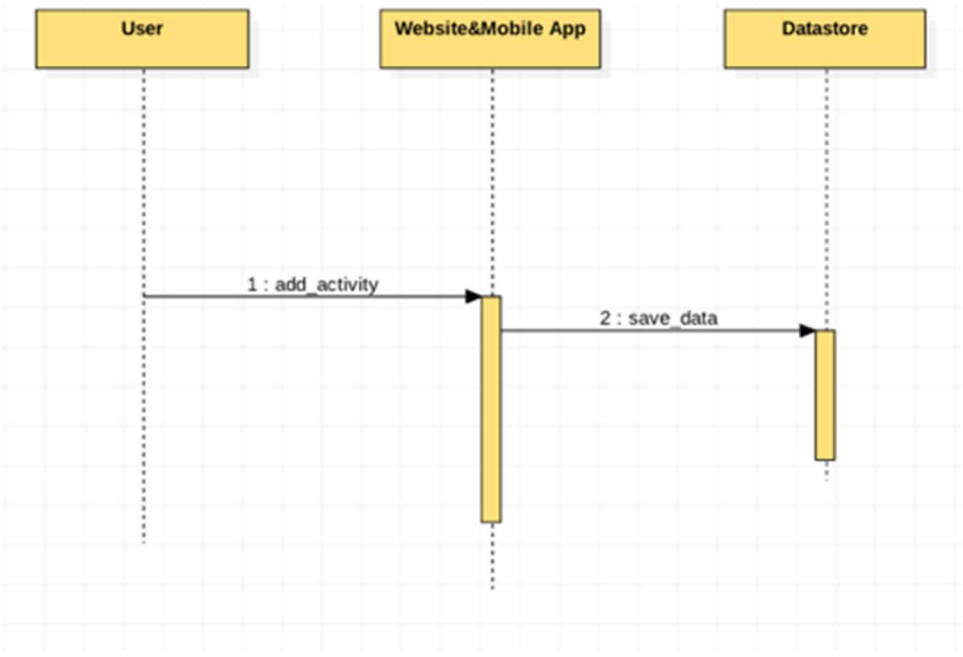


Figure 6-5 - Add activity

6.3.5.5 UC-6: Show Activity History

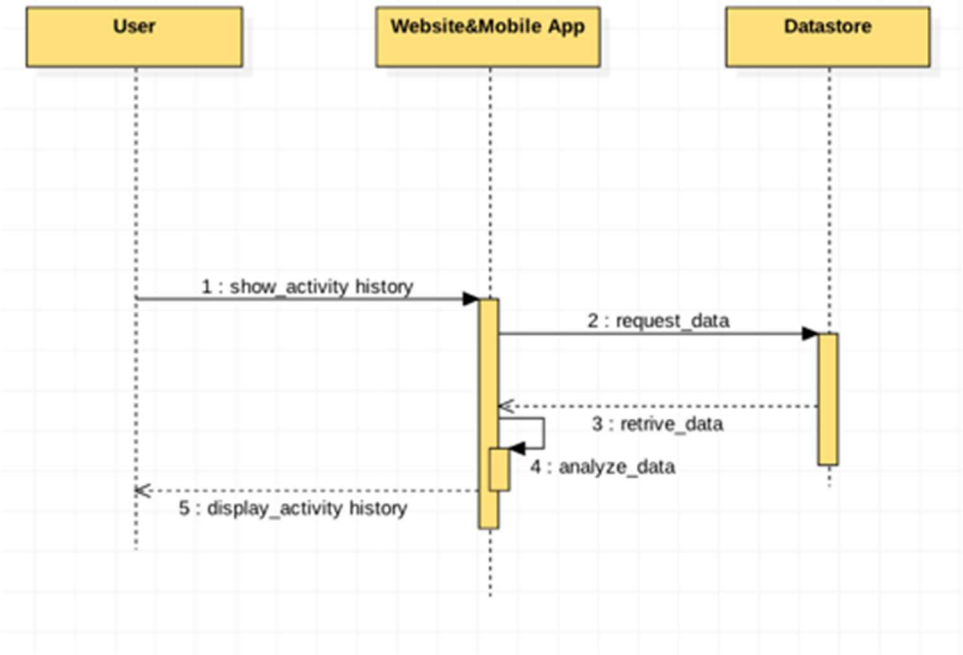


Figure 6-6 - Show activity history

6.3.5.6 UC-7: View Health Information

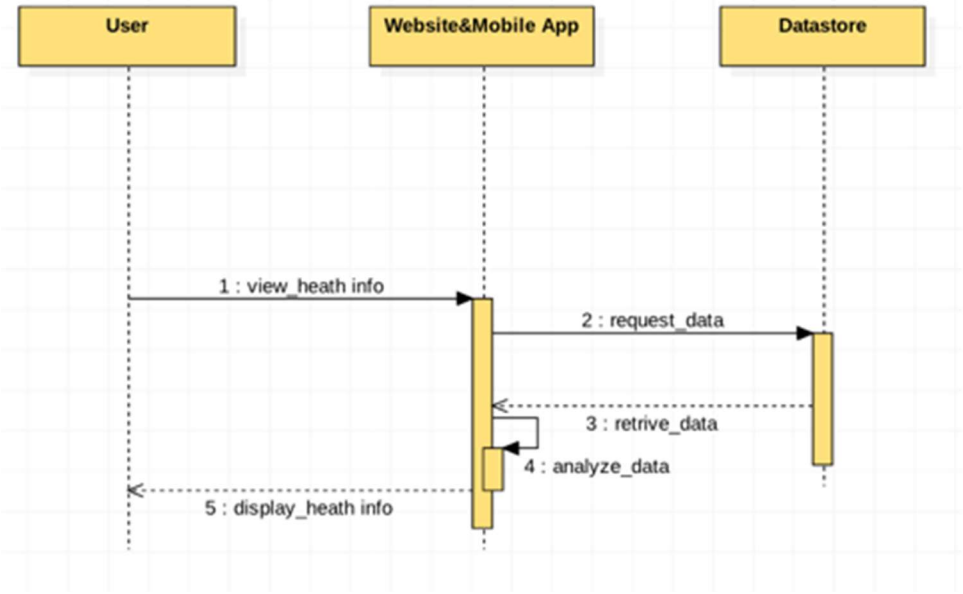


Figure 6-7 - View health information

6.3.5.7 UC-8: View Community Information

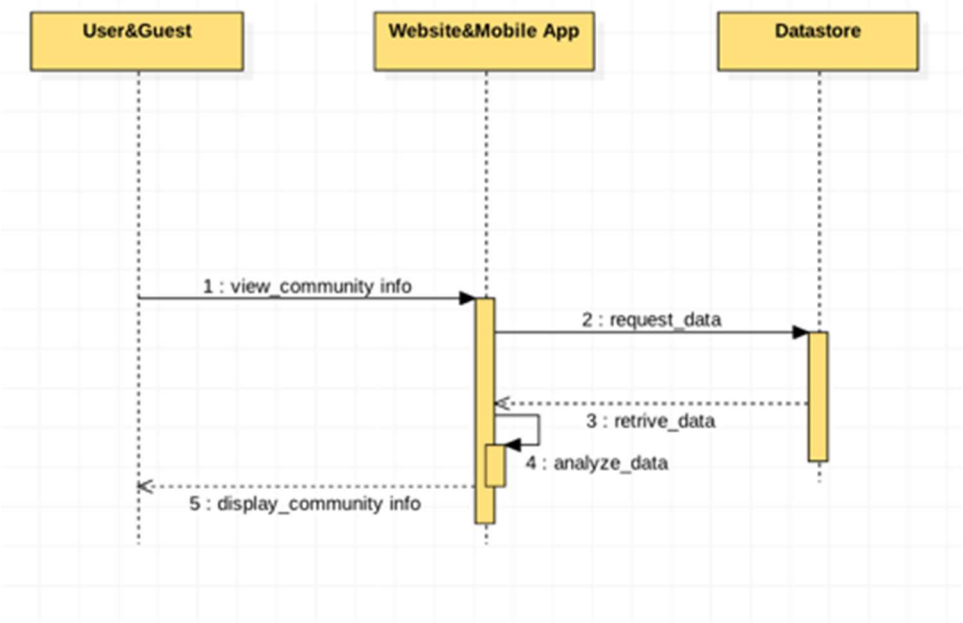


Figure 6-8 - View community information

6.3.5.8 UC-9: Export Information

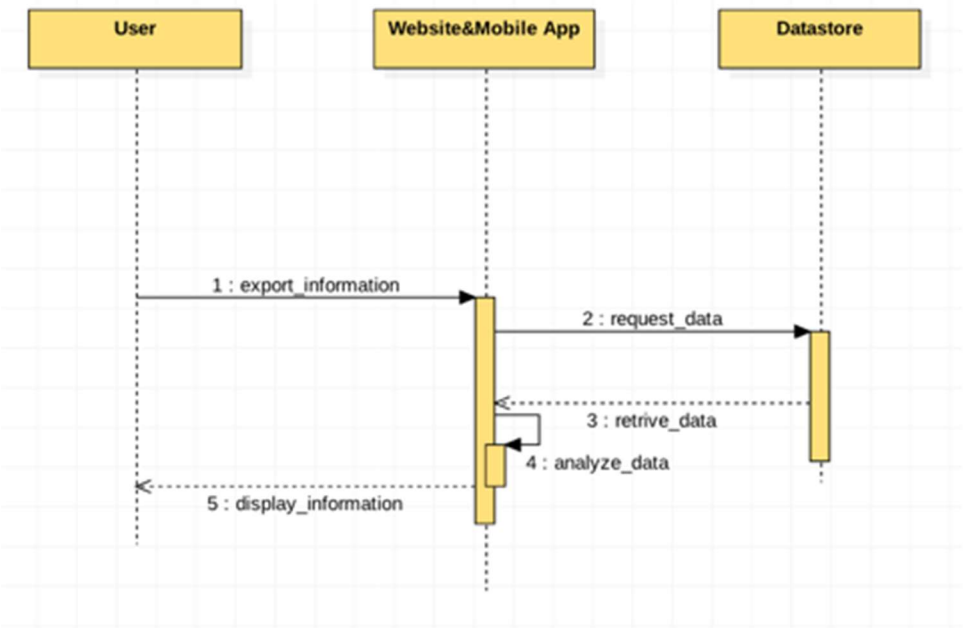


Figure 6-9 - Export information

6.3.5.9 UC-10: Predicting Health Status

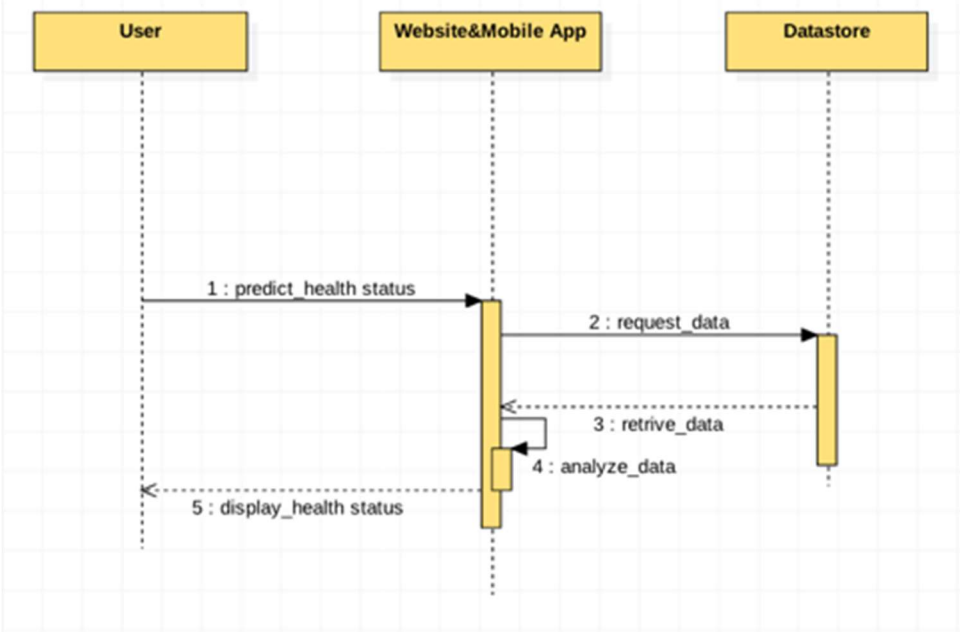


Figure 6-10 - Predict health status

7 User Interface Specifications

7.1 Preliminary Design

Design should be invisible

Users will usually be on the run while using our application (no pun intended). Due to the active nature of our users and how our application will be used, the app needs to be as fluid as the users themselves. They should not have to change their ways of thinking or spend more than a few seconds to start an action while using our app. Since our users will usually be using the application while actively moving or pre/post workout, we need to make sure that the design create a sense of flow that follows the user's current activity.

We want to make sure that when users start to use our application, they don't have to spend more than a few seconds to complete their anticipated action. As the users will most literally be active while using the app, we have to make sure that we cut-down the time in between user actions and action>feedback.

7.2 Screen Summary

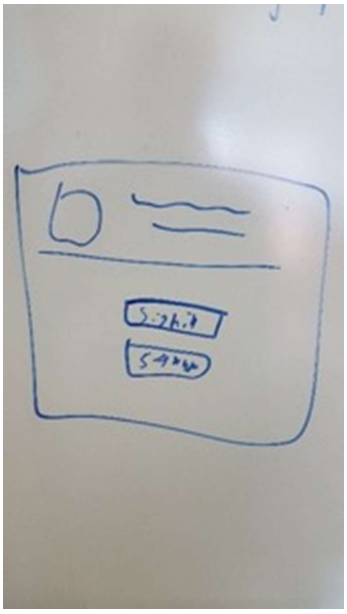
Please see the same named image files for rough depictions of each screen.

1. Landing Screen
    a. This will be the main screen/page that users see when they first open up the website/application.
    b. This can actually be put into two different sub-pages. A landing page for unauthenticated users and one for those we are already signed in.
2. Activity Screen
    a. This will be the screen that users see while recording an activity (running/biking, etc.). We will be modeling some of the design off of already industry proven paradigms while injected our ideas and functionality. While this screen will show the user some information about the current workout, it will be more of a lightweight dashboard compared to the Activity Report screen.
3. Activity Report Screen
    a. This will be the screen that users see after their workout is completed. It will give them a much more in-depth view of their workout.
4. User Profile Screen
    a. This screen will contain information pertinent to the users account. Account and personal information will only be available to be changed in these screens.
5. Community Information Screen
    a. This screen will contain information regarding the comparison of the user and community generated data. This will be less of a 'data-sheet' and more of an interactive display providing insight into the different fitness statistics from the community.
6. Information Export Screen
    a. This screen will provide an easy to access interface to export the user's data into a human-readable format to different data formats.

7.3 Preliminary UI Design

In this section, we will be showing what we think is the preliminary user interface screens for the website and mobile application. Some of these designs are left ambiguous on purpose as some screens may serve more than one purpose in the final implementation. Finally, these designs are of course subject to change as the customer may add/remove/update them.

7.3.1 Landing Screen



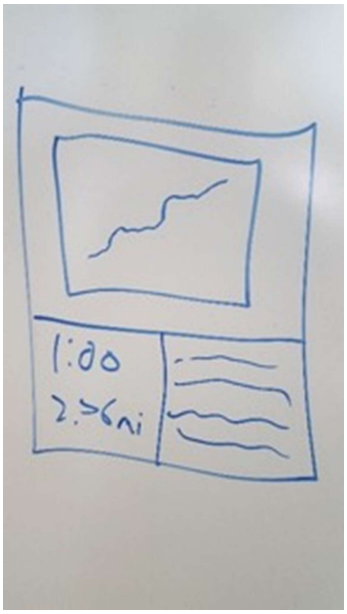
This is a rough sketch of what the landing page will be like when users log in WITHOUT a session.

Up top will be a quick blurb about the application and maybe even a few bits of sample data.

Below, you will find two boxes:

1. *For users that already have an account*
2. *For users that need to create an account*

7.3.2 Activity Screen



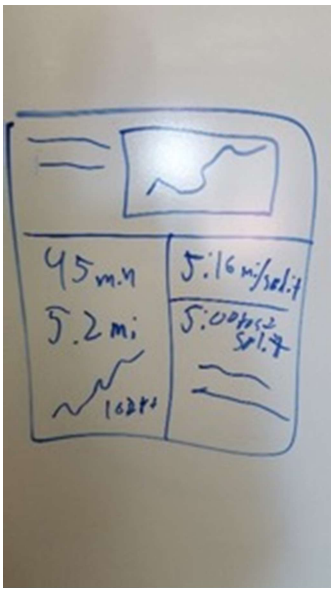
We tried to model our Activity screen after existing market leaders and also on what information we'd want to see.

In the top portion of the page you'll be presented with a map showing your current course.

Below, you will find some more data:

1. *Total Runtime*
2. *Total Distance Traveled*
3. *Time/Distance split information*

7.3.3 Activity Report Screen



Our Activity Report screen is almost like the Activity Screen, just with more information. It allows you to dive much deeper into the data generate during your activity.

In the top portion of the page you'll be presented with a map showing the course that you ran.

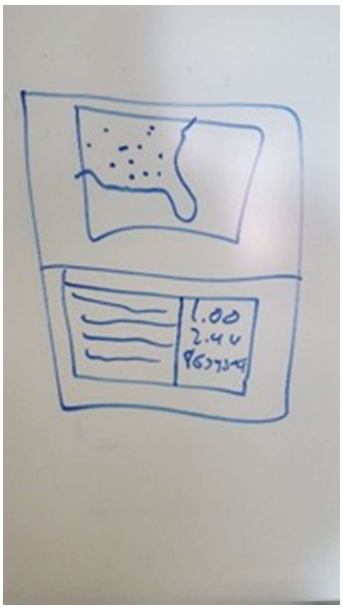
Below, you will find some more data:

1. *Summary of distance and time*
2. *Elevation Change*
3. *More detailed information about your splits.*

7.3.4 User Profile Screen

The user profile/account screen will be like any other profile screen. A synopsis of the user's information and demographical details.

7.3.5 Community Information Screen



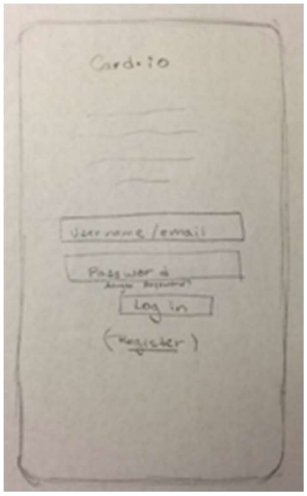
This screen will be the main 'social' screen of our application. We'll have a main map part up top where you can focus in and out of certain areas which will show you a general idea of fitness states in the area.

When you select a certain area or fitness stat on the map, a more detailed representation of that dataset will appear below.

7.3.6 Information Export Screen

The user profile/account screen will be like any other profile screen. A synopsis of the user's information and demographical details.

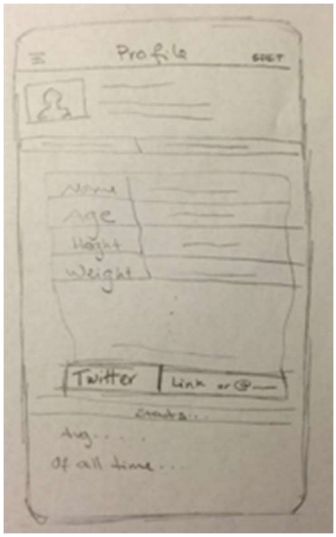
7.3.7 Mobile Landing Screen



The user log in screen will be like any other log in screen. It will ask for username/email and password as well as allowing the user to tap on "Sign in" or "Forgot my password".

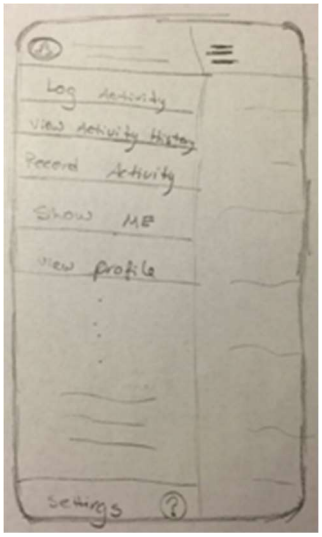
7.3.8 Mobile Sign up Screen

The guests' sign up/register screen. Here guests will be able to create accounts by entering all pertinent information. By creating an account guests will become registered users.



7.3.9 Mobile Profile Screen

The user profile/account screen will be like any other profile screen. A synopsis of the user's information and demographical details.



7.3.10 Mobile Side Menu

This is the side menu a logged in user will see.

7.4 User Effort Estimation

The average amount of effort a user puts in while using the application should not exceed the amount of effort due to the hardest/lengthiest functionality.

Since users will be using the application before/after and sometimes even during a workout, we have the task of making sure the effort of using the application does not interfere with their current activity.

Funnily enough, the most active part for the user will be the least active part for the app. While a user is in their activity, the app will simply be a passive data collector, collecting GPS data and translating that into time and distance stats for their workout.

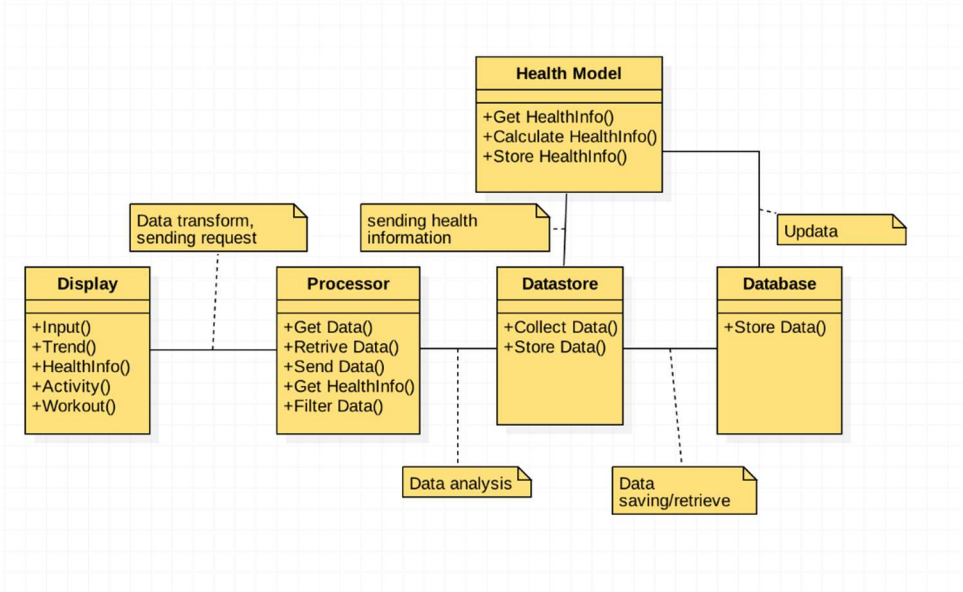
The most effort a user is likely to face is when they:

- Create an account
- Use the community information screen. This is due to the fact that a large part of this functionality is that the user will be looking through other data and entering in stats. This will require the most user input in the app apart from entering in their own stats.

8 Domain Analysis

We will first, derive our concepts and give them a responsibility. Table 8-1 - Concept Definition lists the responsibilities and the assigned concepts of our system. Then we will go in depth to what each concept does, how each concept interacts, how data flows through our system and how the mathematical model of one of our concept is approached.

8.1 Domain Model



8.1.1 Concept Definitions

The concepts and responsibilities from the use cases detailed in section 6.3.

Table 8-1 - Concept Definition

Responsibility	Concept
Display user and system information. Allow user to enter information.	Website
Display user and system information. Allow user to enter and record information.	Mobile Application
This will allow the Website and Mobile Application to retrieve/send user data to/from our server. The Processor	Processor

concept will also allow other interfaces (like universities, etc.) to interact with our data programmatically.	
Allow web, mobile and health model interfaces to create, store, retrieve, update and delete information from the database.	Data Store
Contains “perfect health” model. Allow comparison of users (and communities) with health model. Make predictions about the heart rate, blood pressure and cholesterol of users. Makes recommendation to improve the health of users	Health Model

8.1.2 Concept Explanations

- **Web Interface** – The main focus of this concept is so that users can interact with the app using a modern web browser. While the mobile/native interface may give it increased interactivity in some places, the web interface will allow them to input data at a much faster rate with more detail.

- **Mobile Interface –** This concept is in charge of interfacing with the user via a mobile device. It will display user-requested information (like user profile and activity history) and will allow for user-entered information to be stored (like adding a new activity). This interface will primarily interact the *Data Store* concept as well as the *Guest* and *User* actors.

- **Processor –** Accepts requests from the either the Web and/or Mobile Interface and then passes this information on the datastore.

- **Data Store –** The data store will act as the central repository for all data provided by the users as well the data used by the health model. The web interface, mobile interface, will send and receive data related to the users. The health model receives user data for the purpose of enhancing the current model.

- **Health Model –** This concept creates and maintains the models used to predict the health of the user. These models predict the heart rate, blood pressure, cholesterol levels, and the health index of the user. Based on these values, the

models can make recommendations to improve the user's health. The health model concept interacts with the Data Store to build and update the model as users input data and also with the Mobile and Health Interfaces to display the predictions.

8.1.3 Association Definitions

Table 8-2 - Association Definitions

Concept Pair	Association Description	Association Name
Website <-> Processor	The website gets data from the user (during login, update profile, etc.) and needs to send it to the Processor for retrieving or updating the user's data. It also does it when it needs to calculate certain values like calories.	Retrieve/Update/Calculate
Mobile Application <-> Processor	Same as the Website <-> Processor	Retrieve/Update/Calculate
Processor <-> Datastore	The processor will retrieve or store any user information in the database, but to do this it needs to do it via the datastore.	Retrieve/Update
Health Model <-> Datastore	The Health Model concept will generate out state-level health model decision tree and export it as a JSON file, the Datastore concept needs to read these files to look-up user information and national statistics.	Store

8.1.4 Attribute Definitions

Given the long list of Attributes, we decided to merge this list with the Operation Signature list found in section 10.2 **Attribute Definitions & Operation Signatures** below.

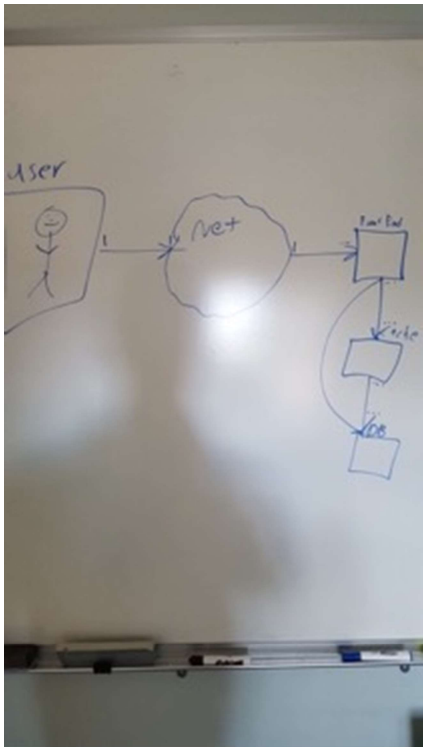
8.1.5 Traceability Matrix

In this section we will map each concept defined in section 8.1.1 above to each of the use cases defined in section 6.3 above.

Table 8-3 - Domain Model Concept and Use Case Traceability Matrix

Use Case	PW	Web Interface	Mobile Interface	Processor	Data Store	Health Model
UC-1	13	X	X	X	X	
UC-2	22	X	X	X	X	
UC-3	8	X	X	X	X	
UC-4	22	X	X	X	X	
UC-5	36	X	X	X	X	X
UC-6	16	X	X	X	X	
UC-7	36	X	X	X	X	X
UC-8	15	X		X	X	X
UC-9	20	X		X	X	
UC-10	14	X		X	X	X
Max PW		36	36	36	36	36
Total PW		202	153	202	202	101

8.1.6 General Data Usage Flow



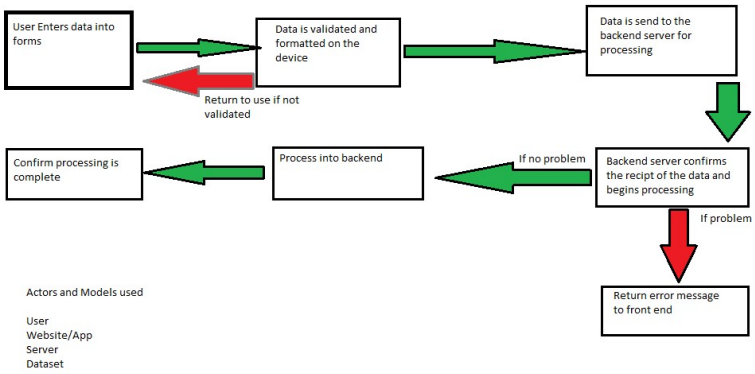
While the idea for the system may seem complex, the actual workflow behind it is very simple.

The user will be using the website which in turn will access the World Wild Web.

Once the data gets sent to the servers, it will go through the different levels.

1. *Frontend- This will handle displaying the data*
2. *Cache- 1st level of data retention*
3. *Database- 2nd level of data retention.*

Below you will find a more concise workflow of how the data is processed. A full-size file is available in the resources directory.



8.2 Mathematical Model

Our application uses several models in order to make several predictions about the user. Firstly, we will use a Correlated Stratified Monte Carlo Sampling method to generate our state population. While we want to generate models which can predict the health statistics of an individual on a state level, the state level information required to make such predictions does not exists. Secondly, we use regression trees in order to classify our sample data in order to predict the heart rate, cholesterol and blood pressure of the individual. Given a set of input data, each model should return a given range where the user's health statistic should lie (e.g. the cholesterol predictor returns a range of cholesterol levels that a user should have based on our dataset). The categories used to classify our data set include basic information about the user, such as the age, gender, weight, height and waist circumference.

The algorithm by which generate our sample will be inspired by Particle Filtering, which is a class of Monte Carlo algorithms which we can use create a sample population based on marginal distributions of our population. In this way, we can simulate a sample which accurately represents the states' populations. To generate this model, we will use two techniques: Completely Random Sampling and Correlated Sampling. The first sampling method will us only a Pseudorandom Number Generator in order to generate a random sample. We then augment this sampling by using current research produced by the researchers in the field to make our simulations more accurate. (The periodicity of this PRNG and the Correlated Sampling research is discussed in the Algorithms section). To test the accuracy of our sample, we will compare our statistics on a National and State Level in order to measure how close our sample matches the results of current publications

The algorithm which we will use for our regression tree is inspired by the ID3 algorithm, which first starts with a root node with an attribute that minimizes the Shannon Entropy of the whole dataset, splits the dataset based on the node attribute, makes decision tree node based on the attributes, and recurses through the remaining subsets. The ID3 algorithm is well-suited for Boolean classification (yes/no decisions), but our goal is to return a health statistic based on the user's physical properties. Instead of using Shannon Entropy to decide when to branch, we will aim to minimize the range of our prediction. Since we are aiming to minimize our range or predictions, we can either let the algorithm continue until it has no more attributes to choose; in this case we can return a range of health statistic within a confidence interval. We can also fix a particular range such that a node will not be split if a given health statistic if the confidence interval is smaller than the fixed ranged we specified. We will also try to reduce the computational cost of creating the model by using a method called particle filtering, where compress the large range that attributes like height can take into bins such that each participant has equal probability to be placed in any bin; this allows us to classify our data set more quickly by carefully reducing the number of choices our model is allowed to make and makes the decision making more amenable to the modified ID3 algorithm.

We will test the accuracy of our models by reserving a part of our sample data solely for cross-validation. This is so that we may simulate the performance of the model as if users are inputting their personal information and to. In later stages, we will also further test the validity of our hypothesis (the accuracy of our model) by using an ensemble learning method named boosting, where we shall develop the model through iterations, where each iteration learns from the incorrect decisions made in the previous iterations to produce a more accurate model.

The reason for choosing such a simple model like the Decision Tree is because of the data we need to making meaning prediction is not yet open-sourced. While there exists other more sophisticated algorithms which we could use to generate our prediction, there does not exist any information on a state level which we can use for predicting the health statistics of an individual; our decision to simulate a state's population was one of last resort. Given the uncertainty of the type of data which we would be able to find, we decided to used the Decision Tree so that it can accept any data which we were able to generate. Since the models are built and verified using the sample state populations which we generate via Supervised Learning, the accuracy of our prediction is tied much more strongly to the accuracy of our data, instead of the model.

9 Interaction

This sections expands on our system's design by explaining how actors and concepts in the system interact, unlike sequential diagrams, here we show more details in the interactions themselves.

First we should present how each concept fit together in the overall system. As you can see from the figure below, the Website and the Mobile Application both interact with the Processor, this is mainly for retrieving and sending user data, this is done via JSON-over-HTTP in a RESTful manner. The Processor and the Datastore interact via regular PHP method calls, done so that the Processor doesn't have to implement any database-style queries, decoupling them and allowing us to change databases if need-be. Finally, the Health Model concept and the Datastore interact via JSON files that get generated once a week. The architecture of our system resembles "Microservices" where the processor provides web services to any interface and/or program.

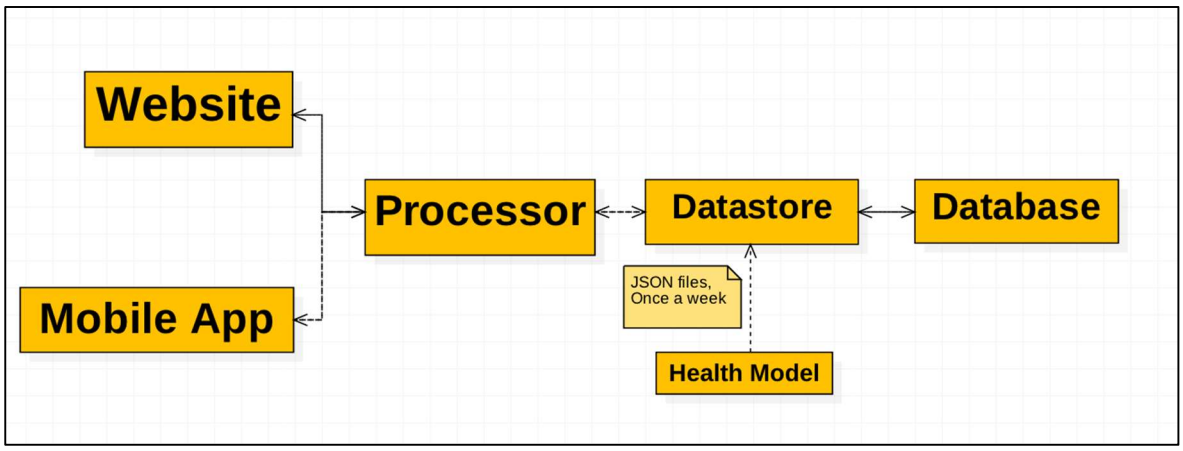


Figure 9-1 - Overall System Diagram

9.1 Responsibility

When we assign the different objects/models certain responsibilities, it's imperative that we bring long-term ideas into the picture. It's very rare that something will stay the same from conception to production. There are multiple facets of design and implementation that you have to take into consideration.

- **Scope –** When creating objects, you have to make sure that its current actions don't extend past its expected area of action. This helps stop objects from getting involved in interactions that don't have anything to do with their respective actions.

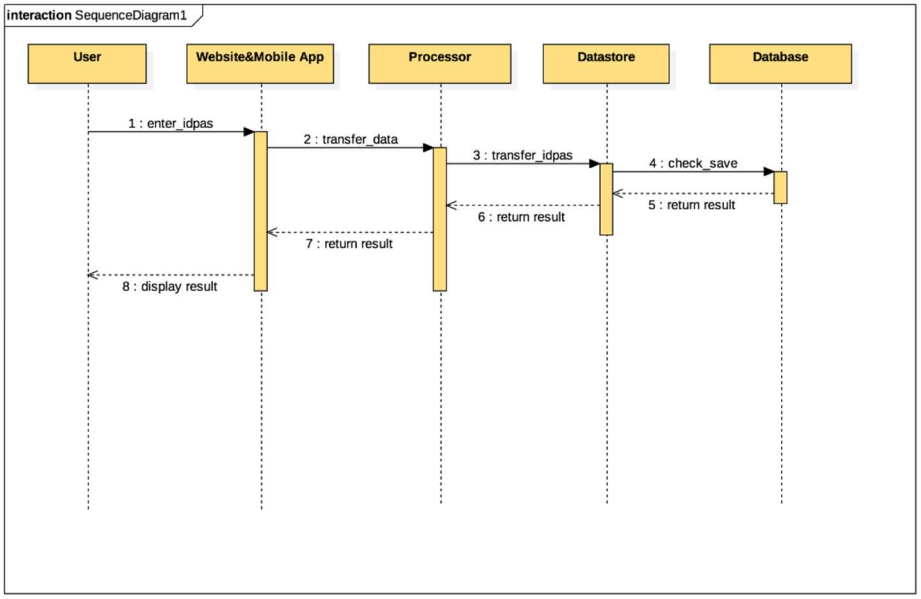
- **Scale –** We are not worried about the code at this point, but rather if the objects design or ideas can scale well. Would a basic/kernel level object conceptually be able to handle hundreds or thousands of action points instead of a few dozens.
- **Relationships –** Are the relationships between the actors and models responsible and logical? A user should never directly access to touch the database, that is why we have the front end/data-store models between those two actors.
- **Model-View-Controller (MVC) Design Approach –** Even though the figures and sections below focuses on the interactions and relationships between our objects (actors, concepts, etc.) it should be mentioned at this stage that our system design approach has been a "Model-View-Controller" approach. We chose this as we are building a (mostly) web application and this approach is very popular in the web (as well as very well documented and tested). So as we go discussing our system, we will break it down into the view (website and mobile application), the model (user data and health model) which is housed in our data-store (backed/persisted by the data base) and finally the controller (our processor/server) which will allow users to interact with the data.

To make sure that you can view the interaction diagrams clearly and neatly, we have also attached exported images of them along with this report.

9.2 Interaction Diagrams

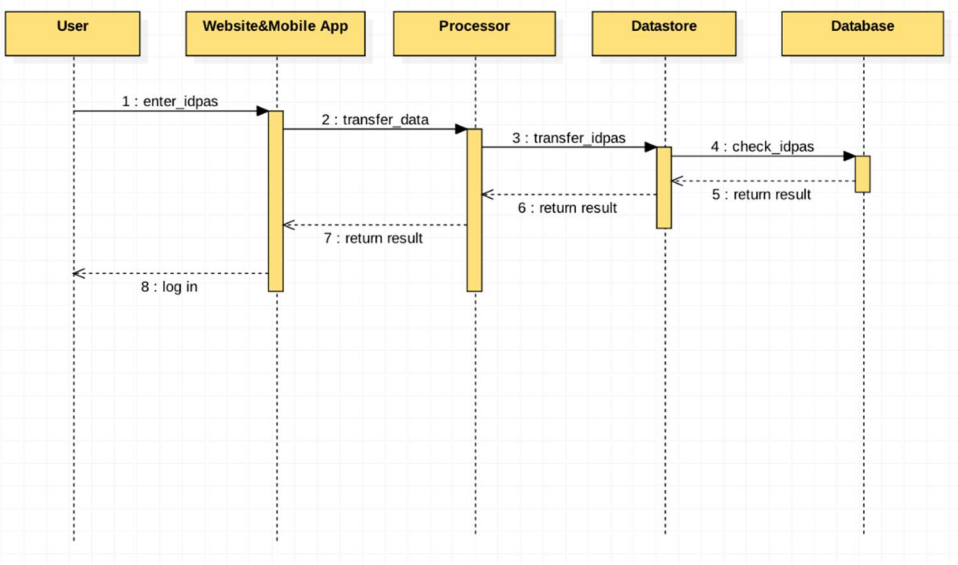
In the previous report, we had already created sequence diagrams for all of our use cases. In this section we thought we should describe, in greater details, the interaction between the concepts of our system, this will be done by creating interaction diagrams of four of our core use cases and discussing each one to present our assumptions, sub-sequences, exceptional paths etc.

Use Case 1 – Sign Up



When the user clicks on the “Sign Up” button, our website and mobile app would send a request to processor to prepare storing the user’s information. While the user finish setting the user name, password, the processor would send the data into Datastore. When all the data has been stored, the processor would return the result so that the user could use the system.

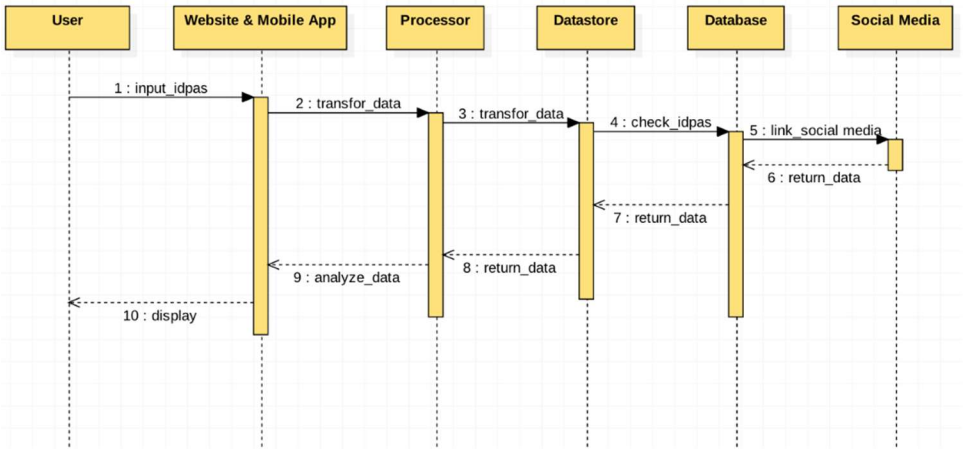
Use Case 2 – Log In



The whole progress of LogIn use case is similar to the SignUp use case. When the user clicks on the “LogIn” button, our website or mobile app would send a request to processor to prepare comparing the user’s information. While the user finish inputting the user name and password, the processor would search the username and password in the Datastore. If the inputting data is consistent to the data stored in the Datastore, the processor would let the user log into the

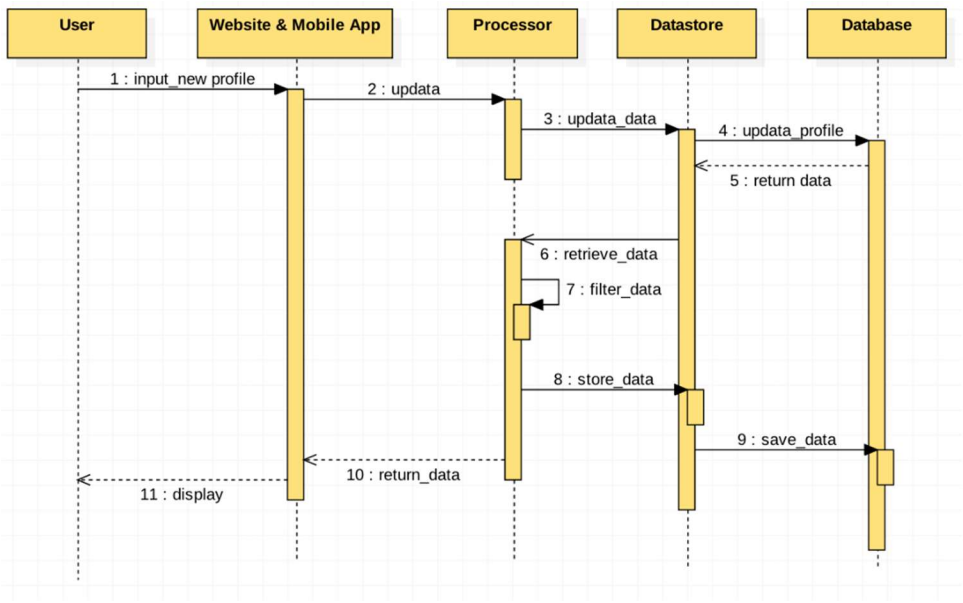
system and show the main interface. If the inputting data is incorrect, our website or mobile app would refresh the LogIn page so that the user could input again.

Use Case 3 – Link Social Media



The " Link Social Media" use case is the function that allows user use their account to link social media. When user login our website or mobile app, they can choose to link social media. Our website would send the request to processor, and the processor would search username, and password in the database. After check, database would allow user to link social media, and the user can show their health information.

Use Case 4 – Update Information



The " Update Information" use case is that can make user to modify their profiles, when user input their request though our website or mobile app, and our system would send the update requirement to processor, and when the processor gets the instruction, it would send update to datastore, finally, our database would get the update requirement, and filter the user

information, and send it to processor, processor would modify the profile depend on what users imputed, and completing this step, the update information would be store in the datastore, and meanwhile, would also return the update information to our website and mobile app, and display to users.

9.2.1 Use Case 5 – Add Activity

The "Add Activity" use case presents the interactions between the concepts needed to allow a user to add a new activity. In our system an activity is any food item or workout session the user wants to add to their account. These activities will be kept in the user's account and will aid the health model in calculating health status. All activities will be stored in the database so later the user can see them from the history view.

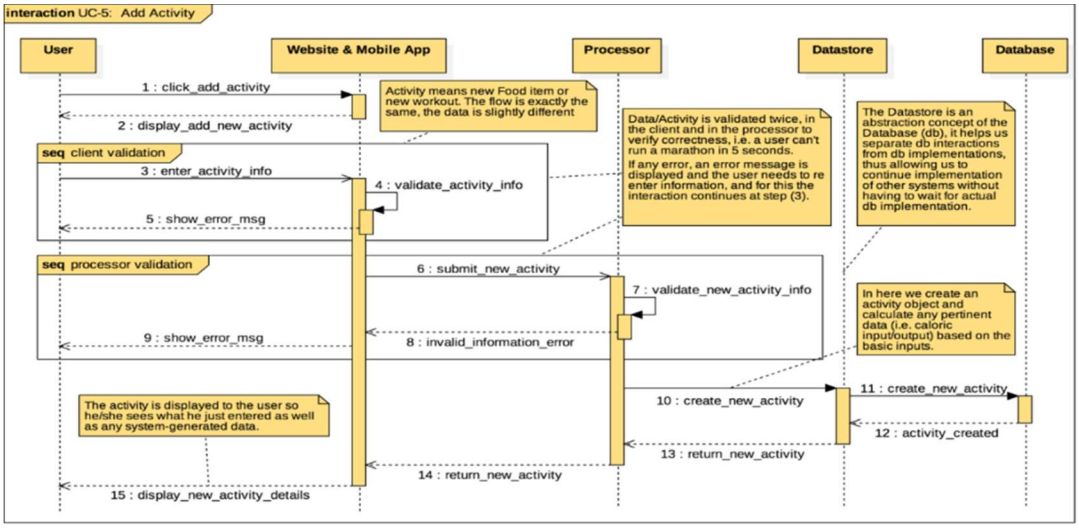


Figure 9-2 - Interaction diagram UC-5: Add Activity

9.2.1.1 Notes

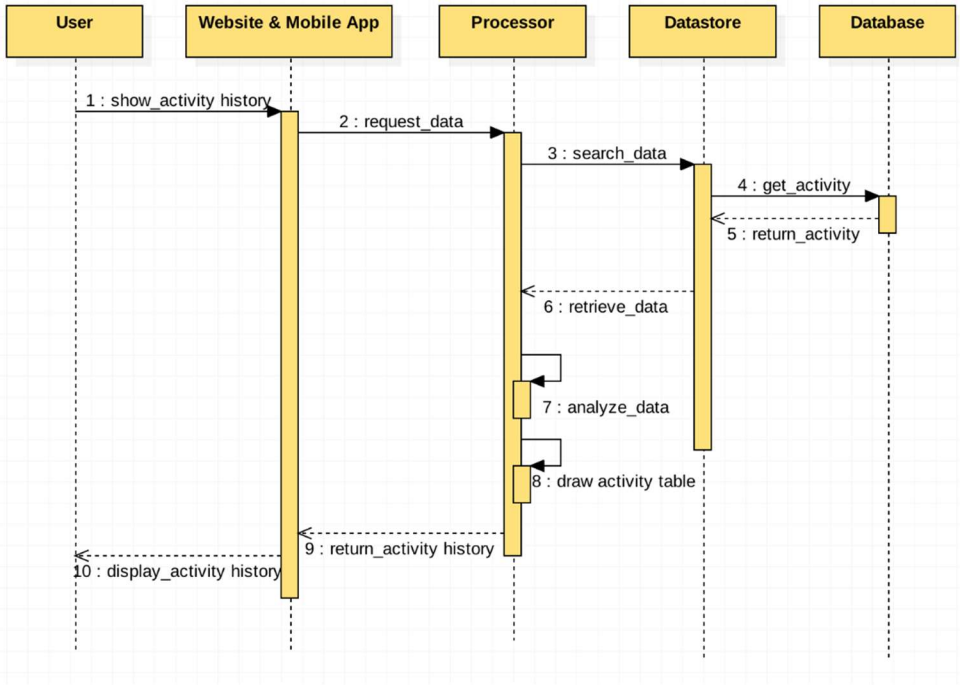
Let's start with some assumptions taken while designing the interaction diagram seen in Figure 9-2. The initiating actor here is a user, which means the user should have an account and already be authenticated/logged-in. Another assumption taken is that the user will interact with our system via the website or mobile application, both of these have the same ability of creating a new activity and the flow of data is similar enough that it both of these interactions can be grouped under one concept here. Another one is the idea of data validation, as shown in the diagram we will do validation twice; in the client we validate input, i.e. things like valid values, no numbers in text-only fields, negative values, etc., and in the processor where we do a bit more complex validation, things like impossible speeds, lengthy distances, future entries, etc.

As far as the interaction goes, the steps are self-explanatory, the user initiates by navigating to the add new entry view, the interface will display a new activity form where the

user will choose the type of activity and add activity details. Here is where we do our first validation step (client-side validation), if any error is encountered, the user is shown an error message and be allowed to re-enter the information (fixing any errors) and resubmit. Once everything passes client validation, the interface will send it to the processor for further validation, again same as with the client, during a failed validation an error will be sent to the interface to display to the user and the user will be allowed to re-enter and re-submit. Once all validation passes, the processor will perform all needed calculations (i.e. caloric input/output, pace/distance, etc.) given the activity details and ask the data store to create a new activity. The data-store will interface with the database (our other participant) which will store this new activity as part of the user's history and once this is done it will let the processor know the activity was created. Finally, the processor will reply to the interface that the activity has been created so the interface can display a successful message with the details of the new activity. This entire interaction is sequential in nature, meaning that the user will have to wait until the entire transaction is processed to finally see the successful message.

One last note we would like to point out is the architectural/design element used to create this diagram. As explained above, we are using the MVC as our architectural approach, meaning the user is interaction with the interface (view) modifies the profile/activities and gets health information via the (model) and the processor (controller) is in charge to actually doing the data modifications via the data-store.

Use Case 6 -- Show Activity History



When users require to show activity history, our website or mobile app would send a request to the processor, then the processor would make datastore connection, and datastore would

connect to database to retrieve useful data. and the data would be sent to the processor, and the processor would analyze data, and after that, would draw an activity history table, then this table would be returned to the website or mobile app, and user can see the activity history.

9.2.2 Use Case 7 – View Health Information

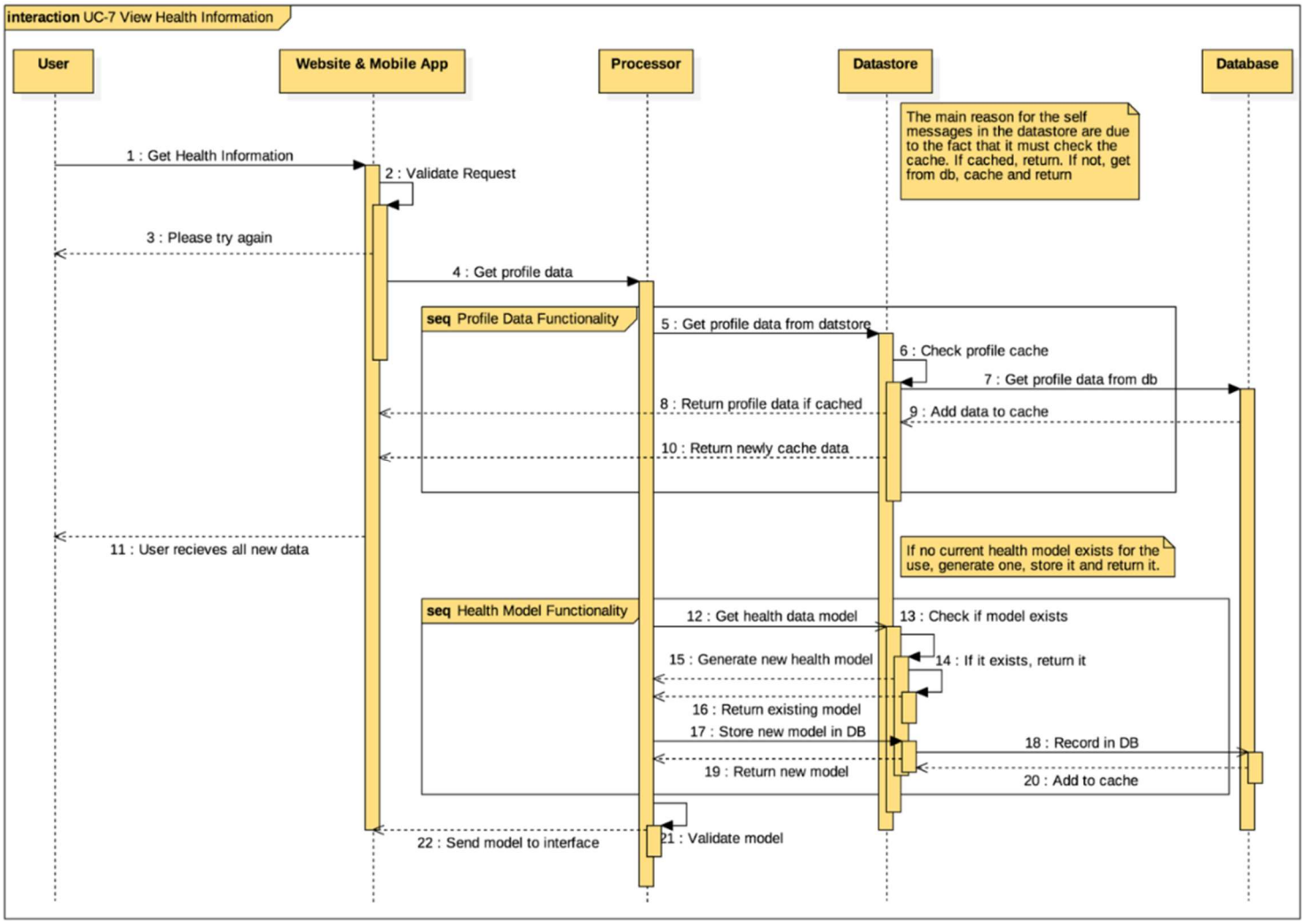


Figure 9-3 - Interaction diagram UC-7: View Health Information

9.2.2.1 Notes

UC-7 describes how one would view their health information from either the Web or Native Application interface. This use-case is mainly a passive one (from the users POV) as it is just about generating, returning and displaying data to the user. Since this use-case interacts with a large number of different datasets and models, asynchronous functionality is important.

This will make sure that the user or system will never be locked up while trying to make a request.

As this deals with a large number of data requests and data generation, this will be one of the largest computationally expensive tasks. While it is a computationally expensive functionality, the user can passively stand by since all of their information is already entered in the database. Depending on the scale of the operation, user process feedback will be important so they know what is happening within the system.

9.2.3 Use Case 8 – View Community Information

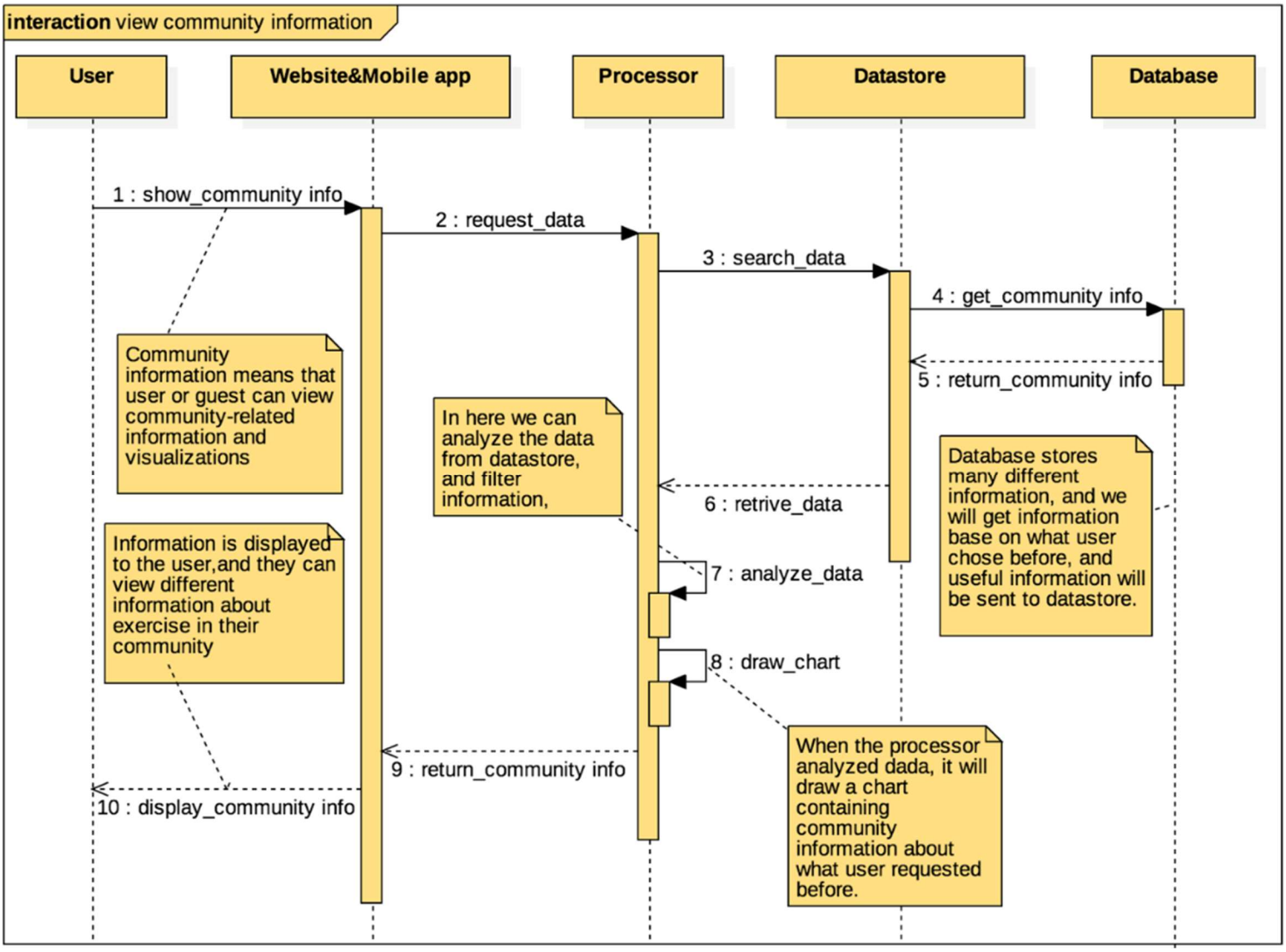


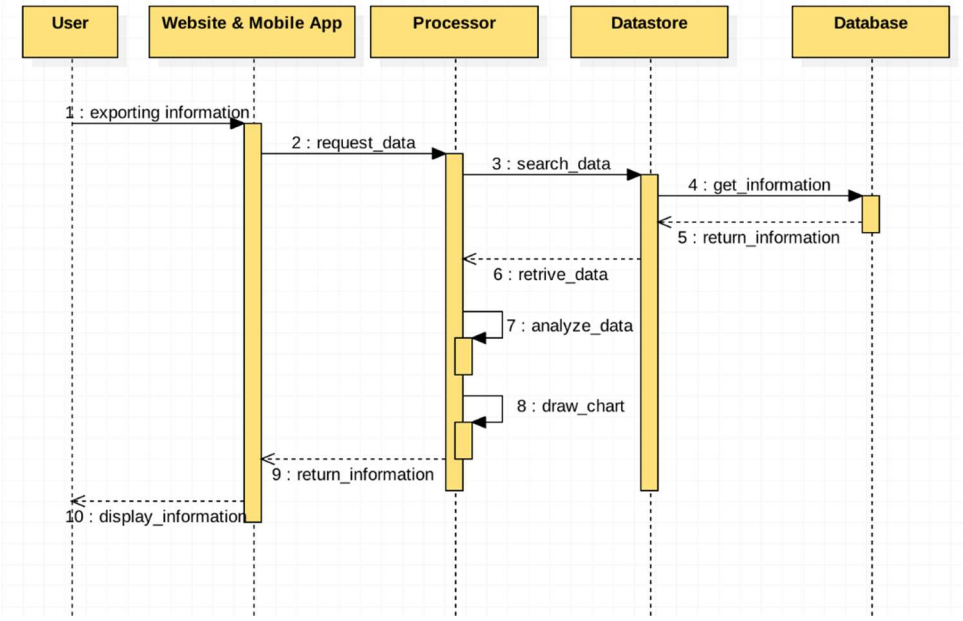
Figure 9-4 - Interaction diagram UC-8: View Community Information

9.2.3.1 Notes

UC-8 describes how a user can view the community information, to check what kind of exercise is popular in their communities, and how other people do exercise. when users require to use the Community Information functions, the website & mobile app would send a request to the processor, then the processor would access to data store to retrieve useful data from the database. In this use case, we need the location and the what kind of exercise in the database. Then the processor would calculate the data and then make a chart of community information. When all work done, users can view the community information on the screen.

This is about how user can get information from our website and mobile application. It is a important function in our project, we store community information in our data-store, and user can check information which is already in our data-store. User and guest can use their accounts to access to website and mobile application, and our system will display information depending on what user request after analyzing data from data-store. And they can make their health schedule effectively by comparing with other people in their communities.

Use Case 9 -- Exporting Information



When users require to show activity history, our website or mobile app would send a request to the processor, then the processor would make datastore connection, and datastore would connect to database to retrieve useful data. and the data would be sent to the processor, and the processor would analyze data, and after that, would draw an activity history table, then this table would be returned to the website or mobile app, and user can see the activity history.

9.2.4 Use Case 10 – Predicting Health Status

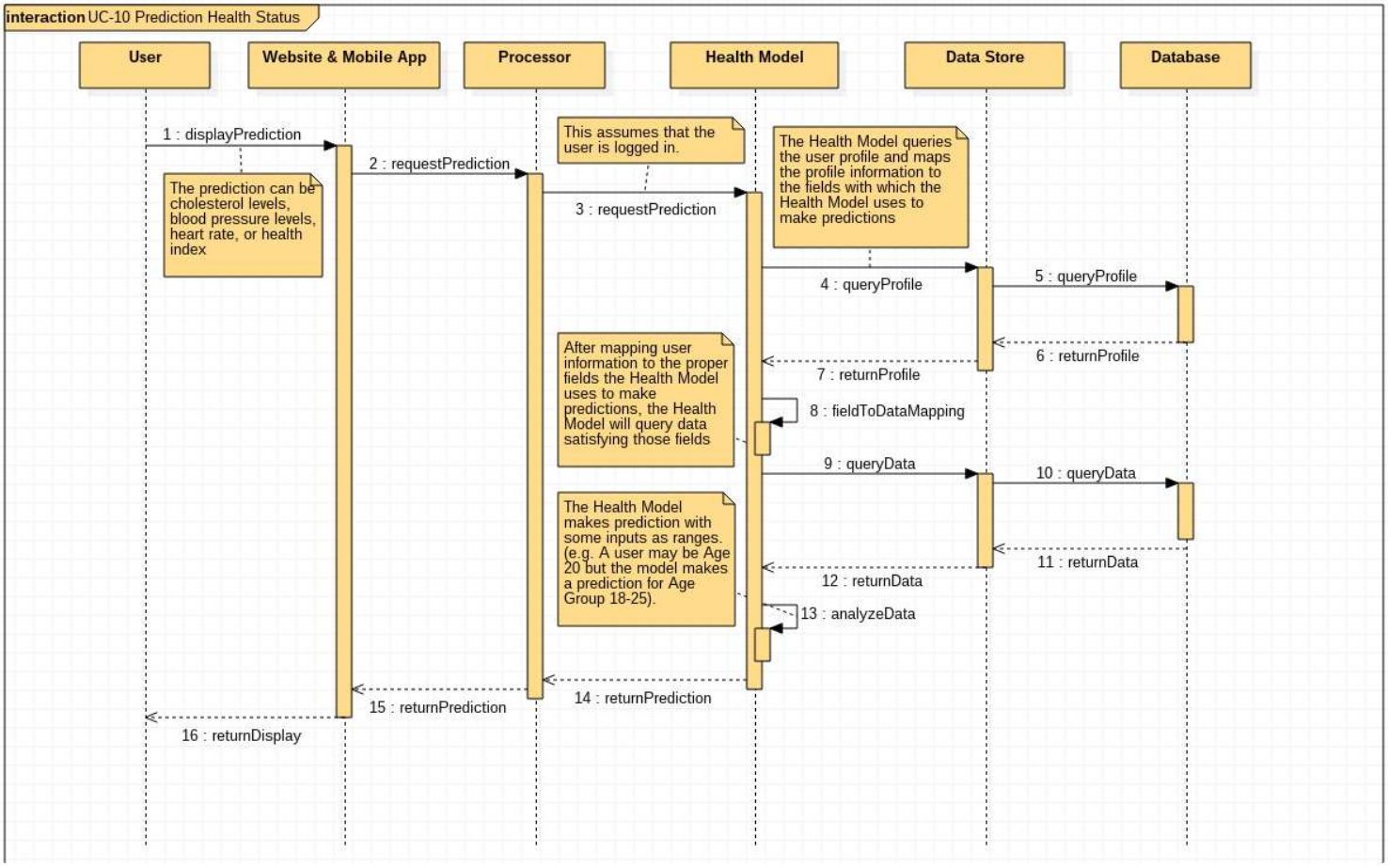


Figure 9-5 - Interaction diagram UC-10: Predicting Health Status

9.2.4.1 Notes

UC-10 describes how a user can view any health statistic our models provide, which are cholesterol levels, blood pressure levels, heart rate levels and health index. This use case assumes that the user is signed in and that a session variable is stored within the browser. This session variable stores information about the user's profile and exercise level, and that the user has inputted basic information to the profile of the user. The user initiates this use-case by requesting the Website (or Mobile App) to display the health statistic. This display request is passed from the Website to the Processor, that then forwards this to the Health Model. The Health Model sends a query to the Data Store who then sends the query to the Database. This query is to get the user's profile information and exercise level. This is necessary because certain attributes to the model are expressed in ranges. For example, a user may be 20 years old, but the model may express age in the form of Age Groups (e.g. Ages 18 – 25). Once the

Database returns this The query made is based on the profile information of the user. The data is returned to the Data Store who then forwards the data to the Health Model. The Health Model then performs calculation on the given data to predict the health statistic for the user. The resulting statistic is pass to the Processor, that then forwards it to the Website (or Mobile App) interface. This interface then returns to the user the display for the health statistic. If, however, there is no session variable, the Processor will cancel the display request and the Front End will return a default error view to the user.

One choice that arose when detailing this interaction diagram was how the Data Model and the Data Store interacts with the Processor. We considered that the Processor first queries relevant information from the Data Store. After receiving the information, the Processor then passes the data to the Data Model. After the Data Model performs the required calculations for the health statistic, the statistic is returned to the Processor, that ten forwards it to the Front End. Having the Processor be an intermediate point between the Front End, Data Store and Health Model would make this process more coordinated, we decided that the same result can be achieved with fewer steps if the Processor interacts with the Data Model, who now has the ability to interact directly with the Data Store.

10 Class Diagram and Interface Specification

In this section we will go into details of each class that makes up our concepts, we will present class diagrams that show their associations with other classes, attributes, class signatures and other details. This will allow us to start breaking down our system into concrete parts that can be implemented during development. We later design how these parts communicate with each other so implementation and testing can be streamlined given the short timeframe we have.

10.1 Class Diagrams

We will break down the class diagrams into the five concepts we have, the Mobile Application, the Website, the Processor, the Datastore and the Health Model.

10.1.1 Mobile Application

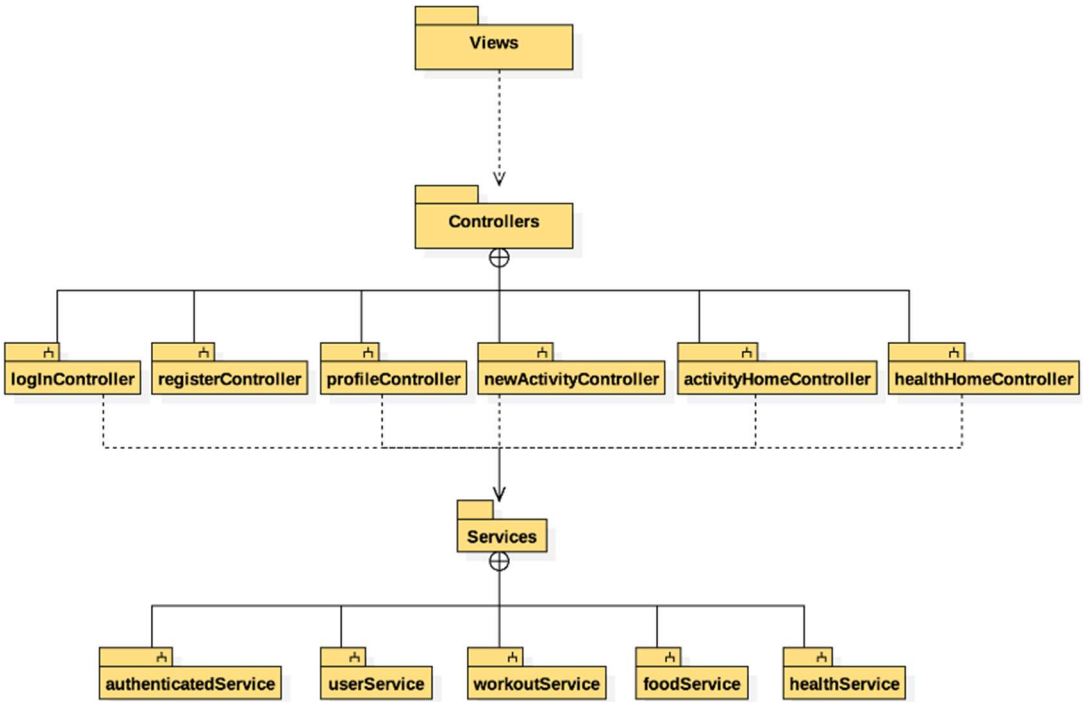


Figure 10-1 - Mobile Application Overview Diagram

The Mobile Application concept will be broken down into three packages (or sub-systems) which follow the MVC architectural principle which we will go into a deeper explanation in Section 5.1 below.

To implement this MVC principle we need three parts or packages, first the views at the top are what the user interact with, in our mobile application these views are HTML-like pages, since there are plenty of views we aren't listing them here. The next package is the controller, this one allows the processing of events sent by the views (i.e. user clicks on the "Log in" button), it also allows the mobile application to interact and modify the model (or data), for example if a user trying to register enters their information the registerController can verify and validate the data entered before passing it to the next level. Finally, the services package will allow our mobile application to pull or push data to the processor (via RESTful APIs).

It is important to note that each package has a distinct role and that not all of the classes contained in the controller package will interact with all the classes in the services package and so on. To see how these interact with each other, please refer to Figure 10-2 - Mobile Application Class Diagram below.

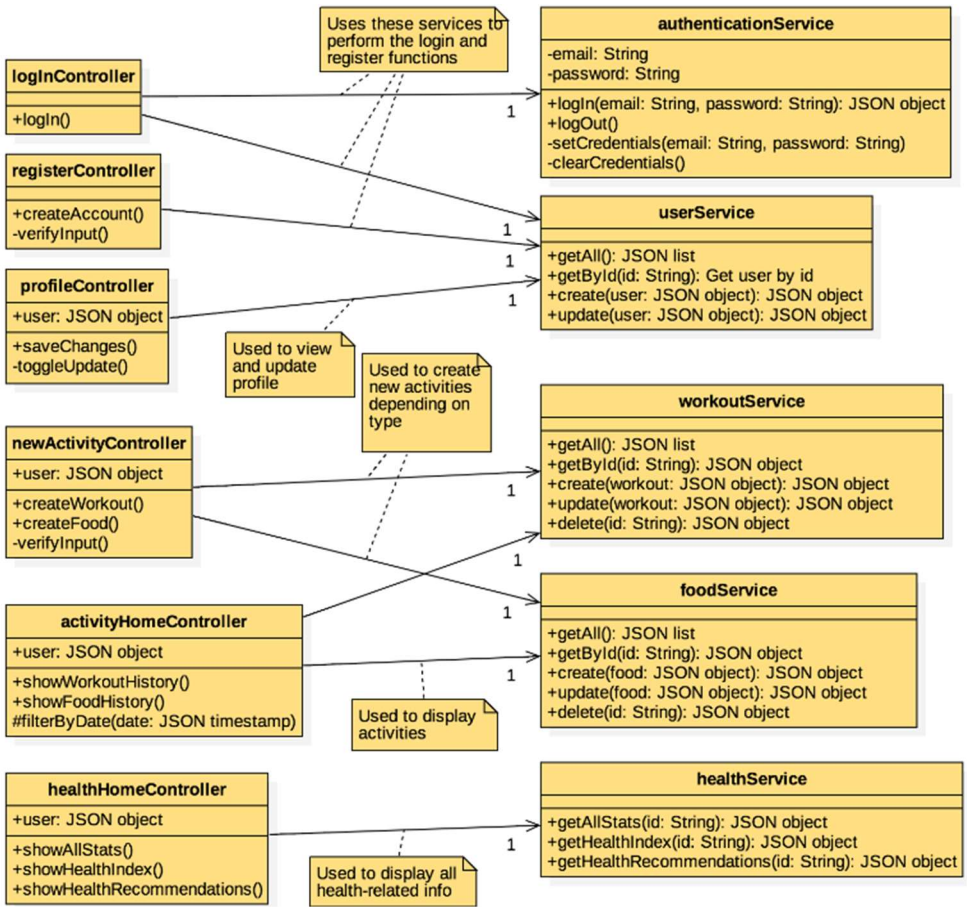


Figure 10-2 - Mobile Application Class Diagram

10.1.2 Website

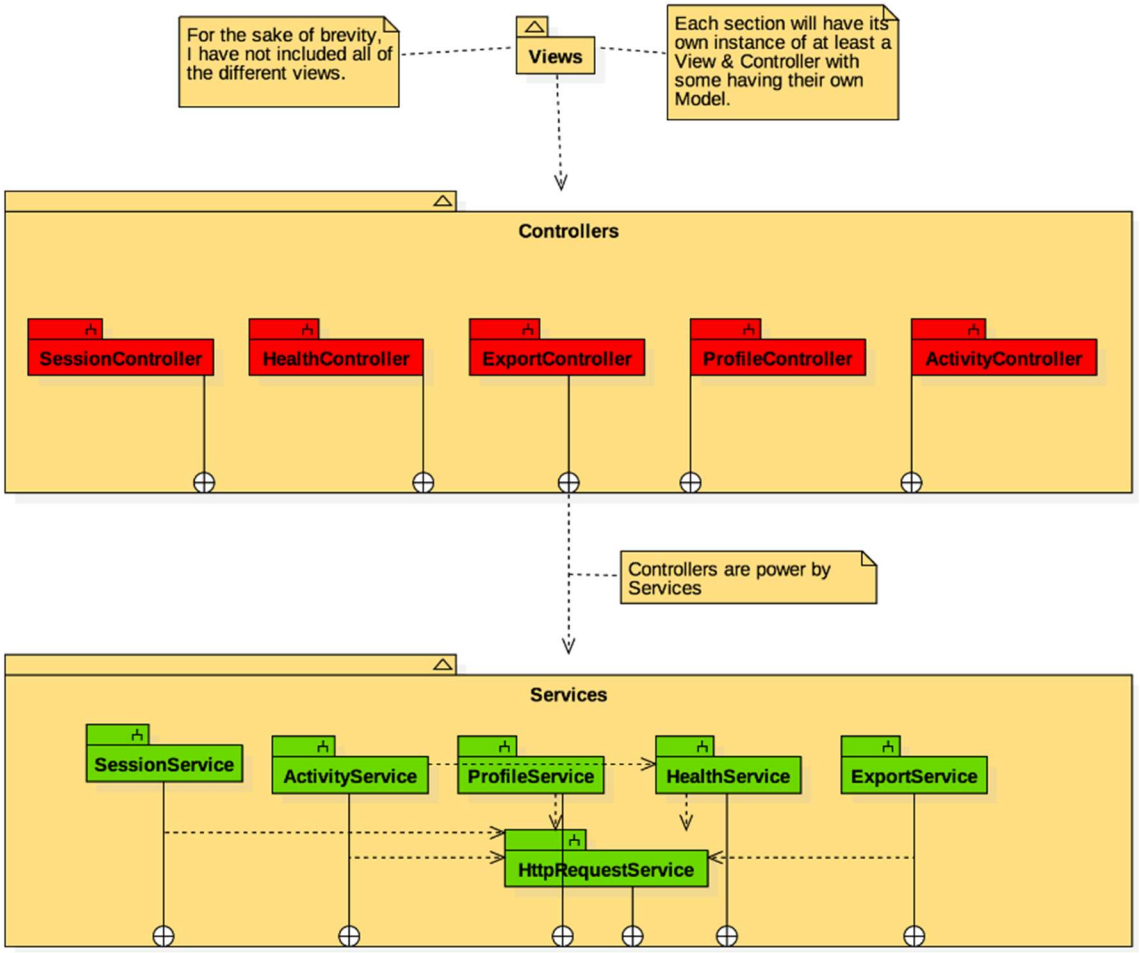


Figure 10-3 - Website Overview Diagram

We will be use the idea of a MVC architecture for our website. While it usually models a Model, View, Controller paradigm, due to the style of our website we are deviating from that a bit. Instead, we will have a View, Controllers & Services powering our site.

10.1.2.1 Views

The views layer will be what displays information for the user. This will be what contains all of the html and user-facing elements for the site & application. The user will always have to interact with a View to enact functionality within the system.

10.1.2.2 Controllers

The controller layer contains Controller objects that will power the View objects in the View layer. They contain the pieces of functionality that take requests and pass the data to and from the views. If a user executes a request or functionality from the View layer, they will have to pass through to the Controller layer[1] to accomplish the end goal of that request. In our implementation of the MVC architecture, Controllers are the middleman between the View & Services layer.

10.1.2.3 Services

The services layer is the main processor of information within our architecture for our site. It is the entry & exit point for all data that goes through our site. They process most of the information that comes to & from the controller so that all the controller has to do Is to set it up for view rendering.

We included an extra HttpRequestService object to handle all of the HTTP Requests that our system makes. This allows us to have a single focal point for request handling & management.

10.1.3 Processor

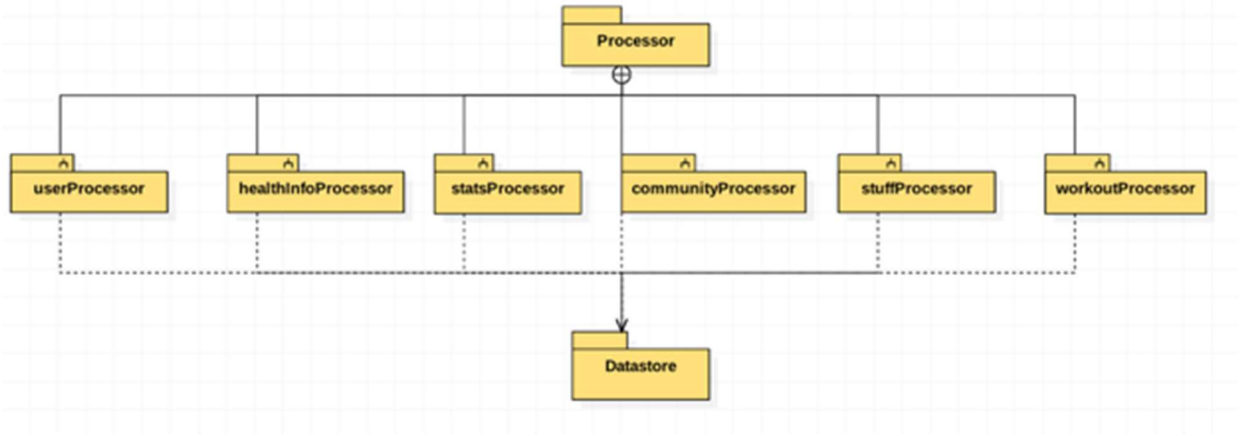


Figure 10-4 - Processor Overview Diagram

The Processor concept will have six separated logical units to deal with different logical groupings of functionality. The Processor is as a bridge between website/mobile app and the Datastore, it will be in charge of all of the "business logic", that is, all of the logic needed to

[1] Except in cases of very direct/sparse information requests.

finish a needed task within the system. We will go into a deeper explanation its implementation in section 12.1.3 below.

10.1.4 Datastore

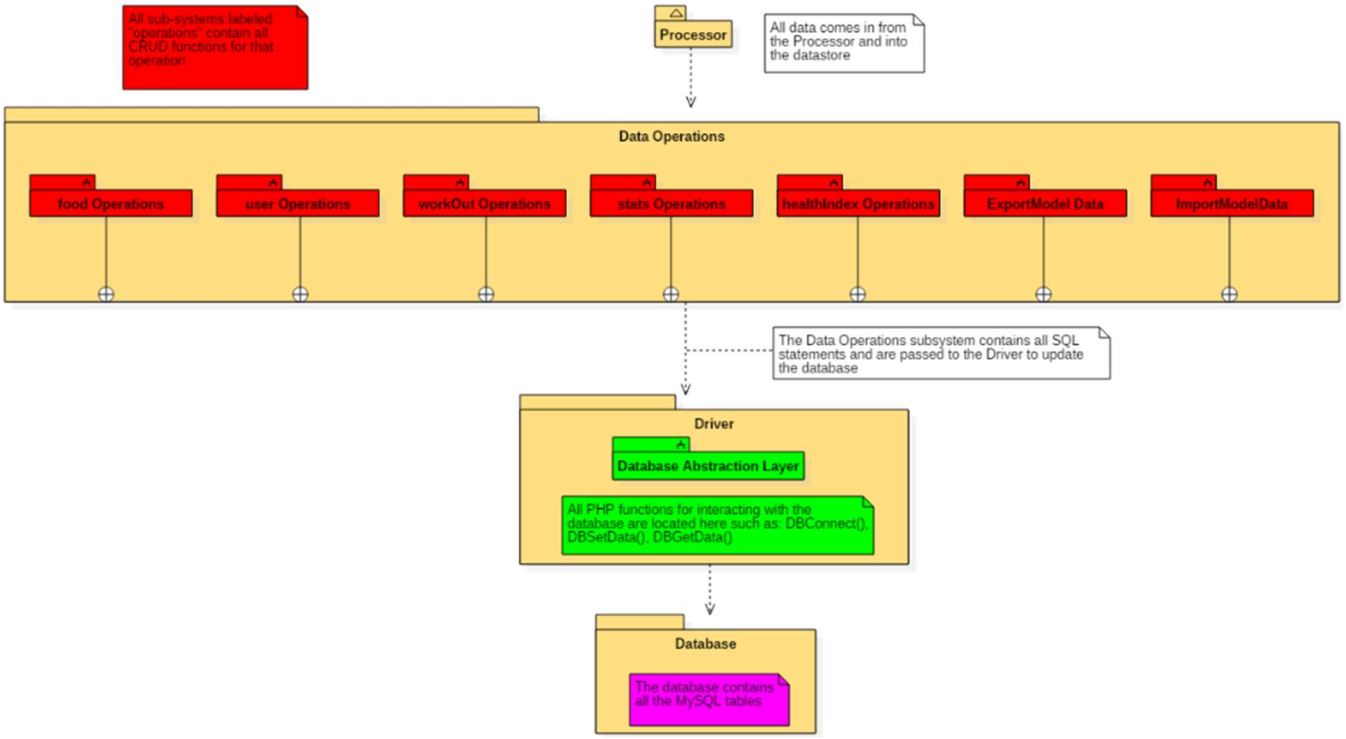


Figure 10-5 - Datastore Overview Diagram

Figure 10-5 - Datastore Overview Diagram shows on overview of the Datastore. We will break this concept into two logical systems, the Data Operations system and the Driver system.

The Data Operations system will allow other concepts to interact with the Datastore. Similar to the MVC paradigm, the function signatures provided by each class in this system can be seen as views, where the Processor is the user. The Data Operations system will handle any and all Processor requests in a simplified/abstracted manner and will handle any idiosyncrasies that a typical database has. The data operations sub-system is divided into seven primary functions. Food Operations, user Operations, workout Operations, stats Operations, and healthIndex Operations will accept a CRUD request from the processor.

The Driver Is in charge of interfacing with the database. This will allow us to modularize our architecture in such a way that if we ever change our database we only need to change this specific class while our entire system can stay as is. This is, in part, thanks to our MVC approach. The driver sub-system will contain the data abstraction layer for the communicating with the database via the PHP PDO library.

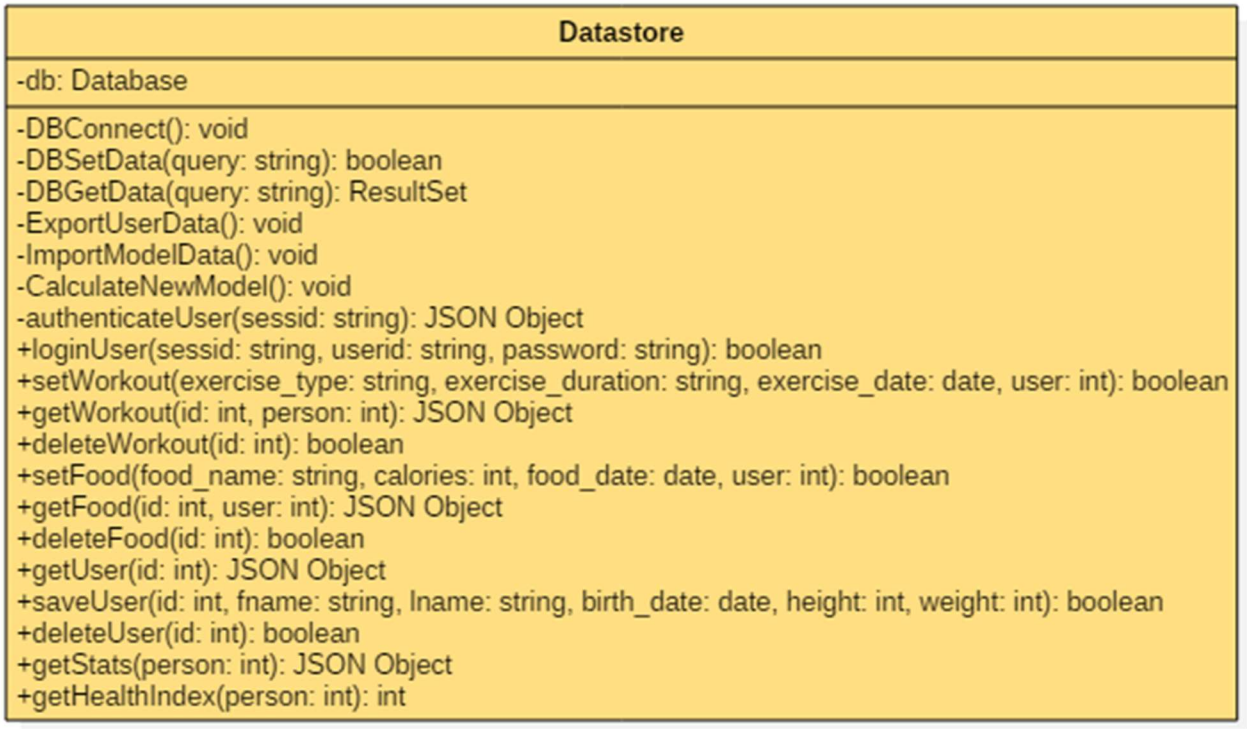


Figure 10-6 - Datastore Class Diagram

The figure above shows all methods from the data operations and driver sub-systems into a single class diagram.

10.1.5 Health Model

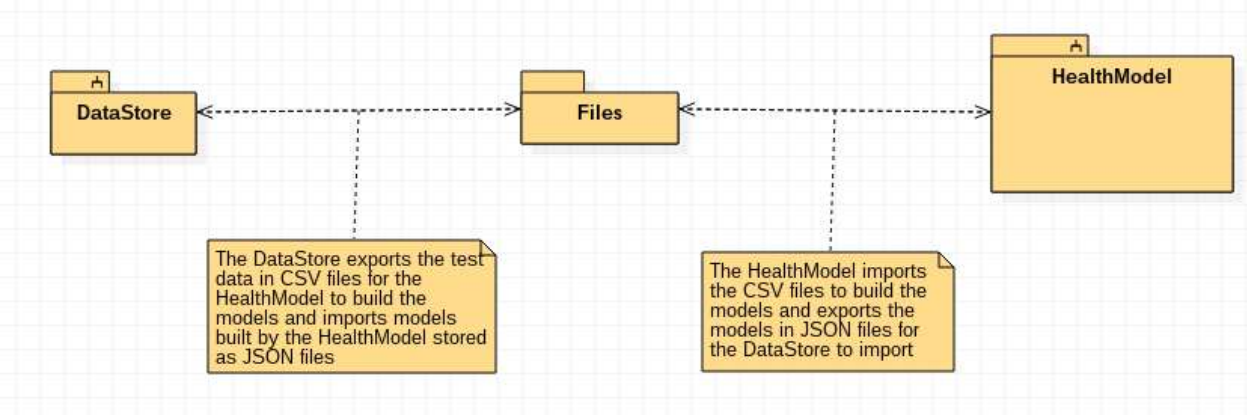


Figure 10-7 - Health Model Overview Diagram

The Health Model concept is one self-contained subsystem. While it contains objects of two other user-defined classes like the Regression Tree and Node, these classes are used only in the Health Model and do not independently exist to contribute to the model and do not interact with the other classes of the other classes. The Health Model communicates with the Datastore only. The implementation of this is discussed below in the section 12.1.5 below.

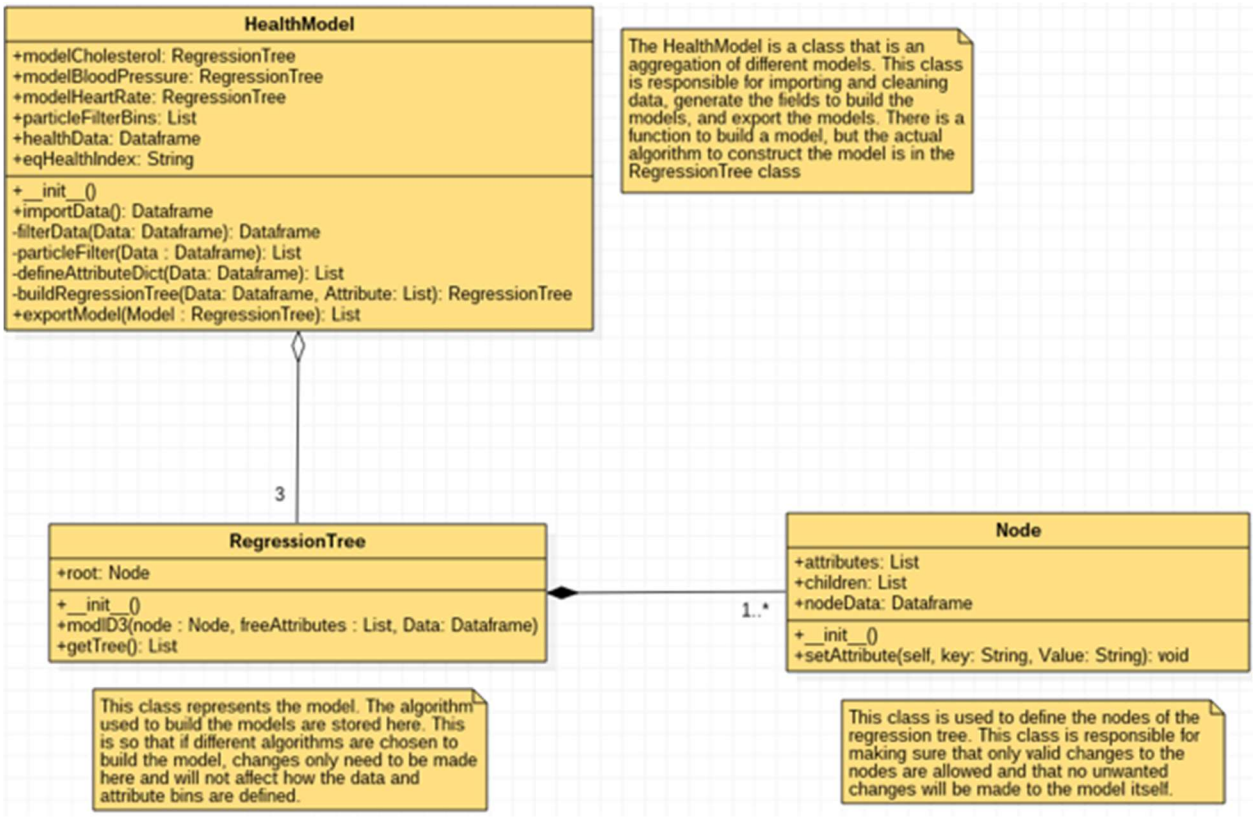


Figure 10-8 - Health Model Class Diagram

Figure 10-8 - Health Model Class Diagram above shows how the Health Model concept is broken into classes and how each class interacts. We will explain what each of the attributes and functions represent in section 10.2.5 below.

10.2 Attribute Definitions & Operation Signatures

This section will focus on explaining our approach for our classes that make up our subsystems and overall concepts.

10.2.1 Mobile Application

As seen in Figure 10-2 - Mobile Application Class Diagram, our Mobile Application system consists of 11 classes broken down into two systems the Controller system and the Services system.

First, the controller classes (denoted by the name format *Controller) are in charge of handling web events (i.e. onClick, onSubmit, etc.). We will have a controller class for each important view (i.e. HTML page). Breaking down the controllers into logical groups we get:

1. **User creation, authentication and management:** The controllers logInController, registerController, profileController will handle this by implementing functions as “logIn()”, “createAccount()” and “saveChanges()” respectively. These controllers will use the data entered in the HTML text fields, clean, filter and validate it and then send the data to the processor (via our services classes).
2. **Activity creating, viewing and management:** The controllers newActivityController and activityHomeController will handle these pieces of functionality; the earlier will implement “createWorkout()” and “createFood()” while the latter will implement “showWorkoutHistory()” and “showFoodHistory()”. Similarly, these functions will get user-entered data via the HTML fields and send them to our processor server via the service classes.
3. **Health Statistics:** Here the controller healthHomeController will be in charge of fetching data from the processor server, formatting it and passing it to the HTML views for display purposes. Using functions like “showAllStats()” or “showHealthIndex()” the user will select different views (via HTML buttons) and trigger these functions in the controller.

The Services system handles all data interactions with the Processor concept. All data interactions are done using RESTful services and similarly to our controller classes we broke them down into logical groupings we call resources, keep in mind the way our services interact with our processor server is via CRUD-like operation on our resources:

1. **User Resources:** The service classes userService and authenticationService will handle all user resource interactions. These classes will implement functions such as “getAll()”, “getById()”, “create()”, “logIn()” and “logOut()” respectively that will allos the controller classes to get these resources from our processor server.
2. **Activity Resources:** The classes workoutService and foodService will handle all of the activity-related functions in a similar fashion that we do on the user resources.
3. **Health Resources:** The class healthService will handle the fetching of all of the user-related health information. This will be done using functions like “getAllStats()”, “getHealthIndex()”, etc. which are self-explanatory. This resource is the only one that may not follow our CRUD-like pattern since it is technically not a resource in as

described by that pattern, which is why we may fallback to RESTful-style operation patterns.

10.2.2 Website

While the mobile app & website use the same frameworks, they will be packaged a bit differently due to their platform differences.

Each of the different objects in each section (Views, Controllers, Services) correspond to a different piece of functionality. The export controller and its own respective service and so on. While there may be more views than controllers, this does not present a problem to us as multiple views may use the same controller & service. They are grouped together due to their related functionality.

10.2.3 Processor

The main function of the Processor is to the handle interactions (data) initiated from the website and mobile application. It will interact with them via a RESTful-like API interface. The Processor will, for lack of a better word, process requests by fetching data from the Datastore and run specialized business logic to respond to the website or mobile application. It will be in charge of validating, filtering and cleaning information before storing it into the Datastore. The specific implementation and calculation is described below.

- Types:
    - **Datastore:** This is an array for storing the pointer of different Datastore in the system, and the value of the array is String.
    - **Community:** This is an array for storing the communities which would be called for trend, health information, and the value of array is String.
    - **Health information:** This is an array for storing the health information which would be used to displayed to user, and the value of array is String.
    - **Stuff:** This is an array for storing what user inputs into system which would be called for viewing, and the value of array is String.
    - **Stats:** This is an array for storing the results what are analyzed and calculated for displaying, and the value of array is String.
    - **User:** This is an array for storing the user's profiles, ID, and password, and the value of array is String.
    - **Workout:** This is an array for storing what user did in past, and comparing the different user's health data, and the value of array is String.

- Functions:
    - **GetXXInfo():** This function is to collect information from what user inputted, and calculate them, and store in Datastore.
    - **UpdataXXInfo():** This function depends on what user asked, and processor will retrieve data from Datastore, and updates them, and restore into Datastore.
    - **Filterdata():** This function is to delete irrelevant data from a bunch of data, and just keep the useful data. And depending on what user asked, sends the data to relevant processor unit to dealing.
    - **DrawChart/Table():** This function will analyze and calculate data from Datastore, and make data into visualization.
    - **DeleteXXInfo():** This function is to delete useless data, clean the Datastore.
    - **Analyzedata():** This function is to deal with relevant data from Datastore, and calculate it, display the result to user.

10.2.4 Datastore

See Figure 10-6 - Datastore Class Diagram for a list of these functions. The Datastore will accept commands from the processor. The processor will request data be fetched and retrieved from the database. The Datastore is the only system that interacts with the database directly

1. Driver functions
    1. Connect: A connection to the database is established.
    2. Query: If a SELECT query is run it will return a ResultSet
    3. Query: If an INSERT/UPDATE/DELETE query is run it will return a Boolean.
    4. authenicateUser: This authenticates the user based upon the authtoken.
    5. Typecast: There is a bug in PHP-PDO which causes integers to be cast as strings when read from a database. This function converts then back to integers/floats.

2. Data Operations functions
    1. “Food” functions: These functions perform queries involving food
    2. “Workout” functions: These functions perform queries involving workouts
    3. “User” functions: These functions perform queries involving the users.
    4. getAggregatedHealthStats: This pulls all health stats for a given state.
    5. getEthnicities: The model can only accept 7 possible values for the ethnicity attribute. This function with provide a front-end with those 7 possibilities.

6. getHealthStats: This will return the health stats from the health model based upon the features of a user (their height, weight, age, etc.).
7. ExportModel: Is run once per week and will create a JSON file to be imported by the health model.
8. ImportModel: Is run once per week after the health model creates a new model. The health model saves its results into a JSON file, which this function imports into the database.

10.2.5 Health Model

While the Health Model is the concept defined in the Domain Analysis and Interaction Diagrams, the implementation of the Health Model requires three classes: Health Model, Regression Tree and Node classes. The implementation and its motivation is described below and in section 12 below.

The Health Model class is that generates the models for predicting Cholesterol, Blood Pressure, Heart Rate and Health Index for an individual. The models used to predict Cholesterol, Blood Pressure and Heart Rate are instances of the Regression Tree class (which is discussed below), whereas the HealthIndex is an equation we developed to outperform the BMI. This class is responsible for generating the model and exporting so that it may be imported into the database. This critical point will be further discussed in the System Architecture section. The attributes for this class are:

a. **modelCholesterol:** This is a Regression Tree that stores the model which is used to predict the cholesterol of the user given the user's profile and exercise level.
b. **modelBloodPressure:** This is a Regression Tree that stores the model which is used to predict the blood pressure of the user given the user's profile and exercise level.
c. **modelHeartRate:** This is a Regression Tree that stores the model which is used to predict the blood pressure of the user given the user's profile and exercise level.
d. **particleFilterBins:** This is a list that contains the bins of all the relevant fields to the models. These bins for some attributes may be single discrete values (such as Male or Female) or they may be ranges (such as between 60kg -65kg in weight). The attributes have ranges such that any such attributes of a given participant of our sample model will have equal probability belong to the ranges of said attributes. For example, the age of participants are grouped into age groups such that the age of any given participant has equal probability of belonging to one of the determined age groups.

e. **healthData:** This is a data frame that contains the data used to generate the model
f. **eqHealthIndex:** This is a string that contains the equation used to calculate the Health Index. Unlike the health model, this is generated by us and is hard-coded into the class. This model is subject to change as the model will undergo refinement as our research and understanding of our problem domain matures.

The methods of this class are the set of methods that cleans, wrangles, and analyzes the data. They are given as the following:

a. **__init__:** This is the constructor of the class and must be present in any Python class
b. **importData:** This function is responsible for importing a .csv file which contains the participants used to generate the model. This function returns a Dataframe that will be used to generate the model
c. **filterData:** This function is responsible for cleaning the data so that only the relevant fields remain, and to transform the data so that it is more amenable to build the model. This function takes in the imported data frame and returns a cleaned data frame. The data will be filtered differently depending on whether we are using the data to predict cholesterol, blood pressure or heart rate.
d. **particleFilter:** This function takes in fields which take on many values are grouped into normalized bins such each bin contains an almost equal number of participants of each bin. The bins may be a slightly uneven distribution because the number of participants may not be evenly divisible by the number of bins.
e. **defineAttributeDict:** The particle filter function generates fields which the model uses to generate predictions. This function depends on the particleFilter function to transform attributes which take on many values and transform them into bins. This function returns a list of the different groupings of each attribute and is fed into the function that generates the Regression Tree
f. **buildRegression Tree:** This function is used to generate the model. The proper data frame and list of attributes are given as input. The output of this function is a regression tree and will be stored in one of our attributes.

The Regression Tree class is the class that is used to generate the regression tree. The Health Model class will use this class to generate the required models and to export the models. The Regression Tree is composed of nodes of class Node, described below. The attributes for this class are:

- **root:** this attribute is the root node of the tree of class Node

The methods of this class are:

- **__init__:** This is the constructor of the class and must be present in any Python class
- **modID3:** This function will generate the regression tree. The algorithm to this function is a modified version of ID3 classification algorithm. Instead of using the entropy of the dataset to make our decisions we aim to minimize the range of our predicted field in our dataset. When there is an attribute that minimizes the range, children to the node are created and this algorithm recourses. For each of those children, the data frame that fits all of the attribute choices and that of the children are passed onto the children, so as to ensure that each node can calculate the range. The recursive function ends when the range the there are no more attributes to consider or when there are no choices that minimize the ranges of the predicted values. This function takes in a node of class Node, a list which keeps track of free attributes (name freeAttributes), and a data frame that is passed to allow the formula to make decisions.
- **getTree:** This function returns the structure of the tree in the form of a list

The Node class defines the building block used in the class Regression Tree. This node is rather simple as it contains attributes and functions that are relevant only to a node. The attributes for this class are:

1. **attributes:** This is a list which keeps track of the attributes that have been assigned. It can be said that the attributes with nonempty values is the classification of that node. For example, if a node has an Age value set to the group 18-24 years old and the Gender set to Male, then this node contains a data frame which where the participants are of that age group and gender.
2. **children:** This is a list which keeps track of the children of a node. It is initialized to be empty, but children nodes can be appended to the list. This flexibility is required as different fields have different number of bins and we do not have any information on how the model will be build a priori.
3. **nodeData:** This is a data frame which contains participants that satisfy the nonempty fields in the attributes list of the same object

The functions of this class are:

1. **__init__:** This is the constructor of the class and must be present in any Python class
2. **setAttribute:** This function is used to set the value of an entry in the attributes list of a Node instance. This function prevents unwanted fields to be set in our attributes list and thus will be used set values to our different attributes in our attributes list.

10.3 Traceability Matrix

Table 10-1 - Class Traceability Matrix seen below shows how each class can be traced to one of our earlier defined concepts. We chose to approach our system this way since each concept has their own specific function within the system.

Table 10-1 - Class Traceability Matrix

Class	Mobile Application	Website	Processor	Datastore	Health Model
logInController	X				
registerController	X				
profileController	X				
newActivityController	X				
activityHomeController	X				
healthHomeController	X				
authenticationService	X				
userService	X				
workoutService	X				
foodService	X				
healthService	X				
SessionController		X			
HealthController		X			
ExportController		X			
ProfileController		X			
ActivityController		X			
SessionService		X			
ActivityService		X			
ProfileService		X			
HealthService		X			
ExportService		X			
HttpRequestService		X			
userProcessor			X		
healthInfoProcessor			X		
statsProcessor			X		
communityProcessor			X		
stuffProcessor			X		
workoutProcessor			X		
food Operations				X	
user Operations				X	
workOut Operations				X	

stats Operations				X	
healthIndex Operations				X	
ExportModel Operations				X	
ImportModel Operations				X	
Database Abstraction Layer				X	
Health Model					X
Regression Tree					X
Node					X

11 Design Pattern

When users require to show activity history, our website or mobile app would send a request to the processor, then the processor would make datastore connection, and datastore would connect to database to retrieve useful data. and the data would be sent to the processor, and the processor would analyze data, and after that, would draw an activity history table, then this table would be returned to the website or mobile app, and user can see the activity history.

11.1 Creational patterns

Abstract factory:

The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes. In normal usage, the client software creates a concrete implementation of the abstract factory and then uses the generic interface of the factory to create the concrete objects that are part of the theme. The client doesn't know (or care) which concrete objects it gets from each of these internal factories, since it uses only the generic interfaces of their products. This pattern separates the details of implementation of a set of objects from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface. We use this pattern in data collection, we create a class called Datastore to do the data storage work. Every time after system get data from user, no matter what kind of method we use, we call class Datastore to do the data storage job.

11.2 Structural patterns

Composite:

In software engineering, the composite pattern is a partitioning design pattern. The composite pattern describes that a group of objects is to be treated in the same way as a single instance of an object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly. We use JSON format information which contains many different data together. Then the composite pattern could extract JSON data and save all the relative data into a composite form variable.

11.3 Behavioral patterns

Command:

Command pattern encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations. In our project, we use this pattern in website & mobile app to control different requests from web side & mobile app and send to the datastore.

11.4 Concurrency patterns

Thread pool:

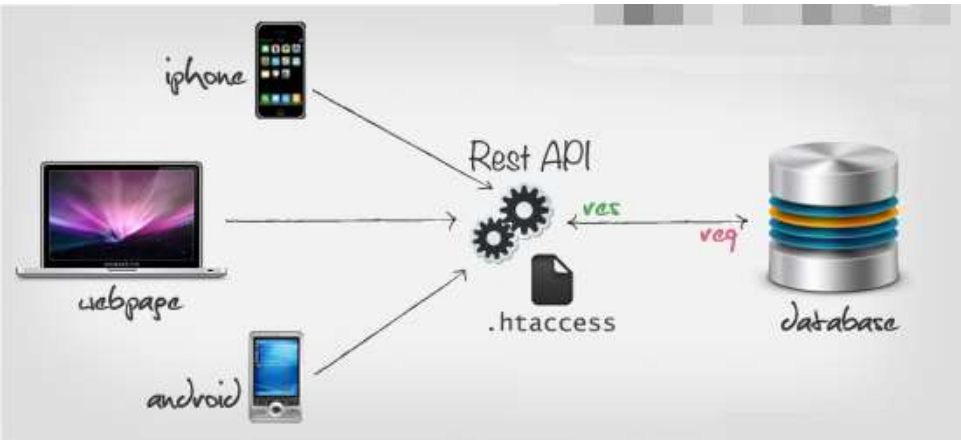
In computer programming, the thread pool pattern (also replicated workers or worker-crew model) is where a number of threads are created to perform a number of tasks, which are usually organized in a queue. The results from the tasks being executed might also be placed in a queue, or the tasks might return no result (for example, if the task is for animation). Typically, there are many more tasks than threads. As soon as a thread completes its task, it will request the next task from the queue until all tasks have been completed. The thread can then terminate, or sleep until there are new tasks available.

12 System Architecture and System Design

In order to implement and maintain a stable system, you must have an outline set of rules and agreements they govern how different parts of the system interact with each other. These guidelines along with a design blueprint help turn different unrelated parts into a cohesive system.

12.1 Architectural Styles

As we hinted at earlier, we will be using the Model-View-Controller architecture style for our system with the Processor also serving as a “Microservice”. This will allow us to group related functionality into different categories. The MVC approach not only gives us a much cleaner program architecture and layout, but gives us the opportunity to create common points for logic and data to flow through. Finally, it also allows us to separate our team into sub-groups with concrete responsibilities and ease the integration of our modules since MVC is well documented.



12.1.1 Mobile Application

As explained at the beginning of this chapter, our system will follow an MVC architectural style. Our views will be HTML-like pages that the user will interact with. Our model will be user data, activity data and health data maintained by the services we described in section 10.2.1 seen above. Finally, the controller portion of this architectural style will be handled by our aptly named “controllers”.

A final thought on this style, Android, Apple and other mobile devices also have the same idea of a view. When you are using an application on your device/platform of choice, you are interacting with the view, the part that you touch, type and swipe on.

12.1.2 Website

This subsection of the architecture style deals with the part of the code that display information back to the user. This will usually be the first/last point of travel for the data in its

journey within the system. It could be static pages that never change, or be the endpoint for data coming from the system.

The concept of a view can be different depending on the platform you are currently working on. If you are on a browser, the view can be thought of the HTML, JSP, PHP ...etc page that actually renders and displays the data. You are actually seeing the view layer when you read through a webpage.

12.1.3 Processor

As stated in section 10.2.3 above, the processor is a RESTful-like server that will act as a bridge between the website/mobile app and the Datastore and provide some specialized business logic.

What this means is that the processor will handle different instructions, when the processor gets requests from the website/mobile app, it will fetch data from our Datastore. After that, data will be analyzed, filtered and prepared in the processor, and the result will be sent back as a response (or in some cases stored back into our Datastore). Meanwhile, the Processor will provide data that our website can utilize to create visualizations.

12.1.4 Datastore

The data store is essentially an abstracted meta layer that 'handles' the interaction between all other classes and objects. It does not represent the data itself, except it represents the available interactions between the objects and the database. All objects, classes, services go through the data store to interact with the database.

The Datastore could be seen as a set of drivers for each specific language. Python will have its own specific driver/interface as opposed to PHP.

12.1.5 Health Model

The model is the layer of the MVC architecture that stands for & handles all of the domain & data specific logic for the system. The Health Model class is responsible for importing and cleaning data, normalizing the attributes and for generating the models. The Health Model class is composed of models, which are objects of type Regression Tree. Since the models for Cholesterol levels, Blood Pressure levels and Heart Rate are generated in the same manner, it

was natural to abstract the actual generation of the model to a separate class called Regression Tree to reduce duplicate code. This implementation also encapsulates the actual generation of the models and ensures no other objects can affect how the models are generated. By implementing the RegresssionTree class, we can also quickly change the classification algorithm quickly by changing only one function in the Regression Tree class. The Regression Tree class is composed of nodes, instances of Node class. This class contains information about the attributes which have been used to classify the tree. This class also contains a function that is responsible for setting only valid attributes to nonempty ranges. This added layer of abstraction prevents the classification algorithm to use attributes which have not been approved beforehand.

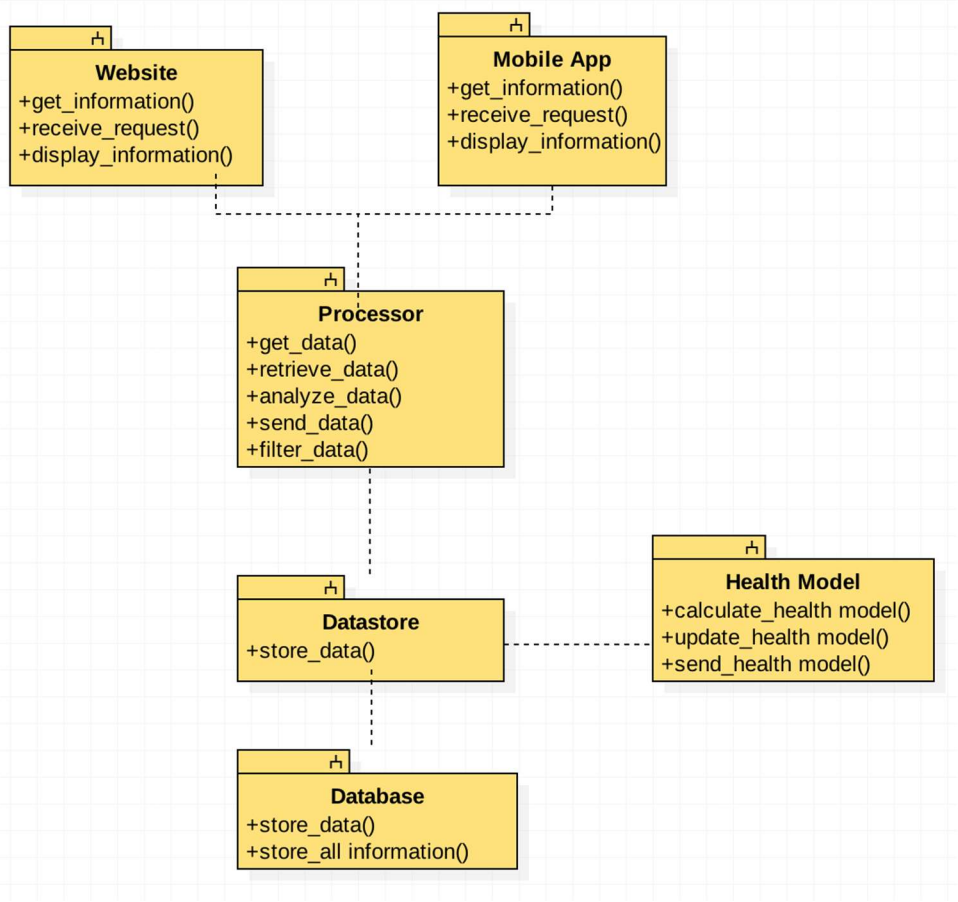
The Health Model is responsible only for generating the model, and will update the model on a weekly basis. The Datastore class is responsible for exporting the test data to a CSV file, which the Python implementation of the Health Model will use to generate Regression Trees (for Cholesterol, Blood Pressure and Heart Rate levels) or an equation (for the Health Index). The models are stored in the form of JSON files for the Datastore class to import and store on a table in the database.

The reason for this separation of responsibility of generating the model and querying the model is because the Datastore and Processor interaction is programmed in PHP, whereas the Health Model class is implemented in Python. While it is possible to implement the Health Model in PHP, Python contains many libraries (namely the SciPy stack) designed for data analysis. We also wanted to maintain simple and uniform implementation of the interface Processor and the Datastore.

While this increases the coupling between the Datastore and Health Model class, it also reduces the time needed to respond to predictions upon a user's request since the models themselves will be stored in the database as a separate table. This means that the model does not have to be generated every time a user queries data that requires out models. This is all possible because the implementation of our model is a Regression Tree, a type of classification tree. As such, a prediction is made by querying data of particular attributes; such a model can be equivalently represented as a sequence of if-else statements the database uses to respond the user-requested predictions. Our Health Index predictor is simply an equation that can easily be stored in the database and quickly be used to return the Health Index of the user.

12.2 Identifying Subsystems

In this section we will be explaining how our concepts are broken down into systems and subsystems as well as how these subsystems interact. Finally, we will explain our thought-process behind breaking our concepts in a specific way.



12.2.1 Mobile Application

Figure 10-1 - Mobile Application Overview Diagram shows the overview of the Mobile Application concept as well as the different systems and sub-systems that make-up this concept. It has been explained in other sections but in short, we will have three systems with about 11 defined sub-systems.

1. **Views:** This system is in charge of interfacing with the user, serving HTML-like template pages with all the information provided by the other two systems. For the sake of brevity, we did not include all of the views in this section, but you can assume that each user-interface page designed for our mobile application is a subsystem of the Views system.
2. **Controllers:** This system handles all events from the views, uses data entered by the user on HTML inputs (i.e. text fields, selection boxes, etc.) and translates them into system data that can be sent to the Processor via the services. The reverse of this is also a responsibility of the controllers, that is, to fetch data from the Processor (via the

services), translate these into data that can be displayed in HTML-like fields and passed them over to the views for display.

3. **Services:** The services are objects in-charge of interfacing the controller to the processor server via RESTful API calls and/or CRUD-like operation.

Finally, the way these systems refer to each other can be seen in the same figure while the subsystems connection can be seen in Figure 10-2 - Mobile Application Class Diagram above.

12.2.2 Website

Since our native application and website are built using the same technology (AngularJS/HTML & CSS) they have the same subsystems. While the application uses a custom framework to compile and the components into installable packages for each platform, the website uses the technology stack in its vanilla form. Please see section 5.2.1 for an overview of the MVC architecture which the website & mobile application use. We will still have systems for:

- Display(HTML)
    - We use Twitter Bootstrap as the main scaffolding for our website. It allows us to easily prototype and develop pages while not worrying about low level design elements.
- AngularJS will have some part of the display subsystem since it handles the routing & on screen elements.
- Functionality
    - We will be using JavaScript libraries for the functional part of our website. Vanilla JavaScript, jQuery, Bootstrap & AngularJS will provide us with the functionality needed.

12.2.3 Processor

The Processor subsystem in interacts with website/mobile app and Datastore. The website/mobile app send requests and then, depending on the object/request, will forward it to different functional units. After processing, processor will send result to Datastore to store, and send result to website/mobile app to display.

12.2.4 Datastore

The Datastore is divided into 2 subsystems to avoid coupling. The driver is the only system that contains the actual PHP-PDO sql commands. Should a different DBMS need to be used, then only this subsystem would need to be updated? The driver is simply a database abstraction layer

The data operations subsystem creates a *Database* object defined by the driver. Library-independent *Database* methods are used by the data operations subsystem. Example: the processer sends the command "getUser(12)". The data operations subsystem receives this command and does the following:

$db=new *Database*();

$db.prepare('SELECT * FROM users WHERE id=?');

$db.execute(id);

The driver implements the appropriate PHP-PDO commands to complete the query.

12.2.5 Health Model

Figure 10-7 - Health Model Overview Diagram shows how the Health Model subsystem interacts only with the Datastore subsystem via CSV and JSON files. The Datastore exports test data which the Health Model imports to generate the health models. Once the models are generated, the Health Model exports the model in JSON format so that the Datastore can import them and store the models onto the Database

12.3 Mapping Subsystems to Hardware

Mapping our systems and sub-systems to their hardware and/or software stacks could be dependent on a number of factors. It is important when generating a map of this to think of not only the current, but future needs of your system.

12.3.1 Mobile Application

The Mobile Application concept (and by extension all of its systems and sub-systems) will *have* to run in Android and iOS devices (specifically Honeycomb – 3.0+ for Android and iOS 9.0+ for iOS devices). This is because of non-functional system requirement REQ-34 which states the above.

As far as software stack/framework to implement these systems, we will be utilizing the Ionic Framework (aka Ionic) which is a framework that allows us to build a mobile application for multiple devices and operating systems using the same underlying code, specifically HTML, JavaScript, CSS and Angular JS. More information about this framework can be found in section **Error! Reference source not found.** (**Error! Reference source not found.**) seen at the end of this report.

12.3.2 Website

The website will follow much of the same sub-system attributes as the native platform applications. As It will still use the AngularJS framework, we will use the idea of a MVC architecture. The website is only dependent on the front end servers and to some degree, the client's hardware. We will support modern browsers and Internet Explorer 10+.

12.3.3 Processor

The Processor system and its subsystems will reside on the same physical server on which our Datastore system is. The reason behind this is to implement it in the same language (PHP) and with similar constraints (IIS Server vs. Apache Server, etc.) as the Datastore.

12.3.4 Datastore

The Datastore class will reside on the same physical server which supports our database, this server will most likely be an IIS server, but we will decide once we are further in implementation.

12.3.5 Health Model

The Health Model system and its subsystems will reside on the same physical server on which our database and Datastore system is.

12.4 Persistent Data Storage

The system uses the open source database management system MySQL. The database will initially be loaded with health related data from the CDC. This data is used to build the Model. The users will provide data that will be used to update the model on a regular basis. The health model will use JSON files saved on the hard drive. The JSON file will serve as intermediate between the health model and the database. The Datastore will periodically import data from the JSON file created by the health model and keep it in memory as it will be used for any health index queries and community index queries. This in-memory model will be updated as often as the health model creates a new JSON file. Finally, we discuss the format of this file in a later section.

12.5 Network Protocols

12.5.1 HTTP

HTTP(s) (Hypertext Transfer Protocol) is one of the most widely distributed applications built upon the Internet. It helps power webpages, links and transfers information from clients to servers. We will be using the HTTP protocol in multiple ways within our project. While we will be using it multiple ways, those pieces of functionality can be boiled down into a single type.

12.5.1.1 Communication between the Website/Mobile Application & Server

HTTP will not only help display and link together our website so that the client (user) can interface with it, it will be the transport mechanism for sending and receiving data within the Client-Server architecture of both the website and the mobile application. If a user wishes to view a certain page, HTTP will power that request. If that same user wishes to export their information in the CSV or JSON format, HTTP will facilitate the transfer. This is allowed through HTTP's many different actions or verbs.

HTTP won't only be used on the desktop/laptop browser platform. Our native applications will use the HTTP protocol to get information from our servers and it send it back to the servers when it is appropriate.

12.6 Global Control Flow

In this section we will cover some of our system's flow works. There is nothing special about the overall execution of our system with the exception of the Health Model concept which we will explain below in the Time Dependency Section.

12.6.1 Execution Order-ness

Because of the nature of web-based applications (which also spill over to mobile applications) our system is designed with an event-driven approach in mind. Even if initially users have to follow the same process (registration and logging in) everything else after those two steps can be done in the order users desire.

12.6.2 Time Dependency

The only time dependency this application has is in regards to the generating the models. The Datastore gathers data from the database and exports them in a CSV format. Then a Python script will run to generate the models from the exported dataset. Once the models generated, they are exported in the form of JSON files for the Datastore class to import and store into the database.

12.6.3 Concurrency

Similar to section 12.6.1 above, because of the web-based nature of our system it is designed so that each event our users generate will be handled by a processing thread of its own.

The only exception of this is, once again, the Health Model system since this is a scheduled process that runs once-a-week in a single-threaded fashion. I will process whatever new input data is given, calculate the new model and generate some output data that the Datastore system will then persist into the database.

12.7 Hardware Requirements

Just as we talked about mapping subsystems to hardware, actually generating the hardware requirements will be dependent on a number of factors. The requirements from a single small system will be vastly different than a system that handles multiple frontends & backbends while also being able to handle mobile platforms. If our application is small and only has a hundred or so users at a time, we could theatrically run everything off of one server, Frontend and backend. If our scale continues to increase:

- Multiple frontends and a single database.
- Even more frontends and multiple databases that are mirrored and sync'd up
- Android and/or Apple mobile phones
    - Apple: iOS 9.0+
    - Android: Honeycomb – 3.0+

This approach gives us flexibility in case of performance and redundancy requirements. We can spread out the user load over multiple machines and even have failures due to the database redundancy.

While this would solve the problem for the browser side requirements, we would still need a system for handling the mobile platform requests. We could theoretically have a set of servers that handle the requests from the mobile devices which then pass the information to the databases.

13 Algorithms and Data Structures

In this section we cover the algorithms used for creating our health model and how the data structures used to represent said model.

13.1 Algorithms

13.1.1 article Filtering and Correlated Stratified Monte Carlo Sampling

Particle filtering is a class of genetic-type Monte Carlo method used to generate the sample population. From a finite set of probability density functions we generate a sample that estimates the internal state of our dynamic population given our partial understanding of its characteristics. In this way we can generate a state's population given partial understanding of the citizens' properties. The particular particle filtering method employed is called the Sequential Importance Sampling, where we start with a set of samples (namely 1% of the population), and the percentiles of the characteristics given some aspect of the populations (e.g. the distribution of height against age).

The general methodology for one attribute is described as follows (it should be noted this sampling method can be generalized to N dimensions; in our sampling method, we used 10+ dimensions to characterize our population, depending on the sample we generated): Let there be a set of N samples, where each sample x is sampled to belong to particular weighting associated with the relative posterior probabilities (marginal distributions):

$$\{(w_k^{(i)}, x_k^{(i)}) \ : \ i \in \{1, \ldots, N\}\}.$$

Below is the proposal distribution (ideally the target distribution) which approximates our population, where x denotes one sample among the sample population and y denotes the characteristics which each sample can be attributed.

$$\pi(x_k | x_{0:k-1}, y_{0:k})$$

Since we are trying to approximate the target distribution {pi}, we will have to approximate it using our marginal distributions (p) in order to generate a sample. The formulation of the proposal distribution defined using the probability distribution is given below. The following equation is important as it is what allows us to approximate the target distribution given incomplete knowledge

$$\pi(x_k | x_{0:k-1}, y_{0:k}) = p(x_k | x_{k-1}, y_k) = \frac{p(y_k | x_k)}{\int \ p(y_k | x_k) p(x_k | x_{k-1}) dx_k} \ p(x_k | x_{k-1}).$$

One of the assumptions made in the above probability distributions is that the different marginal distributions (p) are continuous functions. Since we were only able to find cumulative distributions in the form of percentiles, we were forced to interpolate points between the percentiles. Given that only the values of the percentiles were given, we were restricted to using linear interpolation between points; any more advanced techniques of interpolation, like splines, requires information which we is unavailable. The formulation for linear interpolation is given below, where x_0, x_1, y_0, y_1, are the values of the percentiles which are given, and x, y are the values which we are interpolating:

$$y = y_0 + (y_1 - y_0) \frac{x - x_0}{x_1 - x_0}$$

One of the unique techniques which we integrated into our sampling method was the ability to sample the population in such a way that reflects the current observations of the population made by the various health communities. Since the population's health and physical attributes on a state level are not yet available open-sourced, we were forced to rely on national level information, which does not give use the state resolution we need, and state level percentiles, which do not give us the proper attribute-level resolution we need. In order to overcome this lack of information, we incorporated additional information that can improve the accuracy of our simulated population. Studies show that certain health statistic are correlated, sometimes strongly correlated with each other. Therefore, we coupled the random number generated with one statistic to another. For example, since there is a strong correlation between the Systolic and Diastolic blood pressures of an individual, we have defined one in terms of the other [http://www.ncbi.nlm.nih.gov/pubmed/18192832]. Other such correlations exist for the different components an individual's cholesterol [http://atvb.ahajournals.org/content/17/6/1114.full].

Since some of the attributes chosen can take on a range of values (e.g. An adult's age can range from 18 - 120), we used a method called particle filtering in order to generate a set of discrete bins that can be used by our model to make predictions. We use our sample data as the distribution of a particular characteristic of an individual. For example, given the set of participants found in our data, we use the distribution of the participants' ages as our sample distribution. One visualization of the Particle Filtering can be found below. Since there is no expectation that any given characteristic found in the data is uniform, we perform particle filtering for the attributes which take on a range of values. So, particle filtering will be applied to a characteristic like Age and will be grouped to generate Age Groups such that any participant in the CDC survey will have equal probability to fall into one of the bins.

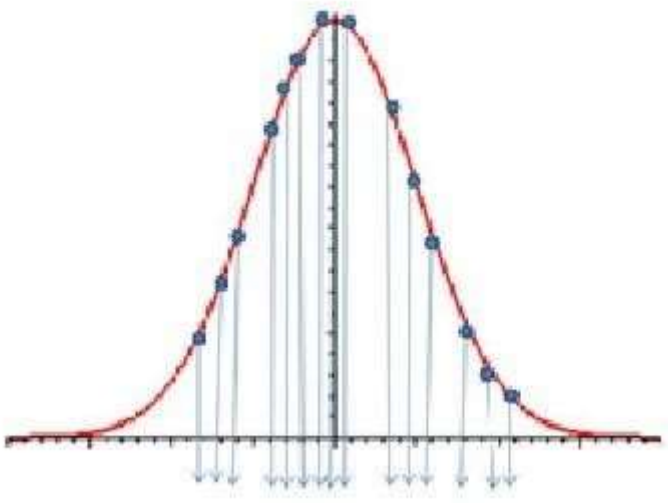


Figure 13-1 - Particle Filtering a Gaussian Distribution (Source: Virtual Labs)

Some modifications were made to our particle filtering method in order to resolve ambiguities in defining our discrete groupings for each attributes. Since many participants may all be the same age, many of our entries are duplicates of each other. This creates the awkward situation where a characteristic of an individual is of particular value that is at the edge of two bins. For example, there may be discrete Age Groups 18-22 years old and 22-28 years old. For individuals who are Age 22, there is an unclear method of classification. To avoid situations like such, we introduced 'noise' to the attributes where there are duplicates. The noise introduced is generated by a Pseudo-Random Number Generator to generate values in the range [0.0, 1.0).

The PRNG used is called the Mersenne Twister, where the period length of the sequence numbers is a Mersenne prime. The implementation used has a period of (2**19937) - 1, which is much greater than our sample size. In this way, we can be sure that the noise introduced each time will not be a repetition of the noise introduced prior.

Another modification made to the algorithm is the grouping of particular bins of particular characteristics. From our data we see that a significant portion people in our sample do not exercise regularly at all. When a particle filter is applied to exercise related attributes, the resulting filter contains many [0, 0] bins. To resolve the issue of multiple identical bins, we remove the [0, 0] bins and take the first bin of non-zero width and change the lower bound to zero. We account for this change in our algorithm by minimizing the weighted average of the ranges of the partitions created by each choice of a given attribute.

13.1.2 Decision Tree

The algorithm used to clean and gather the data is very straightforward. Since the data is gather from the CDC, it is structured in a form close to the one we desired. We first filtered any irrelevant entries (NULL, NaN, etc…) and separated the data based on the health statistic we chose to look at. This model known to be robust for being able to handle various types of input while still giving a good prediction rate. (The performance of our model in particular is detailed in the testing section).

All health statistics (Cholesterol, Blood Pressure, Heart Rate) are predicted using a Regression Tree. The algorithm used is a modification of the ID3 algorithm, commonly used to classify data sets. The ID3 algorithm is best for determining whether a given data point characterized by certain characteristics will fall into one of two results (e.g. Yes/No, Healthy/Unhealthy, etc…) Since the objective of our application is to be able to predict the health statistics of an individual up to a certain range, we needed to change the utility function to optimize. Instead of choosing the attribute where the partition of each choice maximizes the overall information gain, we chose the attribute where the partition of each choice minimizes the ranges of the health statistic. The model algorithm recurses until there are no other attributes available to further filter the information, or any partition of the data will only

increase the ranges of the health statistic. This modification of our decision function aims to generate the smallest prediction ranges for a given health statistic. Given the nature of our utility function, our algorithm is very susceptible to overfitting. It seems natural that splitting the data into smaller subsets leads to smaller ranges of data. To counteract this, we employ static thresholding so that a node with a certain number of participants are not split further. This is to ensure that the leaf nodes contain a statistically significant sample size (See Vassar Stats Reference). An activity diagram of this algorithm can be found the end of the Algorithms Section.

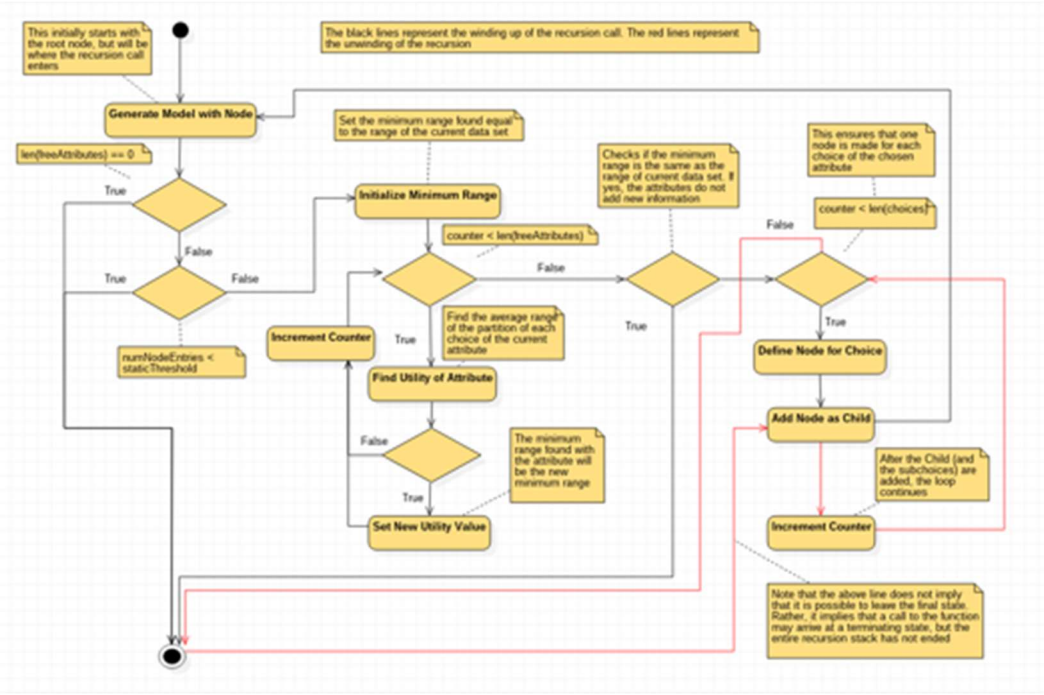


Figure 13-2 The Activity Diagram of the Decision Tree

13.2 Data Structures

13.2.1 Health Model

The main data structure used to implement the decision tree is a Trie. Since we do not know which node will most minimize the ranges of the partitions, we need a data structure which allows us to assign children as needed. A trie was chosen we needed the flexibility to dynamically assign children to a particular node. The nodes of the trie composed of a small hashtable (or dictionary in Python) used to store the characteristic and value that this node uses to filter the data of its parent node. The Trie structure also has the added advantage of being able to generate a JSON representation of the tree.

The Python Script used to generate the Health Model also generates JSON file to export the model that can be imported into and stored in the Database. This is possible because a root node can reach any leaf node by a particular sequence of if/else statements. This is equivalent of stating conditional statements in a query. So, the data structure used to a generate the JSON Tree is a dictionary which contains entries which stores the attribute and choice the node represents, as well as a list to store any children nodes this node may have. If the children list is empty, then this node is a leaf node.

13.2.2 Datastore

The Datastore will read the JSON file via a PHP script and create a JSON object. The Datastore will create an ID for each node (as well as capture the node ID of all children nodes iteratively). Storing it in the database will also any derived attributes to be calculated by the database and accessed via a view.

14 User Interface Design and Implementation

In the following section we will discuss how we implemented our different user interfaces. We will go in details about our frameworks and architectural models used.

14.1 Mobile Application

Initial concepts/mock-ups for the UI of our mobile application were good enough to cover all of the core use cases, specifically those that the mobile application was a participating actor. These are “UC1 – Sign up”, “UC2 – Log in”, “UC4 – Update information”, “UC5 – Add activity”, “UC6 – Show activity history”, “UC7 – View health information", and “UC10 – Predict health status”.

One major change was the decision to separate activities into discrete sections. We now have “Workouts” to record/track workout-type activities and “Foods” to record/track food consumption-type activities, but these views look very similar to the original generic “Activity” mock-ups. Aside from this, other changes were minimal; some fields were added/removed given our evolving understanding of the domain, and added some icons to help ease ambiguity in the sections.

Implementation was done using the “Ionic” framework. Since our expertise isn’t building native mobile-applications (especially for two different platforms) we decided to use a powerful HTML5 SDK that allows us to a build native-feeling mobile app using known web technologies (like HTML5, CSS3, JavaScript). Ionic also comes with CSS templates and icons that allow us to enhance the look of our mobile application with minimal work. It also uses the popular AngularJS framework which allows us to do powerful UI interactions as well as HTML templates fitting the “write-once” philosophy. Finally, Ionic uses the Apache Cordova mobile development framework which allows us to use these web technologies to build mobile application while avoiding each platform’s native language.

The end result of the UI implementation can be seen below. The implementation is still subject to change as our domain expertise increases.

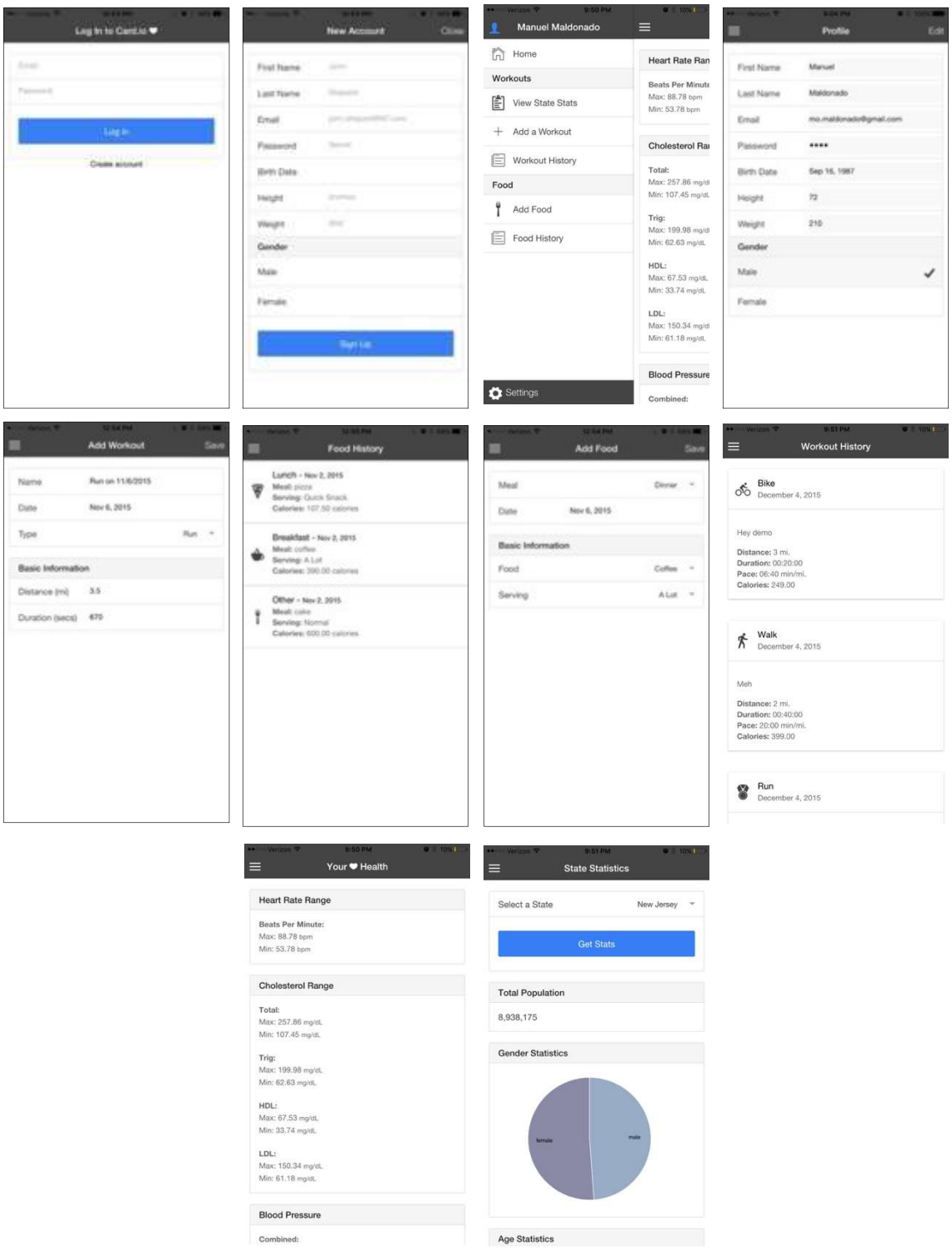


Figure 14-1 - Mobile Application Screens

Finally, we chose to implement the mobile application as an MVC system. Ionic helps us with building these “views” which reacts to the model (or data). AngularJS helps us build the rest by allowing us to create a model (from the data fetched from the server) that the user interacts with and modifies them (add new workouts for example) via controllers. The code for this implementation is provided but an example of this MVC architecture can be seen in Figure 14-2 - Mobile Application MVC Breakdown below where we follow a user trying to log into our mobile application. In the figure, a user can be seen initiating the “log in” action, the login page (our View) which will only handle data display and user input, all the data (our model) is sent to the “logInCtrl” (our Controller) where it will be cleaned, converted to JSON and sent via AJAX-like requests to our server. Once the response form the server is received in the controller, it is parsed and converted into a model that the view understands and sent back to the view where it will be displayed. This flow of data from user to view to controller to server where each piece has its own distinct role is the MVC architecture that we set out to implement.

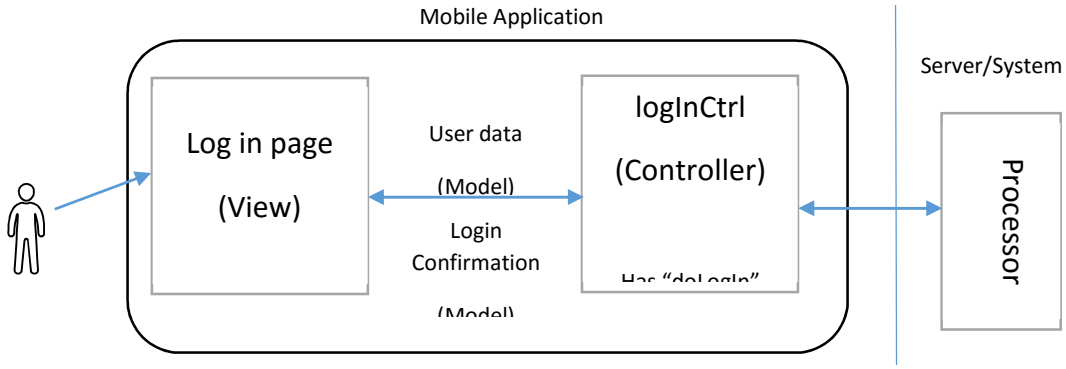


Figure 14-2 - Mobile Application MVC Breakdown

14.2 Website

For brevity, we have only added some of the web screenshots, please find the rest under the Website Screenshots folder.

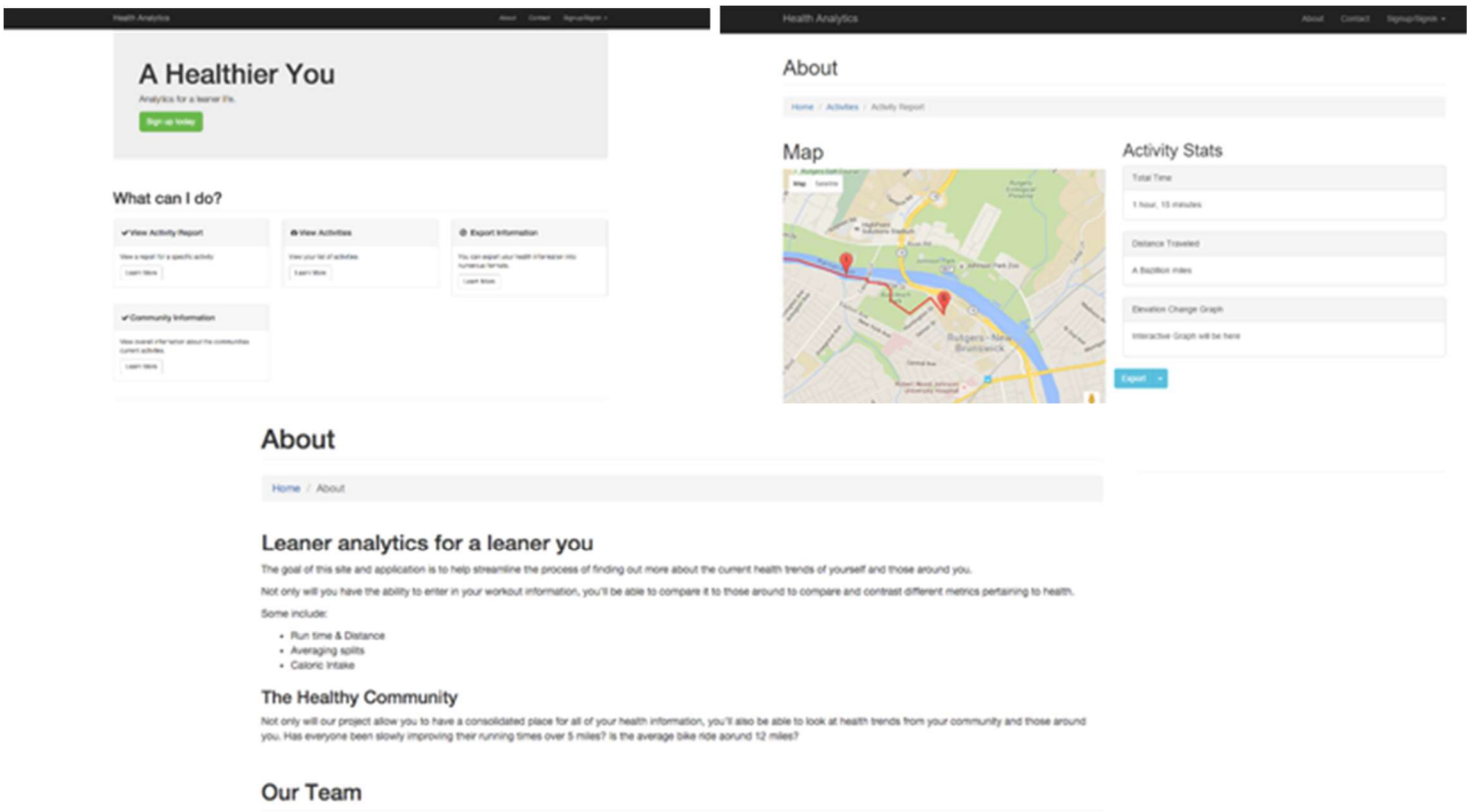


Figure 14-3 - Website Screens

15 Design of Tests

In this section we will address our approach to testing our individual parts of our system as. We will also hint at how we will test integration of the systems.

15.1 Testing Framework Ideas

Since we went with an MVC-Datastore framework, our unit tests were very easy to separate out into the different domains.

15.2 Mobile Application

Since our mobile application was implemented using the Ionic and AngularJS frameworks, we will be using rules of thumb and tools developed for unit testing applications written in these. We will use the Jasmine framework, which is the most popular framework for testing AngularJS apps, and Karma, which is a tool built by the Angular team for running unit tests. The former allows us to define test cases for the behavior of our AngularJS Controllers and Services and the latter will help with running the test cases in real browsers (including the iOS and Android browsers).

15.2.1 Unit Testing

Our approach for unit testing the mobile application will be show in two parts: first, for every AngularJS controller and service, we will have at least one test case that covers all of the functionality in them. Second, the UI will be tested with manual tests, doing a scripted flow with steps like: *register, log in, add new activities, view health information and log out*. This means that we will have 100% test coverage of the controller and services implemented but it may not be 100% coverage for our UI. We chose to not cover 100% of the UI testing because of time constraints. There are frameworks and tools that allow for automated UI testing (like Selenium, etc.) but we are not confident that we will have enough time to implement such tests.

Exploring the controller and services tests further, we will create two files, one for each controller/service. The files will contain all tests cases for that specific module and the format will be “<controller or service>.tests.js”. The format of the file as well as syntax can be found in the documentation for the Jasmine tool, we added a reference to it in our References section.

15.2.2 Integration Testing

For integration testing, the Processor concept designed will provide a list of interactions (API help document) which we will use to integrate and test. This API help document includes all signatures of all request, valid data and error codes. We included integration testing via the same tool we used for unit test cases, since we only needed to integrate our “services” with the Processor. Additionally, we used the same manual scripted test cases used by the UI testing since this goes over all the major functionality of our system and manually validate that the data entered (profile data, workouts, foods, etc.) still persists even after re-installing the application.

15.3 Website

In an effort to not duplicate the section above, a lot of the testing for the website will be shared with the mobile application for now. This is mainly due to the fact that most, if not all of the logic will be shared between the two.

15.3.1 Unit Testing

While the website & mobile application used the same framework and thus, the same testing framework, we need to change the tests to fulfill the functionality of the full website. All of tests at the services level will be the same and even could be its own test library since they will always be the same. The only ones that will change will be due to the changes within the view layer.

15.3.2 Integration Testing

The website will follow a similar approach for integration testing as the Mobile Application. Just as the unit testing is similar to the mobile application, so will the integration testing part.

One of the main pieces of functionality we will have to test is the idea of the responsive design. While not something we can just write a test script for, we will have to manually have users or developers view the website on devices of differing width and height. This will allow us to confirm that the responsive bootstrap framework and our added elements will still display properly no matter the dimension of the view. Since the positioning of the elements is paramount to a successful design, large design changes will have to be extensive tested on different devices of different sizes.

15.4 Processor

Like the Datastore, PHPUnit is used to perform unit tests. There are approximately 20 unit tests for the processor. Given the extensive testing of the datastore, the processor only tests each function specified in the API help one time. Since communicating properly with the processor requires an http connection, cURL functions will be used perform the unit tests.

15.4.1 Unit Testing & Integration Testing

A test for a successful connection to the API assesses that “200 OK” is returned by the server. Like the datastore, a successful unit test will assess that the appropriate JSON string is returned.

15.4.2 Incorrect data type

A unit test verifies the http error “400” is returned whenever invalid data is submitted.

15.4.3 Logic Errors

A unit test verifies the http error “404” is returned whenever a request is submitted with a function that does not exist in the API.

15.5 Datastore

Since the datastore is written in PHP, PHPUnit will used to perform unit test. The help for the API (http://www.rugatech.com/se1/api_help) provides a thorough description of all functions used the API (which in turn are used by both the processor and datastore). Approximately 80 automated unit tests where written for the datastore. These tests provide 100% test coverage.

15.5.1 Unit Testing & Integration Testing

A specific unit test was created to check if the connection function properly throws the exception for an invalid connection. This is accomplished by submitting a bad server name or incorrect password and evaluating that the exception return is correct. Most methods in the API has multiple unit tests (a few only have one). The unit test makes a connection to the datastore and submits a function to the datastore. If everything goes well, the unit test will assess that the returned value is the appropriate JSON string.

15.5.2 Incorrect data types

Most unit tests are validating that an error message is returned if invalid data is returned. In the "addUser" method, if the value "qwerty" is submitted for the weight variable and exception must be thrown indicating that the value must be an integer. Dozens of unit tests purposely submit invalid data and assess that the returned exception is correct.

15.5.3 Logic Errors

Email address is a primary key in the users table. A unit test must run to ensure that an exception is throw if an existing email is provided a second time.

15.6 Health Model

15.6.1 Unit Testing

The procedure to test our model is broken into two main parts. First, since we are working with national-level data in order to make predictions on populations for individual states, we must first generate sample state-level populations, where each sample contains statistics representative of the population of that particular state. For example, we want our simulated sample of inhabitants of New Jersey to be reflective of the physical characteristics and health of those inhabitants. Once we establish our sample population is representative of the actual population, we then analyze our simulated data with our Health Model.

Second, to benchmark the performance of our model, we will use the BMI as it is one of the most common health indicator used to judge an individual's cardiovascular health.

We will say that our model is able to make state-level predictions when both criteria are satisfied. Below is a more detailed description of the methodology of the two subtests.

First, we gather different distributions from various sources that altogether characterizes the population of each state. The sources include the CDC, The Kaiser Family Foundations, Census.gov and other health statistics and data repositories. Then from those probabilities distributions and from the geographical breakdown provided by the Census data, we can generate samples using Monte Carlo Sampling for each state so that the characteristics of the sample populations generated for each state closely mimics the characteristics of the population of the given state. We will also try to generate health statistics (Cholesterol, Blood Pressure, Heart Rate) based on distributions that we may find. This step may prove to be difficult as there are few sources which disclose such information. This hurdle, however, is expected as the US Government is only recently pushing for open sourced medical data to enhance personalized medical (See Article: http://www.phdish.com/blog/precisionmedicine). If

we cannot find state-level distributions of the health statistics mentioned above, we will use the Waist-to-Height Ratio of individuals from a national-level data set to simulate the health statistics of the sample population. The Waist-to-Height ratio has been found to be a better indicator of the BMI. To meet our first criterion, we will compare our simulated health statistic values with state-level distributions which detail the percentage of the population that have at-risk levels of cholesterol, blood pressure or heart rate. (The definitions for what at-risk health statistics can be found from sources like the American Heart Association.)

Second, we generate a Regression Tree for each health statistics for each state-level sample. For each Regression Tree we use our simulated sample in order to test the accuracy of our model. We then compare the accuracy and precision of the state level model to the accuracy and precision of the national level model. It should be noted that this test depends on whether our simulated samples are accurate. By passing this test, we will acknowledge that not only our simulated samples are accurate, but the models generated from our simulated sample can provide accurate predictions on a state-level.

15.6.2 Integration Testing

The integration test for the Health Model is simply to check if the Datastore class can important the JSON decision trees. The integration test for the Health Model and Datastore passes if we are able to specify a set of required physical characteristics of an individual to the Datastore, and the Datastore is able to return nonzero values, which indicates that the Datastore is able to read the JSON files and return the predicted values based on the attribute. The actual values which the Datastore returns will depend on the unit test of the Health Model, as that unit test tests the accuracy of the model.

16 History of Work

Health Model and User Generation

In regards to the Health Model, we were able to develop a set of models based of a simple Decision Tree algorithm to predict the health statistics of an individual. The model can be optimized so as to improve the accuracy and precision of the model. The User Generation algorithm was successful in generating state level samples that closely represent the actual statistics of the corresponding states. More importantly, we have proven it is possible to integrate results from recent scientific papers in order to enhance the sampling technique to generate even more accurate sample populations. By transitioning from national level data to

state level simulated data, we were able to maintain the same level of prediction accuracy while increasing the prediction range from [Min, Max] to [Mean – StD, Mean + StD].

17 Project Evolution and Thoughts

Our project seemed to have its hills and valleys of productivity. Just like an engineering project, there were parts that were wholly dependent on the finish of other parts. While this allowed us to plan ahead and figure out the foundation far in the future, it was very dependent on everyone's contributions.

One factor that would not have been present if this was done in the workforce was the fact that everyone's efforts would most likely be focused on this singular project instead of efforts being spread out between multiple classes and work.

18 Future Possibilities

18.1 Health Model and User Generation

While our model can integrate user inputs, we have not yet implemented this feature. However, we have given thought to how user information can be integrated into the model. As more users give their information, our Health Model will take sample with replacement the simulated data and user input information. In this way, we can slowly phase out our simulated data and use the user information as the sample data. Given that certain regions in the US are more technologically advanced than other regions, it is possible to accumulate user information from New York more quickly than Wyoming. We can take from surplus information from one state as user temporarily. The idea is that by taking surplus user information from various regions, we are able to average out the state-level influence. This is to mimic a nationally-averaged sample. The motivation for this is that we want to phase out our simulated sample as quickly as possible and use information from real users. Once a state has accumulated enough users, we will phase out the temporary substitute user information. This allows us to transition to state level samples generated from our user base.

In the future, we would like to simulate more accurate state populations and substitute the Decision Tree based algorithm to a more sophisticated neural network. We focused on Data Generation because the state level data required to generate models did not exist. Having provided a methodology for generating state samples of a particular quality, we will be able to use our simulated data as a base for building more sophisticated models. One of the main shortcomings of our Decision Tree algorithm is the need to generate a large number of models for each state and health statistic. In the future, we will use neural networks in order to integrate all the models together. This has the advantage of reducing the number of models

which we need to generate, as well as creating a unified model that can also predict health statistics on state level. This means that the unified model can make interstate predictions.

In this iteration, we did not implement a Health Index due to time constraints. In the future, we will implement the Health Index alongside the new model.

19 Current Status

So far, we have finished almost all the system feature, such as website, mobile app, health model, database, and improved our health model algorithm. Here are some outstanding accomplish that we achieved below:

1.Collecting information: We collect information though our website and mobile app, and these information was stored in our database.

2.Health Model: We have a prefect health model; it can give user health information to keep fit. And also can calculate a new model once a week.

3.Mobile APP: User can user our mobile app to connect our system, and input their information, and check other information.

4.Website: The function is same as mobile app.

PHP Server: PHP server that implements a REST(Representational State Transfer-like API system. This gives us all the data functionality (business logic) the mobile app and website need without having to put both the view of the data and the processing of the data in the same place.

20 Future Work

Limited by the time, we can't make everything perfect. The current system needs some improvement to meet more requirements from users.

Firstly, we do need more statistics about health from users and Local, and that would be make our health model more accurate.

In addition, we also need more users to use our website and mobile app, and we can get more useful information. We can give user a better experience.

We know there are some other improvements we should finish. We would try our best to make the system more attractive and useful.

21 References

NHANES 2011 - 2012: Demographic Variables and Sample Weights Data Documentation, Codebook, and Frequencies. 2015. NHANES 2011 - 2012: Demographic Variables and Sample Weights Data Documentation, Codebook, and Frequencies. [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2011-2012/DEMO_G.htm. [Accessed 17 October 2015].

NHANES 2011 - 2012: Blood Pressure Data Documentation, Codebook, and Frequencies . 2015. NHANES 2011 - 2012: Blood Pressure Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2011-2012/BPX_G.htm. [Accessed 17 October 2015].

NHANES 2011 - 2012: Body Measures Data Documentation, Codebook, and Frequencies . 2015. NHANES 2011 - 2012: Body Measures Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2011-2012/BMX_G.htm. [Accessed 17 October 2015].

NHANES 2011 - 2012: Cholesterol - HDL Data Documentation, Codebook, and Frequencies . 2015. NHANES 2011 - 2012: Cholesterol - HDL Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2011-2012/HDL_G.htm. [Accessed 17 October 2015].

NHANES 2011 - 2012: Physical Activity Data Documentation, Codebook, and Frequencies . 2015. NHANES 2011 - 2012: Physical Activity Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2011-2012/PAQ_G.htm. [Accessed 17 October 2015].

NHANES 2009 - 2010: Cholesterol - HDL Data Documentation, Codebook, and Frequencies . 2015. NHANES 2009 - 2010: Cholesterol - HDL Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2009-2010/HDL_F.htm. [Accessed 17 October 2015].

NHANES 2009 - 2010: Cholesterol - LDL & Triglycerides Data Documentation, Codebook, and Frequencies . 2015. NHANES 2009 - 2010: Cholesterol - LDL & Triglycerides Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2009-2010/TRIGLY_F.htm. [Accessed 17 October 2015].

NHANES 2007 - 2008: Blood Pressure Data Documentation, Codebook, and Frequencies . 2015. NHANES 2007 - 2008: Blood Pressure Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2007-2008/BPX_E.htm. [Accessed 17 October 2015].

NHANES 2007 - 2008: Body Measures Data Documentation, Codebook, and Frequencies . 2015. NHANES 2007 - 2008: Body Measures Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2007-2008/BMX_E.htm. [Accessed 17 October 2015].

NHANES 2007 - 2008: Cholesterol - HDL Data Documentation, Codebook, and Frequencies . 2015. NHANES 2007 - 2008: Cholesterol - HDL Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2007-2008/HDL_E.htm. [Accessed 17 October 2015].

NHANES 2007 - 2008: Cholesterol - LDL & Triglycerides Data Documentation, Codebook, and Frequencies . 2015. NHANES 2007 - 2008: Cholesterol - LDL & Triglycerides Data Documentation, Codebook, and Frequencies . [ONLINE] Available at: http://wwwn.cdc.gov/Nchs/Nhanes/2007-2008/TRIGLY_E.htm. [Accessed 17 October 2015].

Lemos da Luz, P, 2008. High Ratio of Triglycerides to HDL-Cholesterol Predicts Extensive Coronary Disease. US National Library of Medicine National Institutes of Health, [Online]. 63/4, 427-432. Available at: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2664115/ [Accessed 16 October 2015].

[This was used as background for more accurate indicators like the Triglycerides to HDL Cholesterol as an indicator to risks of high cholesterol]

Wikipedia, Model-view-controller architectural pattern. [ONLINE] Available at: https://en.wikipedia.org/wiki/Model–view–controller [Accessed: 22 October 2015].

Microsoft Developer Network, Model-View-Controller. [ONLINE] Available at: https://msdn.microsoft.com/en-us/library/ff649643.aspx?f=255&MSPPError=-2147217396. [Accessed: 22 October 2015].

Wikipedia, Database connection. [ONLINE] Available at: https://en.wikipedia.org/wiki/Database_connection. [Accessed: 23 October 2015].

StarUML 5.0 User Guide (Modeling with Sequence Diagram). 2015. StarUML 5.0 User Guide (Modeling with Sequence Diagram). [ONLINE] Available at: http://staruml.sourceforge.net/docs/user-guide%28en%29/ch05_3.html. [Accessed: 23 October 2015].

ProjectLibre, Project Management tool. [ONLINE] Available at: http://www.projectlibre.org/?_ga=1.215819359.958191163.1445567675. [Accessed: 20 October 2015].

Dr. Ivan Marsic, Teamwork and Project Management. [ONLINE] Available at: http://www.ece.rutgers.edu/~marsic/Teaching/SE1/projects.html#TEAMS. [Accessed: 23 October 2015].

Dr. Ivan Marsic, 2012. Software Engineering. 1st ed. Rutgers University: Rutgers University.

W3.org, RFC2616 Regarding HTTP. [ONLINE] Available at: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. [Accessed: 29 October 2015]

Wikipedia, MVC Definition and discussion. [ONLINE] Available at: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Accessed: 30 October 2015]

Wikipedia, Cohesion (computer science). [ONLINE] Available at: https://en.wikipedia.org/wiki/Cohesion_%28computer_science%29. [Accessed: 31 October 2015].

Wikipedia, Coupling (computer programming). [ONLINE] Available at: https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29. [Accessed: 31 October 2015].

W3.org, HTTP Methods definition. [ONLINE] Available at: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html [Accessed: 29 October 2015].

Staffordshire University, Information Services Academic Skills Know-how. [ONLINE] Available at: https://www.staffs.ac.uk/assets/harvard_quick_guide_tcm44-47797.pdf. [Accessed: 31 October 2015].

Wikipedia, Create, read, update and delete. [ONLINE] Available at: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete. [Accessed: 31 October 2015].

Ionic, Ionic Framework. [ONLINE] Available at: http://ionicframework.com. [Accessed: 31 October 2015].

Apache, Apache Cordova. [ONLINE] Available at: https://cordova.apache.org. [Accessed: 08 November 2015].

AngularJS, https://angularjs.org. [ONLINE] Available at: https://cordova.apache.org. [Accessed: 08 November 2015].

Wikipedia, AJAX. [ONLINE] Available at: https://en.wikipedia.org/wiki/Ajax_(programming). [Accessed: 08 November 2015].

Gone Hybrid, How To Write Automated Tests For Your Ionic App - Part 1. [ONLINE] Available at: http://gonehybrid.com/how-to-write-automated-tests-for-your-ionic-app-part-1/. [Accessed: 08 November 2015].

Gone Hybrid, How To Write Automated Tests For Your Ionic App - Part 2. [ONLINE] Available at: http://gonehybrid.com/how-to-write-automated-tests-for-your-ionic-app-part-2/. [Accessed: 08 November 2015].

Jasmine, Jasmine, Behavior-Driven JavaScript. [ONLINE] Available at: http://jasmine.GitHub.io. [Accessed: 08 November 2015].

Karma, Karma. [ONLINE] http://karma-runner.GitHub.io/0.13/index.html. [Accessed: 08 November 2015].

SeleniumHQ, Selenium. [ONLINE] http://www.seleniumhq.org. [Accessed: 08 November 2015].

PHPUnit – The PHP Testing Framework. 2015. PHPUnit – The PHP Testing Framework. [ONLINE] Available at: https://phpunit.de/. [Accessed 09 November 2015].

ID3 algorithm - Wikipedia, the free encyclopedia. 2015. ID3 algorithm - Wikipedia, the free encyclopedia. [ONLINE] Available at: http://en.wikipedia.org/wiki/ID3_algorithm. [Accessed 09 November 2015].

Information gain ratio - Wikipedia, the free encyclopedia. 2015. Information gain ratio - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Information_gain_ratio. [Accessed 09 November 2015].

Industrial And Engineering Applications Of Artificial Intelligence And ... - Google Books. 2015. Industrial And Engineering Applications Of Artificial Intelligence And ... - Google Books. [ONLINE] Available at: https://books.google.com/books?id=5fbyVju-LJ8C&pg=PA375&lpg=PA375&dq=static+thresholding+artificial+intelligence&source=bl&ots=ENXHyENR_C&sig=dJIu9smrDiR_-z1Tf7N8rN1-bYg&hl=en&sa=X&sqi=2&ved=0CCYQ6AEwAWoVChMIgKGcytSByQIVSVgmCh3g7gSL#v=onepage&q=static%20thresholding%20artificial%20intelligence&f=false. [Accessed 09 November 2015].

IEEE Xplore Abstract - A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. 2015. IEEE Xplore Abstract - A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. [ONLINE] Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=978374. [Accessed 09 November 2015].

Virtual Labs, Monte Carlo Localization, 2015. [ONLINE] Available at: http://cse17-iiith.virtual-labs.ac.in/montecarlolocalization/index.php?section=Theory. [Accessed 09 November 2015].

Mersenne Twister - Wikipedia, the free encyclopedia. 2015. Mersenne Twister - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Mersenne_Twister. [Accessed 09 November 2015].

9.6. random — Generate pseudo-random numbers — Python 2.7.10 documentation. 2015. 9.6. random — Generate pseudo-random numbers — Python 2.7.10 documentation. [ONLINE] Available at: https://docs.python.org/2/library/random.html. [Accessed 09 November 2015].

Trie - Wikipedia, the free encyclopedia. 2015. Trie - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Trie. [Accessed 09 November 2015].

JSON - Wikipedia, the free encyclopedia. 2015. JSON - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/JSON. [Accessed 09 November 2015].

Hash table - Wikipedia, the free encyclopedia. 2015. Hash table - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Hash_table. [Accessed 09 November 2015].

Activity diagram - Wikipedia, the free encyclopedia. 2015. Activity diagram - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Activity_diagram. [Accessed 09 November 2015].

UML activity diagram controls are activity nodes coordinating the flows between other nodes: initial node, flow final, activity final, decision, merge, fork, join.. 2015. UML activity diagram controls are activity nodes coordinating the flows between other nodes: initial node, flow final, activity final, decision, merge, fork, join.. [ONLINE] Available at: http://www.uml-diagrams.org/activity-diagrams-controls.html. [Accessed 09 November 2015].

COMP101 - REPETITION (RECURSION). 2015. COMP101 - REPETITION (RECURSION). [ONLINE] Available at: https://cgi.csc.liv.ac.uk/~frans/OldLectures/COMP101/week9/recursion.html. [Accessed 09 November 2015].

Ch4 Intro Statistical Significance. 2015. Ch4 Intro Statistical Significance. [ONLINE] Available at: http://vassarstats.net/textbook/ch4pt1.html. [Accessed 09 November 2015].

Monte Carlo method - Wikipedia, the free encyclopedia. 2015. Monte Carlo method - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Monte_Carlo_method. [Accessed 09 November 2015].

Harrison, R. L., 2010. Introduction To Monte Carlo Simulation. National Institute of Health, [Online]. 10.1063/1.3295638, 17-21. Available at: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2924739/pdf/nihms219206.pdf [Accessed 08 November 2015].

State Category | Health Status | The Henry J. Kaiser Family Foundation. 2015. State Category | Health Status | The Henry J. Kaiser Family Foundation. [ONLINE] Available at: http://kff.org/state-category/health-status/. [Accessed 09 November 2015].

PhDish/Seth Robey. 2015. A Presidential Push for Computer Age Medicine. [ONLINE] Available at: http://www.phdish.com/blog/precisionmedicine. [Accessed 09 November 15].

Assessing Your Weight and Health Risk. 2015. Assessing Your Weight and Health Risk. [ONLINE] Available at: http://www.nhlbi.nih.gov/health/educational/lose_wt/risk.htm#limitations. [Accessed 09 November 2015].

Waist-to-Height Ratio as a Predictor of Coronary Heart Disease among Women. 2015. Waist-to-Height Ratio as a Predictor of Coronary Heart Disease among Women. [ONLINE] Available at: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4012298/. [Accessed 09 November 2015].

Body mass index - Wikipedia, the free encyclopedia. 2015. Body mass index - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Body_mass_index. [Accessed 09 November 2015].

Pearson product-moment correlation coefficient - Wikipedia, the free encyclopedia. 2015. Pearson product-moment correlation coefficient - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient. [Accessed 09 November 2015].

Coefficient of determination - Wikipedia, the free encyclopedia. 2015. Coefficient of determination - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Coefficient_of_determination. [Accessed 09 November 2015].

Lipid Conversion Factors - Screening and Treatment of Subclinical Hypothyroidism or Hyperthyroidism - NCBI Bookshelf. 2015. Lipid Conversion Factors - Screening and Treatment of Subclinical Hypothyroidism or Hyperthyroidism - NCBI Bookshelf. [ONLINE] Available at: http://www.ncbi.nlm.nih.gov/books/NBK83505/. [Accessed 09 November 2015].

Hsiao, P.J., 2007. Significant correlations between severe fatty liver and risk factors for metabolic syndrome. Journal of Gastroenterology and Hepatology, 22, 2118-2123.

Rosmond, R., 1998. Endocrine and metabolic aberrations in men with abdominal obesity in relation to anxio-depressive infirmity. Metabolism, 47/10, 1187-1193.

Shekharappa , R. P., 2011. C orr elation betw een bod y mass inde x and car dio v ascular par amet ers in obese and non-obese in differ ent age gr oups. Int ernational Journal of Biological & Medical R esear ch , 2/2, 551-55.

Linear regression - Wikipedia, the free encyclopedia. 2015. Linear regression - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Linear_regression. [Accessed 09 November 2015].

Cross-validation (statistics) - Wikipedia, the free encyclopedia. 2015. Cross-validation (statistics) - Wikipedia, the free encyclopedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Cross-validation_%28statistics%29. [Accessed 09 November 2015].

PHP: cURL - Manual . 2015. PHP: cURL - Manual . [ONLINE] Available at: http://php.net/manual/en/book.curl.php. [Accessed 10 November 2015].

PHP.net, Server-PHP. [ONLINE] Available at: http://php.net/manual/en/features.commandline.webserver.php. [Accessed 09 November 2015].

Wikipedia, RESTful API. [ONLINE] Available at: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 09 November].

Wikipedia, Model-View-Controller. [ONLINE] Available at: https://en.wikipedia.org/wiki/Model–view–controller. [Accessed 09 November].