University of Nevada, Reno

HeartMate: A Competitive and Motivational Fitness Application for iOS Devices

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Science and Engineering

by

Marlon Daniel Chavez

Dr. Frederick C. Harris, Jr., Thesis Advisor

May, 2016

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

MARLON DANIEL CHAVEZ

Entitled

HeartMate: A Competitive and Motivational Fitness Application for iOS Devices

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Dr. Frederick C. Harris, Jr., Advisor

Dr. Sergiu M. Dascalu, Committee Member

Dr. Yantao Shen, Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2016

Abstract

Smartphones today are more advanced than they have ever been before with hardware sensors built into them to detect location, motion, as well ways to communicate with third party hardware through bluetooth, and the internet through cellular or wifi. With the introduction of Apple's App Store, and the Google Play store smartphone devices have been given features one would not think possible on a cell phone. One of the target categories for app developers is health, due to the sensors provided in current smartphones an application can be developed to track a user's health. The apps currently on the market motivate users through mostly goal based challenges between themselves or their friends. HeartMate is an iOS mobile application that utilizes the sensors in the iPhone, and the use of a Bluetooth LE connected heart rate monitor to create a new competitive, motivational, real time social experience using heart rates as a performance measure. HeartMate calculates and keeps track of target workout zones that are specific to the user's fitness level. HeartMate has a social component and can be used to challenge friends to a run by streaming their heart rate data in real time. HeartMate also features the ability to challenge past running workouts of friends or oneself. HeartMate fills the hole missing from motivational / competitive fitness applications on smartphones.

Dedication

For Michael Jackson, Prince and David Bowie.

Acknowledgments

I would like to thank Dr. Frederick C. Harris, Jr., Dr. Sergiu M. Dascalu, and Dr. Yantao Shen for being on my committee, with special thanks to Dr. Fred Harris for giving me the opportunity to do research in the HPCVIS lab, and especially allowing me to continue research in the HPCVIS lab as our research areas digressed.

I would like to thank my mother, father, and sister for their love and support, and pushing me to complete this chapter in my life. With special thanks to my sister, Raquel, for providing the heart rate monitors used in the development of this thesis.

I would like to thank Lane Lassiter, whom I consider my brother as I have known him my entire life, along with his family whom I consider to be my second home.

I would like to thank Melissa Aguilar for her love and support through this long distance journey.

I would also like to thank the people I have met in this chapter of my life that I will never forget, with special thanks to Cameron Rowe for helping me whenever I had questions and believing I could finish in time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile phones have been around since the early 1970s, and were even less mobile than they are today. Early on these devices were strictly used for making and receiving calls, until the Personal Digital Assistant (PDA) was introduced in the 1990s. The PDA was a mobile device that provided users with basic computing and was later combined with the mobile cell phone. In the early 2000s the term, "smartphone" was adopted to describe such a device, thus starting the smartphone era. Even then the technology was still in its infancy. The smartphone at the time was more targeted towards productivity, and was created for that reason.

The smartphone started to become more interesting in 2007, when Apple introduced the iPhone, being one of the first of its kind, and revolutionizing the smartphone market from that point forward. The following year, Apple introduced the App Store to the iPhone Operating System (iOS), and allowed third party developers to create applications (apps) for the device [48]. Google then released their competitor, the T-Mobile G1 with the Google developed mobile operating system (Android), soon after [24]. Android contained a competitor to the App Store called, Android Market, which also allowed for third party developers to create apps and distribute them. Later, Android Market was renamed to the Google Play Store.

With the introduction of these app stores on two of the most popular smartphone platforms, the capabilities of what mobile devices could do exploded, and in came specialized apps. Soon there were apps for everything that would take advantage of the mobile device's hardware. Apps began taking advantage of the smartphone's

accelerometer, gyroscope, GPS, wifi, cellular and bluetooth connectivity to create apps one would only dream of having on such a small mobile device. Due to the hardware and software incorporated into these devices it allowed for health and fitness tracking apps to gain popularity. Since the smartphone is a personal device that many keep in arms reach, there isn't a device quite like it in comparison, which makes it the perfect health and fitness tracker. This has led to a focus on health apps, and even Apple has created specific hardware and software to make it easier for developers to track health on the iPhone.

A variety of fitness tracking apps exist, and they all do much of the same, which is provide information on the duration, distance, and calories burned. They also provide capabilities to gather heart rate information and location data to map the route of a run. Then there are some that incorporate challenges through a social network aspect that allow other users to compete against each other, but in the sense of predefined challenges or goals during an exercise. There aren't any that allow for real time challenges or motivation to keep the user working at their hardest. This is where the use of a bluetooth heart rate monitor communicating with a networked connected device can create such an atmosphere. The iPhone platform provides the perfect development environment to improve upon these health tracking fitness apps by adding that real time motivational competition experience other apps do not offer. This paper presents HeartMate, an iOS app that creates a real time social, motivational, and competitive atmosphere by measuring performance through the use of a bluetooth connected heart rate monitor.

The rest of this thesis is organized in the following manner. Chapter 2 covers other related fitness apps, measuring of running performance, exercise physiology, development environment, hardware and libraries used. Chapter 3 discusses HeartMate's functional and non-functional requirements along with use case modeling, and a detailed design of the system architecture and other interactive components of HeartMate. Chapter 4 goes into detail of how HeartMate was developed. Chapter 5 presents the results of HeartMate through an application walkthrough of its

functionality. Chapter 6 compares features between HeartMate and other fitness applications mentioned in Chapter 2. Chapter 7 summarizes the work completed, as well as presents ideas for future work on HeartMate.

Chapter 2

Background and Related Work

2.1 Other Competitive / Motivational Workout Apps

2.1.1 Overview

With the introduction of the App Store and many health devices, there have been numerous fitness apps to have come to the market to track one's health [50]. These apps or devices track the usual duration, calories burned, and distance traveled, but some also include a way to measure or increase the user's performance during a strenuous activity in another form. To do this a game like experience is created either through a social experience, or in the form of challenge based goals, with some using a proprietary algorithm. This is known as gamification, which is the use of game elements in a non-gaming environment, in this case the environment is health and fitness [20]. Lately, there has been an increase of gamification in mobile fitness apps [33, 53]. There has also been research and development into fitness apps that could replace personal fitness trainers to provide the motivation needed without the expensive cost [32]. Studies have also been done to determine the effectiveness of gamification of mobile fitness apps and have concluded it helps users dissociate from exercise [25]. In order to justify a solution HeartMate provides, other competitive / motivational fitness apps will be discussed in this section focusing on their gamification, motivational, and competitive aspects.

2.1.2 Nike+

Nike+ is a motivational brand from Nike, and provides an entire product line, suite of apps, and hardware for athletes to measure and track their activity performance. Nike+ originally started off with the Nike+iPod Sport Kit, where a sensor that tracked your movement was placed into your shoe, and a wireless receiver plugged into an iPod device to communicate with the shoe sensor [43]. Similarly, they released the Nike FuelBand which was a wearable wristband that tracked how much one moved, and provided a "score" called, NikeFuel. The products mentioned earlier have been discontinued, and their replacements come in the form of mobile applications due to the increase of fitness hardware and sensors already included smartphones [29].

Nike+ Move

Since the Nike FuelBand has been discontinued its replacement comes in the form of the Nike+ Move app. Previously, a user would have to wear a Nike FuelBand in order for them to convert their movement into the form of NikeFuel. NikeFuel is Nike's way of universally measuring how much a person moves, to provide insight to their movement trends, and provide motivation to users to move more in comparison to their friends [36].

The Nike+ Move app is designed to track the user's movement continuously during the day, and increase the user's NikeFuel score. Essentially, this app acts as a glorified pedometer in the sense that instead of steps being shown as the performance measure, it is the user's NikeFuel score as depicted in Figure 2.1a. NikeFuel is then used as a motivator for the user by having them compete against themselves to try and beat their daily NikeFuel average. Nike+ Move, also allows a user to compare their NikeFuel score to the previous day or days throughout the week, as seen in Figure 2.1b. There is also a social network component to the app allowing users to add their friends to see how they compared during the week, which can be seen in Figure 2.2a. Figure 2.2b shows Nike+ Move is also capable of recording the location of where the user's move score was calculated by using location based sensor data

(a) NikeFuel score shown in the Nike+ Move app.

(b) Compare NikeFuel throughout the week.

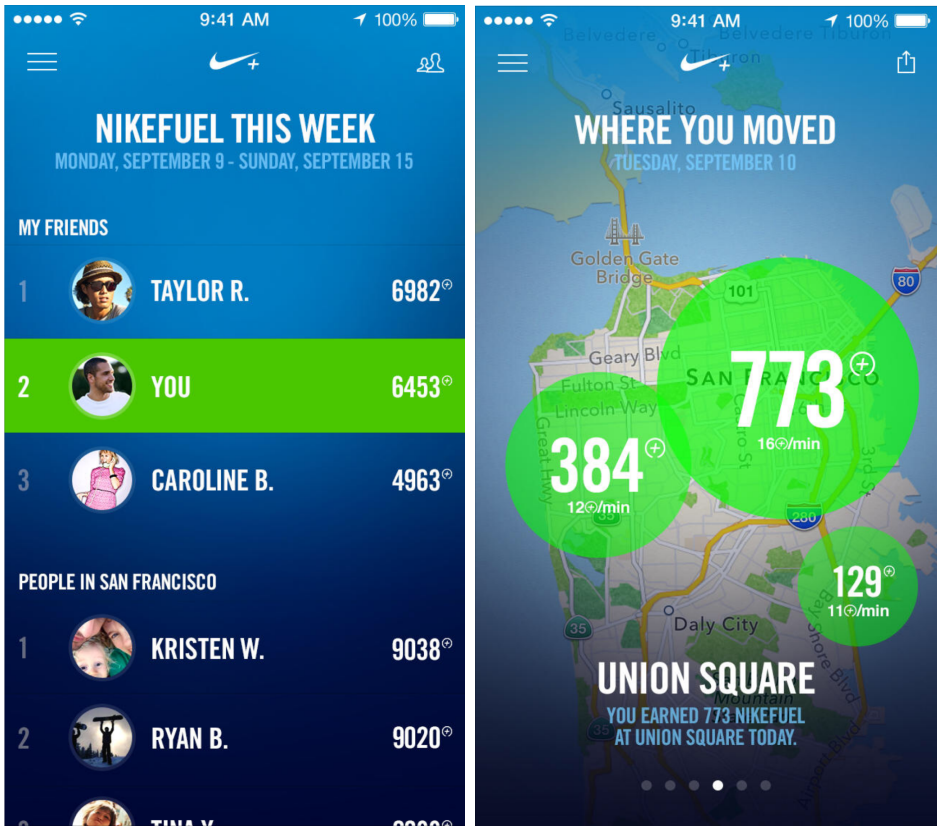
Figure 2.1: Nike+ Move app interface and features, from [37].

[37].

Nike+ Running

Nike+ Running is another app from the collection of the Nike+ brand that focuses on the fitness activity, running. This app still uses NikeFuel, but not in the way utilized by Nike+ Move. Nike+ Move uses it as a motivator, while Nike+ Running is using it similar to that of calories, it is just a quantitative measure of how much energy the user has expended. Nike+ Running calculates calories burned as well as the distance, map of a run, pace and heart rate (through an external heart rate monitor), some of which can be seen in Figure 2.3a.

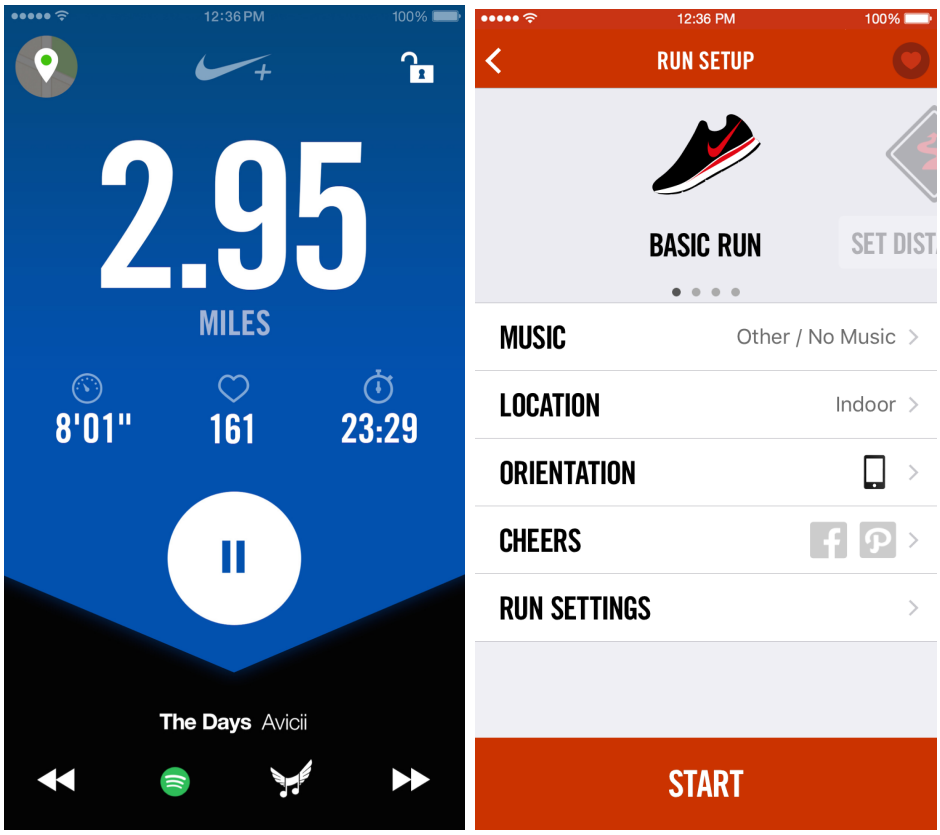
The way Nike+ Running motivates its users by allowing people to share the fact they are starting a run on a popular social network site. This can be seen in



(a) Compare NikeFuel score against friends.

(b) NikeFuel score calculated at specific locations.

Figure 2.2: Nike+ Move app interface and features (cont.), from [37].



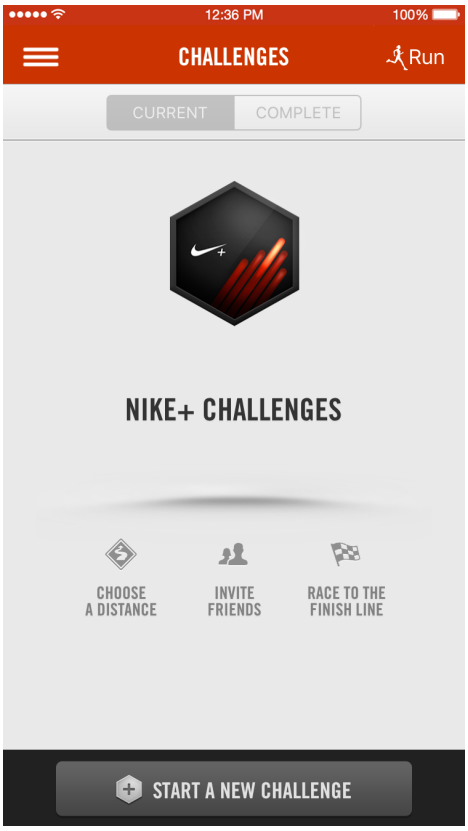
(a) Nike+ Running app tracking a run.

(b) Run setup using cheers as motivation.

Figure 2.3: Nike+ Running app interface and features, from [38].

Figure 2.3b, where a user sets up a running workout and can choose to get "cheers" from either Facebook or Path. Friends of the user running can then interact with the post, which then cheers the user currently running to provide motivation. Nike+ Running also has two different types challenges between friends, one of which is similar to that of Nike+ Move, but instead of comparisons of who has more NikeFuel at the end of the day, it is who has ran the most miles for the current month as shown in Figure 2.4a. The second type of challenge can be seen in Figure 2.4b, it is called, Nike+ Challenge, where a user can invite multiple friends to race to a predefined distance where the winner is the first to the finish line [38].

(a) Compare how many miles ran in comparison to friends.



(b) Challenge friends using Nike+ Challenges.

Figure 2.4: Nike+ Running app interface and features (cont.), from [38].

2.1.3 Fitbit

Fitbit is another popular health and fitness brand. They produce different types of health trackers that allow a user to track steps, distance, calories burned, sleep, weight and more. Fitbit also has an app with the same name that allows fitness tracking without the use of an activity tracker or with one paired to the app [23].

Fitbit App

Similarly to the previous apps mentioned, Fitbit can track a user's run or walk with pace, duration, distance, calories burned and heart rate (requires Fitbit tracker with heart rate support) which is shown in Figure 2.5a. The Fitbit app allows users to stay motivated and compete through the use of predefined goals, and Fitbit Challenges. The app has a social network component that can be used to share stats, and compete between friends through these goals or Fitbit Challenges. Similar to that of Nike+ Move, but instead of NikeFuel, the goals are centered around the number of steps a person takes rather than how much NikeFuel a user can generate. A list of predefined goals or Fitbit Challenges can be seen in Figure 2.5b. The Fitbit app uses notifications to provide a user with an update when they are nearing a goal or have met one [23].

2.2 Measuring of Running Performance / Exercise Physiology

2.2.1 Overview

Exercise physiology is the study of the acute responses and chronic adaptations to a wide-range of physical exercise conditions [41]. Studying these physical exercise conditions provide insight on how to measure a person's fitness performance. In order to create a competitive / motivational running fitness application, one must understand the basic components of measuring fitness to accurately calculate running performance. Specifically, measuring performance through a person's heart.

(a) The Fitbit app tracks daily fitness statistics.

(b) Fitbit Challenges and goals used to compete with oneself or friends.

Figure 2.5: Fitbit app interface and features, from [23].

2.2.2 Heart Rate

Heart rate is defined as the number of beats a heart can produce per minute (bpm) also known as a pulse. Many factors can stimulate a person's heart rate such as physical exercise, sleep, anxiety and stress. There are different contexts to a person's heart rate such as resting heart rate, max heart rate, heart rate reserve, and provide metrics such as calories a person has burned through physical activity. This section will discuss the aforementioned contexts a heart rate provides.

Resting Heart Rate

A person's resting heart rate (HR_{rest}) is the rate at which the heart can pump the least amount of blood needed when a person is at rest. At rest meaning when a person is in a calm state. A healthy person's resting heart rate can range anywhere from 50 to 100 beats per minute (bpm) and are indicators of their fitness level. The lower the resting heart rate, the better fit a person is and vice versa [11].

Max Heart Rate

Heart rate max (HR_{max}) refers to the maximum rate at which the heart can beat per minute. To accurately determine the maximal heart rate, a person must undergo a stress test that pushes them to the point where their heart rate can no longer increase, thus reaching their maximum heart rate. There are ways to estimate a person's maximum heart rate through a variety of formulas. A person's maximum heart rate decreases with aging [49].

The most widely used method of estimating a person's heart rate max is through a person's age. One of the most frequently widespread age-predicted formulas can be seen in Equation 2.1. Where the HR_{max} is equivalent to 220 minus the person's age in years. This calculation is independent of race, gender or age group [34].

$$HR_{max} = 220 - age\ (y) \tag{2.1}$$

Later evidence suggests the use of Equation 2.1 is bias in certain age groups [34, 46, 52]. Estimates of HR_{max} for men and women under the age of 40 years were overestimated, while providing underestimates for people older than 40 years. Equation 2.1 can be inaccurate to ± 10 *bpm*, and a newer equation, Equation 2.2, was created to fix such estimation problems. The Equation 2.2 fixes the issue of under and overestimation while still achieving a slightly better standard deviation of ± 5 to ± 8 *bpm* [34].

$$HR_{max} = 206.9 - 0.67 \times age\ (y) \tag{2.2}$$

Although, Equation 2.1 and Equation 2.2 are estimations of a person's HR_{max}, their error rate are still acceptable when determining the max heart rate of a person.

Heart Rate Reserve

Heart rate reserve (HRR) is the difference between HR_{max} and HR_{rest} as shown in Equation 2.3. HRR is used as a means to factor in a person's fitness level determined by their HR_{rest}, as previously mentioned. The more fit an individual is the lower their HR_{rest}, which increases HRR, and the less fit an individual is the lower their HRR. Thus, HRR will increase with increased fitness and vice versa [46].

$$HRR = HR_{max} - HR_{rest} \tag{2.3}$$

Calories Burned

In order for HeartMate to be a well rounded application a way to calculate energy expenditure must be discussed. Due to the nature of HeartMate using a heart rate monitor, the method used is known as a heart rate based method. A Calorie is 4.184 kilojoules (kJ) and was originally used to determine the amount of energy or heat that was required to heat water from $0^{\circ}\ C$ to $1^{\circ}\ C$ which happened to be 4.184 kJ [28]. In the sense of the human body, during exercise the body needs fuel to continue

producing meaningful work, Calories. Using age, gender (years), weight (kg), average heart rate (bpm) and time in hours, energy expenditure (calories) can be estimated through the use of a heart rate based calorie formula. The formula for each gender male and female can be seen in Equation 2.4 and Equation 2.5 [31].

$$Cal_{male} = \frac{(-55.0969 + 0.6309 \times HR_{avg} + 0.1988 \times W + 0.2017 \times A)}{4.184\ kJ} \times 60 \times T \quad (2.4)$$

$$Cal_{female} = \frac{(-20.4022 + 0.4472 \times HR_{avg} - 0.1263 \times W + 0.074 \times A)}{4.184\ kJ} \times 60 \times T \quad (2.5)$$

Where HR_{avg} is average heart rate, W is weight in kilograms, A is age in years, and T is time of exercise in hours. The coefficients on weight, age, and gender were produced in estimating the slope, and the intercept of the regression line between energy expenditure and heart rate from [31].

2.2.3 Target Training Zones

Training zones provide a way to measure an individual's workout intensity, in the case of HeartMate, the only workout is running. Each zone or intensity is aimed to stress the body in different ways and provide a measurement of how much work an individual is exerting. There are different types of methods to calculate these training zones and this section will focus on how to define these intensity zones through the use of our knowledge of heart rates. There is another method that defines target training zones as a percentage of $VO_2\ max$, but during this discussion one will see how the method used with heart rates will directly correlate with $VO_2\ Reserve$ and can be used interchangeably [12].

VO_2 Max and VO_2 Reserve

When an individual increases their level of exercise intensity there is an increased amount of oxygen being consumed in order to deliver the necessary amounts of oxygen

needed to sustain such level of intensity, known as volume of oxygen consumed (VO_2) . Similar to HR_{max} there is a maximal amount of oxygen consumption that can be reached during exercise known as $VO_2\ max$. $VO_2\ max$ is considered an individual's peak performance output. Measuring $VO_2\ max$ can accurately be done through exercise in a laboratory such as on a treadmill or stationary bike where the air exhaled is obtained and analyzed. The sample analyzed is measured for carbon dioxide which provides a measurement of how much oxygen the individual has burned [12]. Since the laboratory equipment is not easily accessible as are heart rate monitors, there are complicated methods to predict $VO_2\ max$, but a much simpler way is through the use of correlation. Research has previously shown that $\%HR_{max}$ is correlated to $\%VO_2\ max$, but it has been disproved and $\%HRR$ is more correlated to the percentage of VO_2 reserve ($\%VO_2R$). VO_2R is the difference between $VO_2\ max$ and volume of oxygen consumption at rest (VO_{2rest}), where VO_{2rest} is 3.5 $mL/kg/min$ as shown in Equation 2.6 [44, 45].

$$VO_2R = VO_2\ max - VO_{2rest} \tag{2.6}$$

Karvonen Method

The Karvonen method is an efficient method in calculating a target training threshold. Since fitness levels vary from person to person, the Karvonen formula factors in fitness level to produce a more effective training regimen specific to the individual. Instead of training at a $\%HR_{max}$, the Karvonen method uses $\%HRR$ and instead calculates a heart rate training threshold ($HR_{threshold}$) as the sum of $\%HRR$ and HR_{rest} as shown in Equation 2.7. Where I is the intensity percentage of the desired training threshold [34].

$$HR_{threshold} = HR_{rest} + I(HRR) \tag{2.7}$$

The result of Equation 2.7 produces a target heart rate value not only based off

Table 2.1: Target Heart Zones, from [12]

Target Zone	Heart Rate Reserve (%)
Low (Recovery)	60 - 70
Moderate (Aerobic)	70 - 80
High (Anaerobic)	80 - 90
Red Line ($\%VO_2R$)	90 - 100

the $\%HRR$, but one that also takes into account $\%VO_2R$ as they are correlated. The advantage to using the Karvonen method is that it places individuals of all fitness types at intensities relative above their resting state, thus providing an accurate way to compare performance [44].

Heart Rate Zones

Since the method of calculating heart rate thresholds have been discussed there needs to be a discussion of the different target zones. A list of zones along with their corresponding $\%HRR$ percentage values can be seen in Table 2.1. Low or recovery zone is often used to improve the muscles' utilization of oxygen and is considered more of a warm up zone. The aerobic or moderate zone increases the amount of oxygenated blood to muscles improving muscle strength. Anaerobic or High zone, is one where the body produces more lactic acid than it can remove, lactic acid is a byproduct of muscle during exercise that causes muscle fatigue. Red Line or $\%VO_2R$, should only be reached for short periods of time as lactic acid is developing at a high rate, and muscles start needing more oxygen than the body can provide [12, 17].

2.3 Development Environment

2.3.1 Overview

Given that iOS is the target development platform there are specific tools that are required in order to develop applications for iOS. Some third party tools are also used to assist in adding libraries to an iOS project. In this section these tools will be discussed along with the architecture and features of the target platform, iOS.

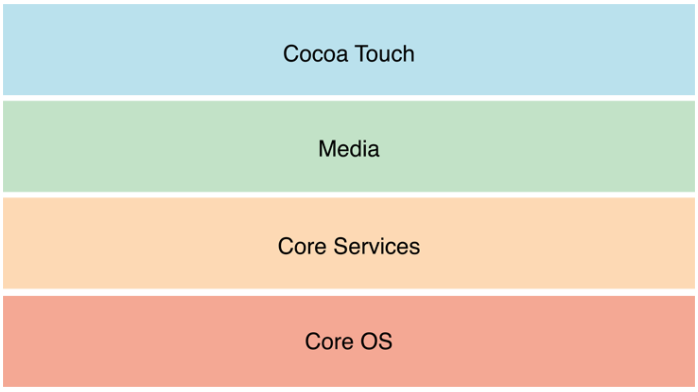


Figure 2.6: Layers of iOS, from [1].

2.3.2 iOS

Architecture

Understanding the way iOS is constructed provides insight on how to design an application for the operating system. The iOS architecture is designed in layers in which these layers act as an intermediary between the underlying hardware. These layers are Cocoa Touch, Media, Core Services, and Core OS which can be seen in their respective order in Figure 2.6. These layers allow apps to indirectly communicate with the hardware, and provide system interfaces that allow apps to be written in such a way that devices are still compatible even with different hardware capabilities. The layers closer to the top are utilized through higher-level frameworks that provide object-oriented abstractions to the hardware, where as the bottom layers use lower-level frameworks. It is recommended by Apple, to use the higher-level frameworks, but does not mean the lower-level layers are out of the question, and could be used when certain hardware features aren't available in the higher layers [1].

The three layers that should be focused on are the Cocoa Touch, Media, and Core Services layers. Cocoa Touch provides high-level features in creating the interface, handling Apple Push Notification Service, and frameworks to user interface elements (UIKit), a map interface (MapKit), and more. The Media layer provides capabilities such as graphics, audio, and video technologies. The graphics portion allows custom art, images, graphics, and animation be displayed on screen that work seamlessly with

the UIKit framework in the Cocoa Touch layer. Audio and video frameworks allow interaction with the underlying hardware to incorporate audio and visual into an app. The Core Services layer provides a closer interface to the hardware than any of the aforementioned layers. Core Services provide frameworks to store data (Core Data), access location data (Core Location), access and store health information (Health Kit) and many more [1].

Health

Baked into iOS is the Health app, which provides a centralized hub for storing and viewing health data. The Health app provides a dashboard of the user's health information stored, and gathered from the iPhone hardware or through a third party application. Developers can create apps that utilize this centralized health hub to provide or access data to or from the Health app using Apple's Health Kit framework. Developers can also access health data from compatible Bluetooth LE devices through Health Kit [7].

Apple Push Notification Service

The Apple Push Notification Service (APNs) is a service that allows developers to send remote notifications to iOS devices using a provider (server). Each device is connected to APNs, and will receive any notifications sent through the connection by APNs. A notification can be sent while the app is running or not, in the case that the app is not running then the user is notified. A developer has to create its own provider (server) for their client app, in which the provider creates the payload notification bundled with the user's unique device token, token generated by the user's device, and sends it to the APNs. Once APNs receives the payload notification along with the device token, APNs handles the delivery of the notification to the client app installed on the user's device [3]. An example of this can be seen in Figure 2.7

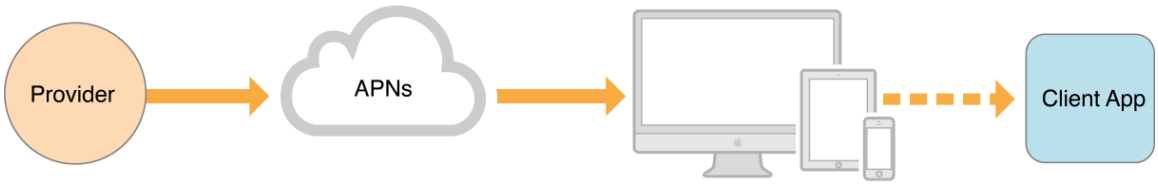


Figure 2.7: Apple Push Notification Service sending a push notification received from the provider directly to the device and client app, from [3].

2.3.3 Xcode

To create an iOS app there are two major development environments, the Xcode IDE which is developed and maintained by Apple, and AppCode which is developed and maintained by JetBrains. Xcode was chosen for this project as it is developed and maintained by Apple as is iOS. There are two parts to Xcode, the IDE, and Interface Builder.

IDE

Xcode is an integrated development environment (IDE) that consists of a code editor, compiler, debugger, and an interface builder. Xcode's interface or workspace is separated into four different areas: navigator, editor, debug, and utility areas. The Xcode workspace can be seen in Figure 2.8. The navigator area allows for easy navigation of all the files, folders, assets and frameworks, and any warnings or errors that may have been generated. During debugging it also provide the program's memory, CPU usage, disk, and network access, as well as breakpoint navigation. The main focal point in Xcode is the editor area, this area displays code and provides syntax highlighting, and code completion. The debug area is separated into two sections, the first section shows a list of objects in memory and their properties during runtime, and the second section is the console log. The utility area provides ways to inspect files, edit attributes of objects on the screen, connections of interface elements to code can be viewed here, code snippets and assets can be accessed here as well [39].

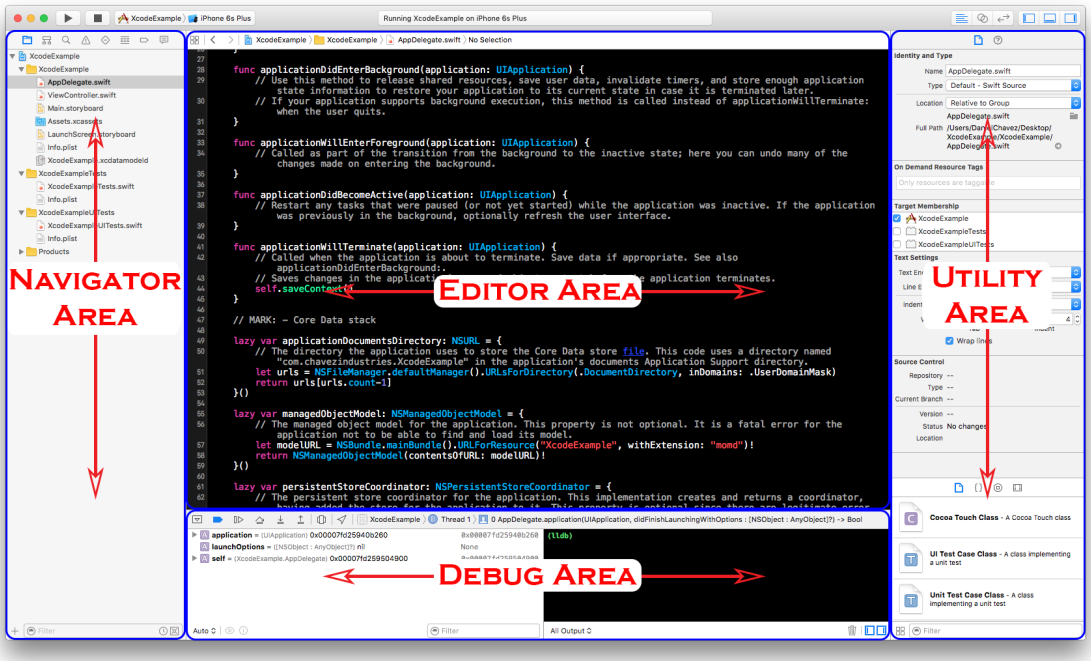


Figure 2.8: The Xcode workspace.

Interface Builder

Xcode offers another important tool called, Interface Builder (IB). Interface Builder allows a developer to easily design and create their interface for an iOS app without having to write a line of code [39]. The layout is similar to that of Figure 2.8, but now the interface canvas takes the place of the editor area, with the left panel showing the view hierarchy, and the right panel being the utility area, which now contains user interface elements that can be added to the view in the center of the canvas. The Interface Builder workspace can be seen in Figure 2.9. Connecting user interface elements to code is as simple as switching to a side by side view of the editor area, and Interface Builder canvas. A user can than click and drag on a button placed within the view, and connect it directly to the code by making a connection with the mouse as shown in Figure 2.10.

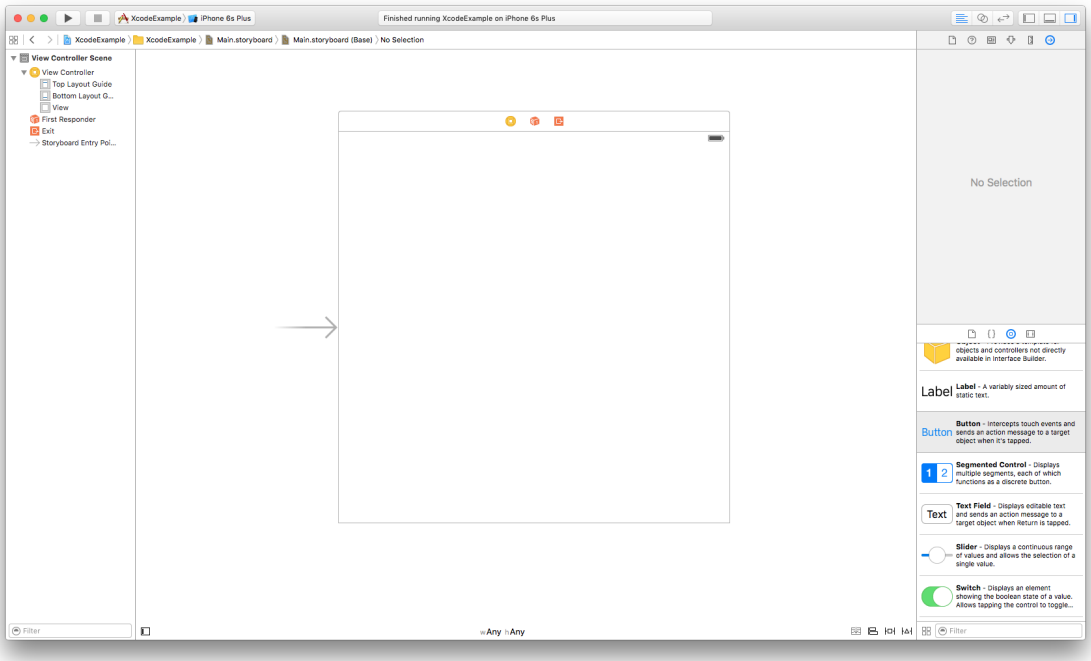


Figure 2.9: The Xcode Interface Builder workspace.

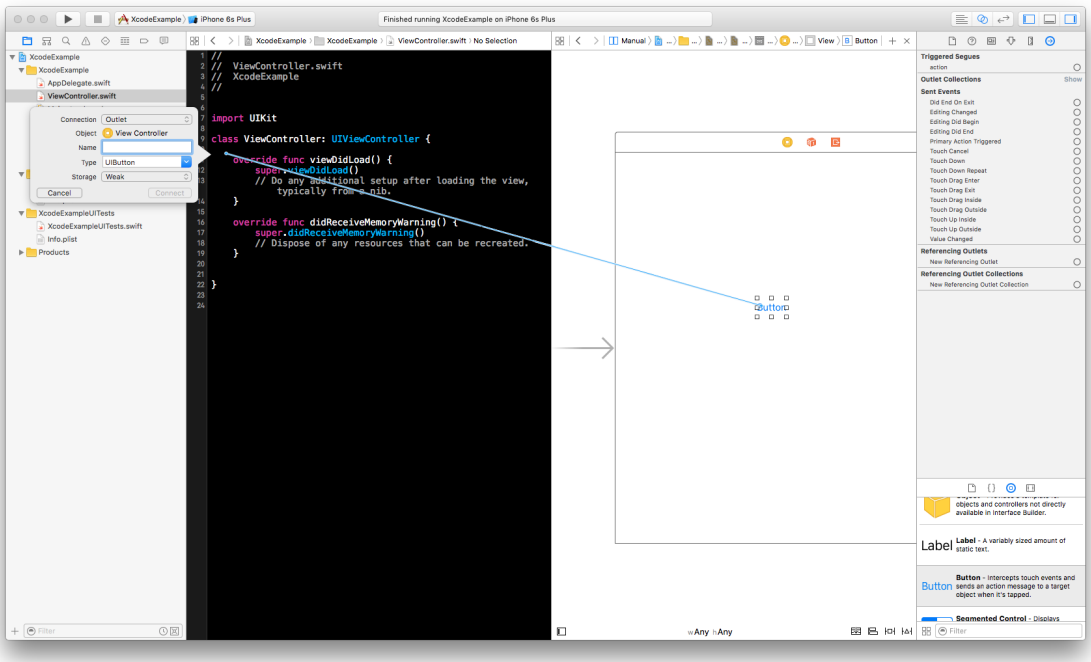


Figure 2.10: An example of connecting a user interface element to code.

2.3.4 CocoaPods

Adding third party libraries to a project can sometimes be a burden especially managing the dependencies. CocoaPods alleviates this problem by adding and managing third party libraries directly into an Xcode project. A Podfile is added to the project directory, which is a plain text file and contains a list of libraries, and dependencies the project needs. Once the Podfile is executed the libraries, and its dependencies are automatically added to the project [18].

2.4 Hardware

2.4.1 Overview

Since, HeartMate is an iOS health and fitness app, there are some specific hardware components of an iPhone that should be discussed along with external heart rate monitors, as well as their communication between devices. This section is to understand the iPhone and the health sensors integrated to assist fitness tracking, as well as third party heart rate monitors and how they communicate back to the iPhone.

2.4.2 iPhone Hardware

Devices

Since its incarnation, the iPhone has shown many forms, especially with advancements in the sensors integrated within them. The current devices on market are the iPhone 6S Plus, iPhone 6S, iPhone 6 Plus, iPhone 6, and the iPhone SE, all of which are running the latest version of iOS 9.

Sensors for Health Tracking

The iPhone being as advanced as it is the hardware integrated into the device provides sensors that are prefect for tracking a user's fitness. These sensors include an accelerometer, gyroscope, barometer and compass. Having sensors such as the ones mentioned provide insight to a way a user is moving, as well as direction. Bluetooth

LE is also incorporated to the iPhone hardware and allow for third party health sensors to be connected directly to the iPhone.

M9 Chip

When Apple released the iPhone 5S, they introduced a new chipset, the M7 Motion Coprocessor, which allowed utilization of the gyroscope and accelerometer to its full potential. The M-Series chipset was refined and improved with each iPhone iteration thereafter, the latest being the M9 chip. In general, the M-Series chipset is designed to continiously process and store data from these sensors and provide an abstraction of the sensor data through either Health Kit, or the Core Motion API.

2.4.3 Bluetooth LE

There are many different communication protocols used to connect hardware devices to each other. These protocols were created for convenience to allow sensors, and hardware accessories to connect to other devices wirelessly. Some communication standards such as Wi-Fi used too much power, and often these sensors or accessories were battery powered and needed to conserve energy. Bluetooth was thus created to communicate with devices such as cell phones, computers, and accessories. The first iterations of Bluetooth failed to deliver a low powered solution. The introduction of the Bluetooth Low Energy (LE) protocol, solved the issue of power, and was made to communicate with devices powered by a single coin cell battery. Bluetooth LE was designed for devices such as motion sensors, light detectors, thermostats, pedometers and heart rate monitors [51]. The Bluetooth LE protocol makes it easy to connect to these devices by providing a generic gateway to communicate directly with the hardware [19].

2.4.4 Heart Rate Monitors

Heart rate monitors are exactly what the name suggests a device whose sole purpose is to gather a heart rate signal from the human body. The popularity of smartphones

integrated with Bluetooth LE have increasingly become the norm, and due to this heart rate monitors are no longer only considered a dedicated medical device. Heart rate monitors are used to measure and track fitness intensity [47]. Some heart rate monitors using Bluetooth LE have been designed specifically to use on the iOS platform [27]. HeartMate will use two generic Bluetooth LE heart rate monitors, Polar H7 and the Wahoo TickrX heart rate sensor.

2.5 Client-Server Model

2.5.1 Overview

Since HeartMate will be a competitive / motivational fitness application with a social network component there is a need for the client (HeartMate) to communicate with outside sources. To accommodate for the social network aspect of HeartMate, a client-server based approach is implemented with our prior knowledge of such architecture [13].

2.5.2 Structure

A client-server based system is where each device acts as either a client or server. Clients and servers are components in a network in which a client-server relationship is established when a client and server component connect with each other through a network. A server provides resources and services, while a client sends requests to the server in order to utilize those resources and services [35].

2.6 Libraries and Frameworks

2.6.1 Overview

This section will discuss the libraries and frameworks used in the implementation of HeartMate. Although, how they are used in HeartMate will be discussed in Chapter 4 and Chapter 5.

2.6.2 Swift

Swift is a new general purpose programming language used to implement software for Apple's operating systems such as OS X, tvOS, watchOS, and of course iOS. Objective-C was originally the only language to produce software for the aforementioned operating systems, but with Swift being the intended replacement for C-based languages it was the language of choice for HeartMate. Recently, Swift was made open source and can now be used to produce software on Linux, and many more platforms to come. Swift provides an easier way to write software as it was built using a modern approach to software design patterns, safety and performance. Swift also provides features such as inferred types, elimination of header files, namespaces, and the automation of memory management. Swift is safe as variables always are initialized before use, while arrays, and integers are checked for overflow. By default objects can never be nil and if a nil object is discovered there will be a compile-time error, by doing this Swift alleviates common runtime crashes. The use of nil is not completely omitted as there are reasons to use such a value, and Swift allows nil through the use of optionals, which are denoted by a '?', which signals the compiler the user understands the value could be nil and it will be handled safely [2]. All of the mentioned features are why Swift was the chosen language for HeartMate.

2.6.3 JSON

JSON or JavaScript Object Notation is a data interchange format that allows serialization of structured data. The structured data is in text format and can represent the primitive types (strings, numbers, booleans and null) as well as objects and arrays. JSON objects are in the same form as the data structures, dictionaries, as they too are a collection of key-value pairs, but JSON is serialized making it possible to send over a network [14]. Two frameworks, Parse and PubNub, to be discussed utilize JSON objects to handle communication between HeartMate, database, and other devices with HeartMate installed.

2.6.4 Parse

Overview

Parse is a Mobile Backend as a Service (MBaaS) and provides a backend for mobile applications. The reason Parse is appealing is because it eliminates the need to setup and maintain databases along with managing or hosting a server. Parse is broken up into sections based on what it can provide for an application. These sections are core, push, and analytics all of which can be accessed through the Parse Dashboard. Analytics provide information on how the application is performing such as active users, installs, and database usage. Push provides information on the notifications sent, and allows for the creation of a push notification [40]. Parse Dashboard, and the core section of Parse will be discussed further in this section.

Parse Dashboard

The Parse Dashboard provides a front end access to the backend server in which a mobile application will communicate and make requests. Parse Dashboard provides access to the core, push and analytics of the mobile app. The Parse Dashboard is shown in Figure 2.11.

Core

HeartMate focuses on the Core functionality that Parse provides. Core provides database and Cloud Code management. Tables as well as their rows and columns of a database can be managed through the dashboard. Parse is capable of managing user accounts, and by default the database is populated with a *User* table. Through the dashboard, manual entry of a user is possible, otherwise it is handled through the Parse API. Parse also allows a developer to provide customizable server code for instances where custom server functionality is needed and this is called, Cloud Code. The Cloud Code is written in JavaScript, a programming language typically used for web based applications, and managed through the dashboard. This cloud code can be called through an API from the mobile application to trigger certain events like

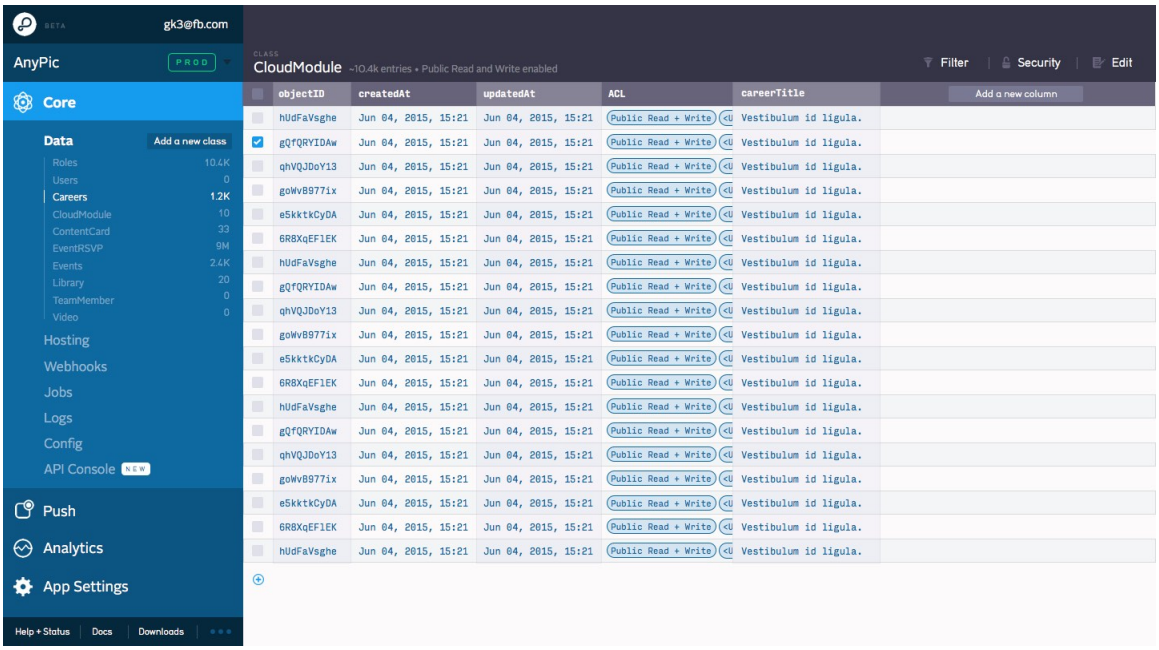


Figure 2.11: The Parse Dashboard interface, from [30].

push notifications, or modifications to the database. Parse also provides methods to define a relationship between tables in a database, and are called, Parse Relations. Relations can use pointers to point from a column in a database to another table and are capable of defining many to many, and one to many relations between objects [40].

Data is stored as a *PFObject* which contains JSON-compatible data so essentially the *PFObject* is JSON, but is recognized as a *PFObject* from the perspective of the framework. Since, the underlying structure of a *PFObject* is JSON this is how data is received and sent from the mobile device. A *PFObject* is schemaless which means the keys do not need to be created on the *PFObject* beforehand and will be created during assignment if it does not exist. Requesting data from the Parse server comes asynchronously and is requested through the use of PFQueries. A PFQuery is built with the table's name, and the conditions provided [40].

2.6.5 PubNub

Overview

PubNub is a Data Stream Network and real time Infrastructure as a Service (IaaS), and one of the primary services offered is the publish / subscribe API that hooks into their data stream network. PubNub offers support for many devices 70+ SDKs such as iOS, Android, JavaScript, .NET, Java and more. With PubNub a developer does not have to worry about the infrastructure it takes to create a real time network, and only need to utilize the APIs provided to start streaming in real time [42]. Only three of the many features of PubNub provides will be discussed in this section which are publish / subscribe messaging, presence and stream controller.

Publish / Subscribe Messaging

PubNub offers streaming of data in real time through the use of the publish/subscribe model. In a publish/subscribe model there is a publisher, and a subscriber. A publisher creates a channel and through this channel they send data (messages), and a subscriber subscribes to the publisher's defined channel and receives any data (messages) the publisher sends. The data sent over PubNub is in the form of JSON. PubNub offers a network to manage the real time transmission of this data as well as the channels associated [42]. An example of the publish / subscribe model using PubNub can be seen in Figure 2.12.

Presence

Presence is a feature of PubNub that works alongside the publish / subscribe messaging feature, and can detect when a user has subscribed to a channel, left or any other customizable state. Information such as how many users are subscribed and if a user has subscribed to a specific channel is all available in real time, and the application can be immediately updated [42].

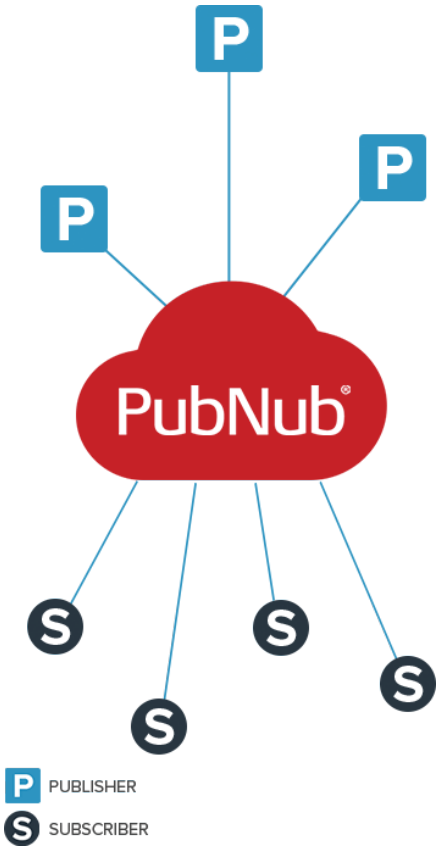


Figure 2.12: Publish / Subscribe model using PubNub, from [42].

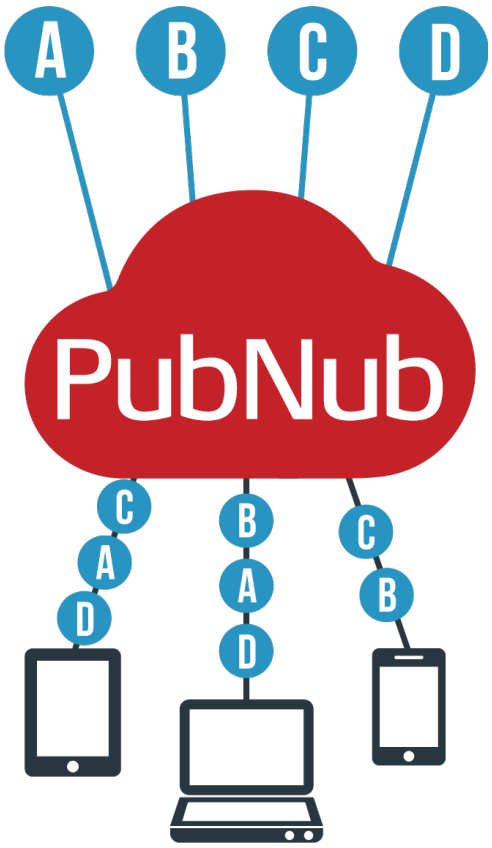


Figure 2.13: An example of channel groups using PubNub, from [42].

Stream Controller

The stream controller feature is a way to manage groups of channels. The stream controller works in tandem with publish / subscribe, and presence. This allows subscribers of a specific channel group receive data in real time all at the same time. This is useful for publishers as they only have to publish to a specific channel group, and any subscribers associated with that group is sent the data [42]. An example of how publishers can publish to specific groups of subscribers can be seen in Figure 2.13.

2.6.6 BEMSimpleLineGraph

BEMSimpleLineGraph is a third party library to create and customize line graphs for iOS. Animation control, colors, and line shape all can be configured, along with different types of interaction with the graph such as popup and touch reporting.

Figure 2.14: An example of a line graph with popup and touch reporting using BEMSimpleLineGraph, from [22].

Popup and touch reporting allow the graph to respond to user touch, and provides a popup above the data point as the user is touching along with its value which can be seen in Figure 2.14 [22].

2.6.7 Charts

Charts is also a third party library for creating graphs for iOS, but this one specializes in more than line graphs. In addition to line graphs, charts can create seven other different chart types such as bar, scatter, pie, combined (i.e., line and bar), candlestick, bubble, radar and line. Legends are automatically generated and can be configurable. Charts can be animated, and there is support for Android, iOS, tvOS, and OS X. The pie chart is the focus of this project, and an example can be seen in Figure 2.15 [26].

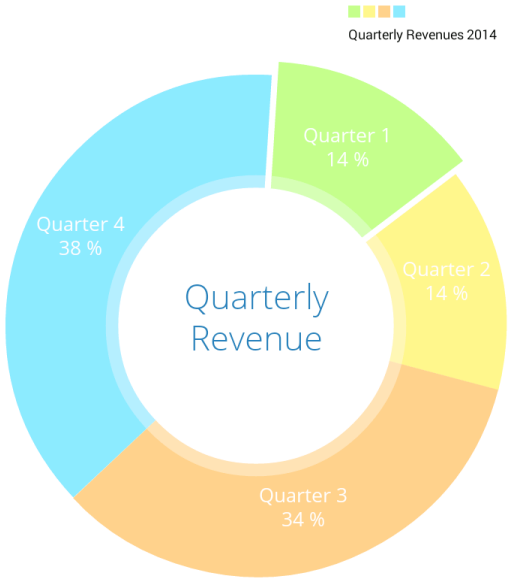


Figure 2.15: An example of a pie chart using the Charts library, from [26].

2.6.8 SWReveal

SWReveal is a third party interface library, and produces a more common menu option system found in many apps. This particular menu style has the main view slide to the left or right revealing a menu underneath [54]. SWReveal makes it easy to integrate into an iOS application. An example of SWReveal can be seen in Figure 2.16.

2.6.9 DPMeterView

DPMeterView is a third party custom gauge-style meter view that can be integrated into iOS applications. Provides easy customization of the meter views to indicate a percentage of progress completed [21]. An example of this interface element is shown in Figure 2.17.

2.6.10 Health Kit

Health Kit is a framework created by Apple, and was created with the increase of health apps being produced. Apple wanted to make it easier for developers to access health data provided by sensors embedded into the iPhone hardware as well as a way

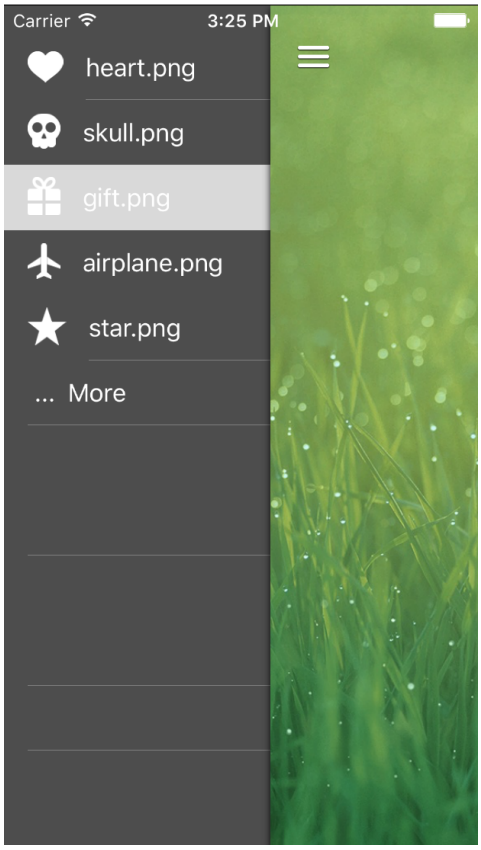


Figure 2.16: The SWReveal menu system, from [54].

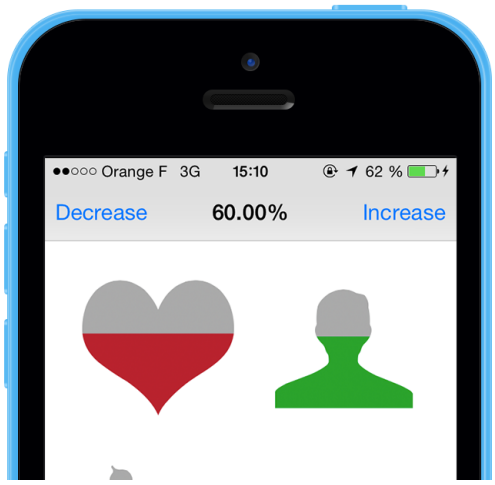


Figure 2.17: DPMeterView interface elements, from [21].

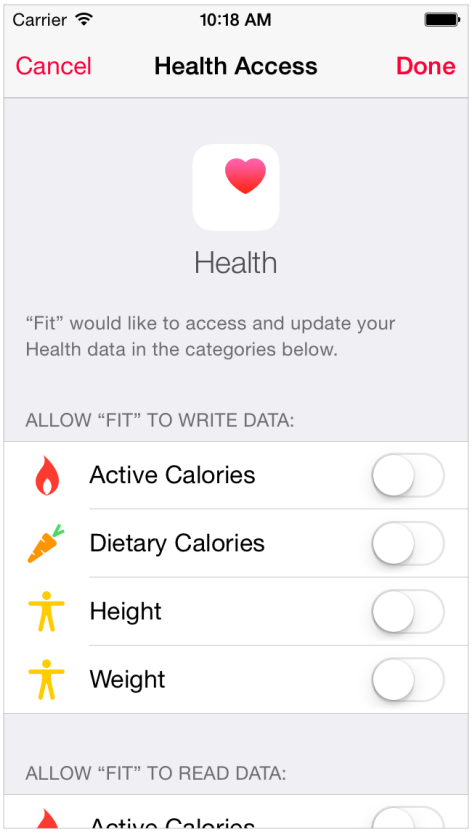


Figure 2.18: An example of requesting permission to access health data, from [7].

to share health data securely and privately between apps. The Health Kit framework works directly with third party health and fitness hardware in which allows the system to access and save data from compatible Bluetooth LE heart rate monitors, and import activity data from the M-Series Motion Coprocessor. The Health Kit framework allows the user to be in control of their data and permission must be granted when an app tries to access such information, which can be seen in Figure 5.6 [7].

2.6.11 UIKit

UIKit is a major framework created by Apple, and it provides API access and management to interface elements of an iOS application. This provides the connection between code and the interface as well as methods to respond to user input [10].

2.6.12 Map Kit

Map Kit is another framework created by Apple, and allows developers to easily embed a map interface onto windows and views. Through Map Kit it is possible to annotate the map, add overlays as well perform a reverse look up for a given map coordinate [8].

2.6.13 Core Data

Core Data framework is developed and maintained by Apple, Core Data is used to manage objects in an application. Core Data does this by providing a way to generalize and automate solutions and tasks related to object life cycle, object graph management, including persistence. Mainly, for the scope of HeartMate Core Data is used for data persistence. Core Data is similar to that of a database and can store objects structured as a table, and relationships between objects can be defined as well [5].

2.6.14 Core Location

The Core Location framework, created by Apple, allows access to location data from the available hardware embedded into the system such as GPS. This location data can be used to define geographic regions as well as monitor the regions for when a user has left the region's bounds [6].

2.6.15 MediaPlayer

Created by Apple, the Media Player framework provides necessary API for playing a movie, music, audio podcast, and audio book files. The Media Player framework gives access to iPod controls for where background media control may be necessary. This framework also provide ways for an app to respond to events sent by external media players [9].

2.6.16 AVFoundation

The AV Foundation framework, created by Apple, provides methods to play and create time-based audiovisual media. Media files can be created, examined or re-encoded through the use of this framework. This framework also provides a technique to translate text to speech [4].

Chapter 3

Design

3.1 Overview

HeartMate is an iOS fitness running application that utilizes a Bluetooth connected heart rate monitor to provide a new competitive and motivational environment through a social component using heart rates as the performance measure. With HeartMate, a user can create an account, add friends, customize their profile, track and map a run, store runs locally and on a server, challenge their previous runs or friends' runs, and challenge a friend to a real time run.

When designing a system it is important to define the non-functional and functional requirements as they are important to creating a quality system [15]. Non-functional requirements are defined as the properties of a system, while functional requirements represent the system's functionality [16]. The functional and non-functional requirements, user interface use cases, architecture of HeartMate and its interactions with the Parse server, and PubNub are described in this chapter.

3.2 HeartMate Requirements

3.2.1 Functional Requirements

Behavioral requirements of HeartMate were considered when creating the functional requirements. Table 3.1 shows a list of these functional requirements.

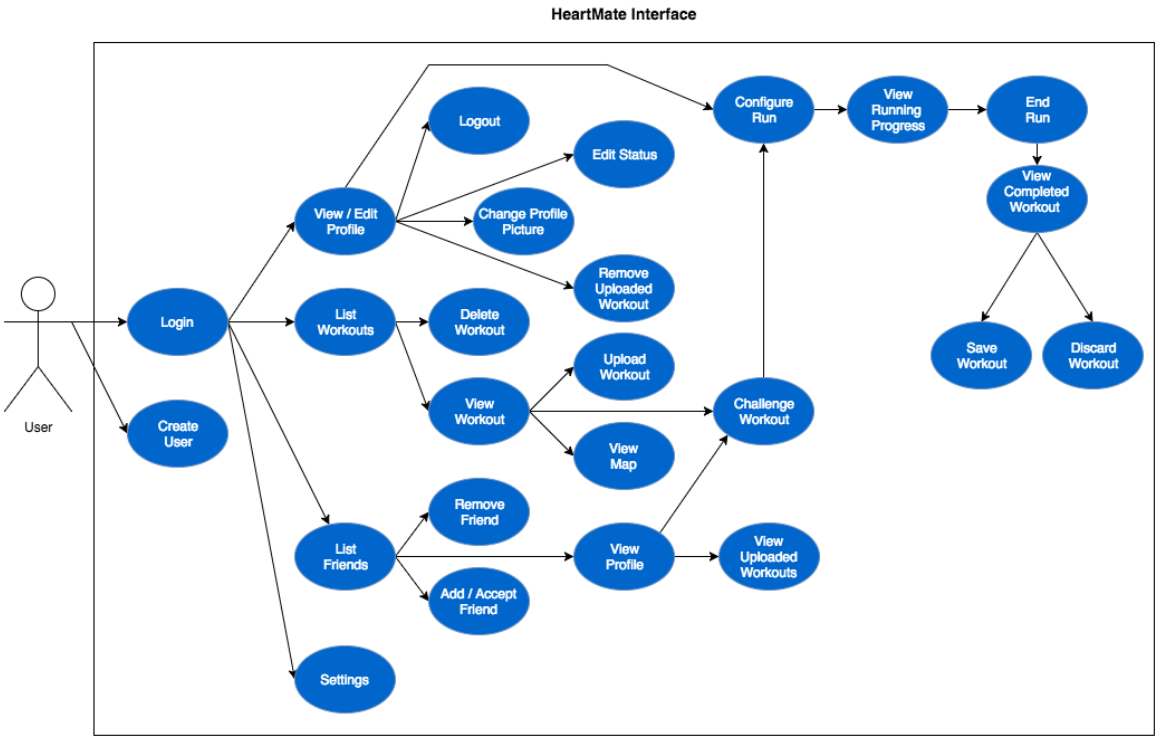


Figure 3.1: HeartMate's use case diagram.

3.2.2 Non-functional Requirements

Implementation specifics and communication with the server from HeartMate were considered when creating the non-functional requirements to complement the behavioral requirements of the system. Table 3.2 lists the non-functional requirements.

3.3 Use Case Modeling

3.3.1 Overview

This section provides a better understanding of HeartMate's responsibilities. This section describes HeartMate's use cases in detail as well as a user's interaction with the application. Figure 3.1 shows a visual representation of HeartMate's use case diagram, while detailed descriptions of each use case will be discussed in this section.

Table 3.1: HeartMate Functional Requirements.

Number	Description
FR01	HeartMate shall create user accounts.
FR02	HeartMate shall authenticate users given the user credentials.
FR03	HeartMate shall allow a user to edit their profile.
FR04	HeartMate shall begin a workout and track the progress of a user.
FR05	HeartMate shall allow a user to manage uploaded workouts.
FR06	HeartMate shall allow a user to configure a workout.
FR07	HeartMate shall allow a user to create a competition from their previous workout or a friend's.
FR08	HeartMate shall allow a user to create a real time competition with a friend.
FR09	HeartMate shall stream heart rate data in real time.
FR10	HeartMate shall save workouts locally, and to a remote database.
FR11	HeartMate shall allow a user to view saved workouts.
FR12	HeartMate shall allow a user to add or accept friends.
FR13	HeartMate shall retrieve a user's friend requests.
FR14	HeartMate shall retrieve a user's friends list.
FR15	HeartMate shall allow a user to remove a friend.
FR16	HeartMate shall allow a user to view a friend's profile.
FR17	HeartMate shall allow a user to view a friend's uploaded workouts.
FR18	HeartMate shall allow a user to enter user specific settings.

Table 3.2: HeartMate Non-functional Requirements.

Number	Description
NR01	HeartMate shall be written in Swift.
NR02	HeartMate shall use Parse to manage a client-server architecture.
NR03	HeartMate shall use PubNub to manage streaming heart rate data.
NR04	HeartMate shall use Parse to store user profiles and workout data.
NR05	HeartMate shall accept JSON objects from PubNub for heart rate data.
NR06	HeartMate must use Health Kit to store workout data locally.
NR07	HeartMate must use Core Data to store location data.
NR07	HeartMate must use a Bluetooth LE connected heart rate monitor to retrieve heart rate data.

3.3.2 Detailed Use Cases

Login

Login is required for the user to access their profile, and use HeartMate's functionality. A user logs in by providing a username and password they used during the creation of their HeartMate account or logs in through Facebook. If the valid credentials are provided the user is then given access to HeartMate's functionality.

Create User

The initial view of HeartMate gives the option for the user to either login, or sign up. A user can either register by providing their first and lastname, username and a password or through connecting their HeartMate account to their Facebook account. If the latter is chosen the user must provide a username after Facebook succesfully links to their newly created HeartMate account. The user will only be added to the database if the username entered is not taken.

View / Edit Profile

This is the initial view the user is shown after logging into HeartMate, and is also known as the "User" tab from the menu. This gives the user access to their profile picture, status, and uploaded workouts. The user can also navigate the app, logout, as well as configure a new run from this section.

Edit Status

A user can edit their status, which adds to the social network component of HeartMate. This status is what friends will see when the user's profile is being viewed.

Change Profile Picture

A user can also provide a picture to associate with their profile. A user can select a picture from their local device, take a picture, or if they signed up using their Facebook account, the user can import their Facebook profile picture.

Remove Uploaded Workout

A user can remove their uploaded workouts from the remote database associated with their profile under the menu option, “User”.

List Workouts

List workouts can be found under the menu option, “Activity”. Here a user will be able to view a list of their previously saved workouts on the device in the form of a list. The list provides information on the workout such as average heart rate, distance, duration, date, and whether the workout took place outside or indoors.

Delete Workout

A user can delete a workout from their list of workouts. The locally saved workout is removed from the device entirely.

View Workout

When a user selects a workout from the list of workouts the user is taken to a more detailed view of the workout. A heart rate graph, level of intensity pie chart, location, duration, distance and calories burned can all be seen in this view. The user has the option to upload their workout as well.

View Map

While viewing a previous workout, if the workout took place outdoors a map of the run can be viewed. The map is overlayed with the path the user took during their run.

Upload Workout

While a user is viewing their workout they have the option to upload their workout for their friends to challenge.

List Friends

Since there is a social component to HeartMate a user can view their list of friends who also use HeartMate. This is accessed through the menu option, "Friends". Their name as well as username is shown in this list along with their profile picture.

Remove Friend

In the list of friends a user can remove a friend.

Add / Accept Friend

Under the "Friends" menu option there is also a way to add a friend by searching for them through the use of their name, or username. The user can also view pending friend requests, and accept or deny a friend request.

View Profile

When a friend is selected from the user's list of friends the user is taken to view the friend's profile. Here the user can view the friend's profile picture, status, view their uploaded workouts, and challenge one of their uploaded workouts.

View Uploaded Workouts

Viewing a friend's uploaded workout allows the user to see their heart rate in the form of a graph as well as their level of intensities through a pie chart. This allows a user to contemplate challenging that particular workout.

Challenge Workout

Challenging a workout can be done either through configuring a run, selecting a friend's uploaded workout from their profile, or using a previous user's workout from the view workout section.

Configure Run

During a run configuration a user can choose the location of their run (indoors, outdoors), create competition, and select the duration of the workout from a predefined list. If a user has selected a workout to challenge, the duration, and competition settings will be automatically set from the workout they are challenging, however the location is still configurable. If a user was challenged to a real time workout the duration, and competition settings will be set to the challenger's preference, but the location is still user configurable. If the user is configuring a run for the first time the global settings will appear for the user to enter data pertinent to accurately calculating workout data.

View Running Progress

Once a workout has started the user is able to view their current progress such as time remaining, current heart rate, current level of intensity, distance traveled, and calories burned. A heart rate meter is shown along with a scale of their zone intensity and heart rate zone thresholds. The user is given the option to end the workout as well.

End Run

The run will either come to an end when there is no more time remaining, or when a user intentionally ends the run. If the user was currently partaking in a real time competition their opponent will be notified and their run will also be terminated.

View Completed Workout

Once a run is complete the user will be taken to the completed workout view in which they can view detailed stats about their workout. These stats include a graph of their heart rate data, pie chart of their workout zone intensities, duration, distance, calories burned along with a map of their run if the location was outside. If the user was in a competition the user will also be shown the heart rate graph, level of intensity pie

chart, and a score of how much longer they were in a higher zone to compare their workout to their opponent

Save Workout

In the completed workout view the user has the option to save their workout. Saving the workout will save to the device locally, or depending on the user's global settings for auto upload the workout may or may not be uploaded to the remote database.

Discard Workout

In addition to saving, the user may want to discard their workout and not save at all in which then the workout is discarded. If the user intentionally ended the run, the user will only be allowed to discard the workout.

Settings

Settings can be found under the menu option, "Settings". Here the user can provide information to accurately track their workout. This information is the user's age, gender, weight and resting heart rate. The user can import this information from Health Kit, which pulls from their health app on their device. App settings can also be configured here in which the user can turn on auto upload of workouts as well as select the amount of automatically uploaded workouts they would want stored at a time.

Logout

A user can logout from HeartMate from their profile view in which the user will then be prompted to login the next time they open the app.

3.4 Architecture

HeartMate adheres to the client-server architecture to manage various aspects of the functionality such as the use cases of the app previously defined. The HeartMate app acts as the client, making requests as well as receiving data from third party

components that act as the server. These third party components include Parse, PubNub, Apple Push Notification Service (APNs) and a Bluetooth connected heart rate monitor. A visual representation of HeartMate, and its interaction with third party components can be seen in Figure 3.2. Arrows from the diagram represent the requests and services each component provides for HeartMate.

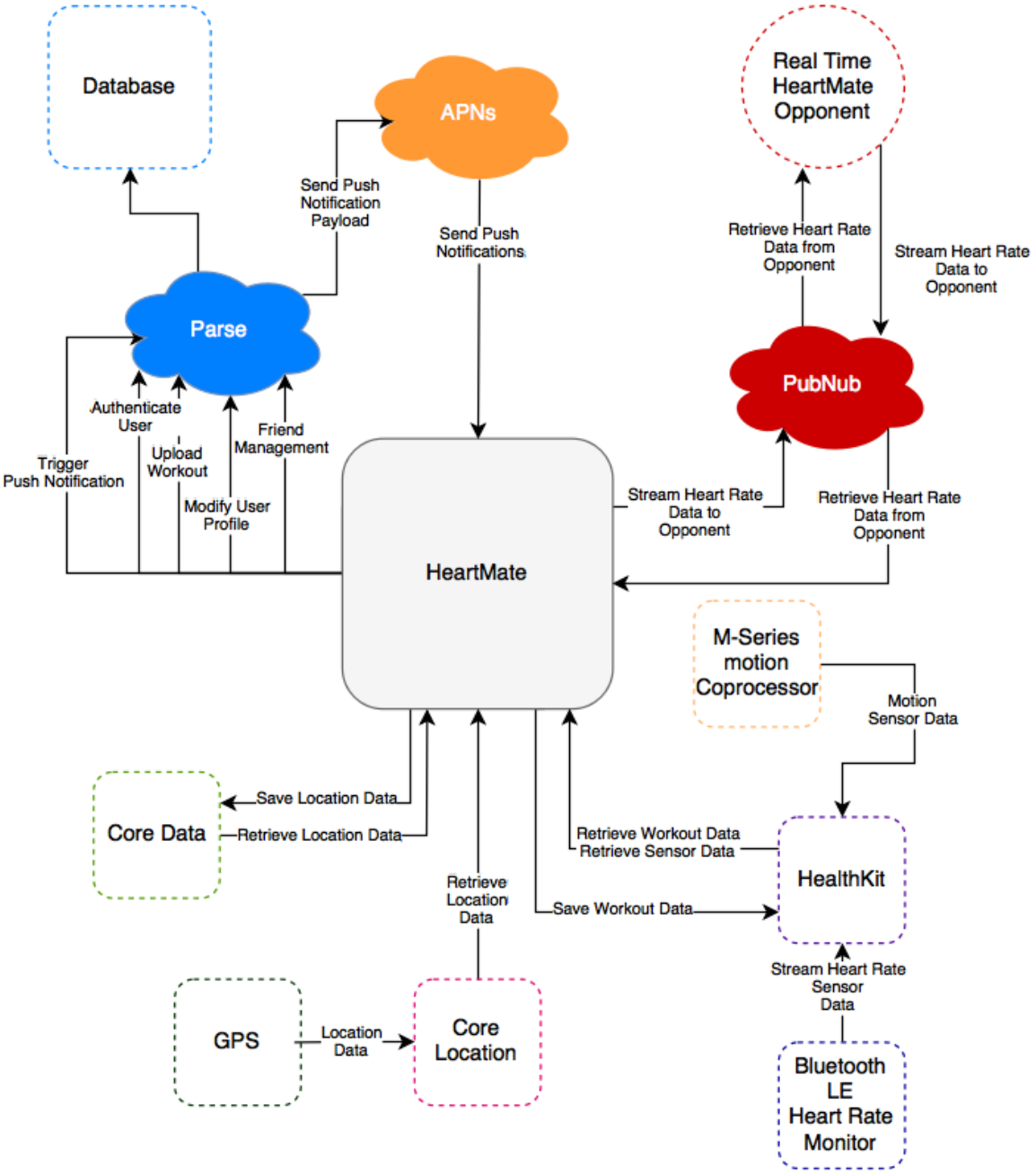


Figure 3.2: HeartMate's system architecture and its interactions with other system and hardware components.

Chapter 4

Implementation

4.1 Overview

This chapter describes the implementation of each component used in the creation of HeartMate. Libraries and frameworks described in Chapter 2 are also discussed to show how and where they were used during the development of HeartMate.

4.2 Social Component

4.2.1 Overview

HeartMate has a social network component that provides users with the ability to search and add friends, as well as create user profiles, and have the ability to upload their workout for others to challenge. This section describes the structure of the database, and added server functionality to manage push notifications. The system components that are discussed can be seen in Figure 4.1.

4.2.2 Database Management

Overview

HeartMate uses Parse as the backend to manage the database information to store user profiles, friend requests, friends, and user workouts. Parse provides an API to access the information stored on the database and will be discussed.

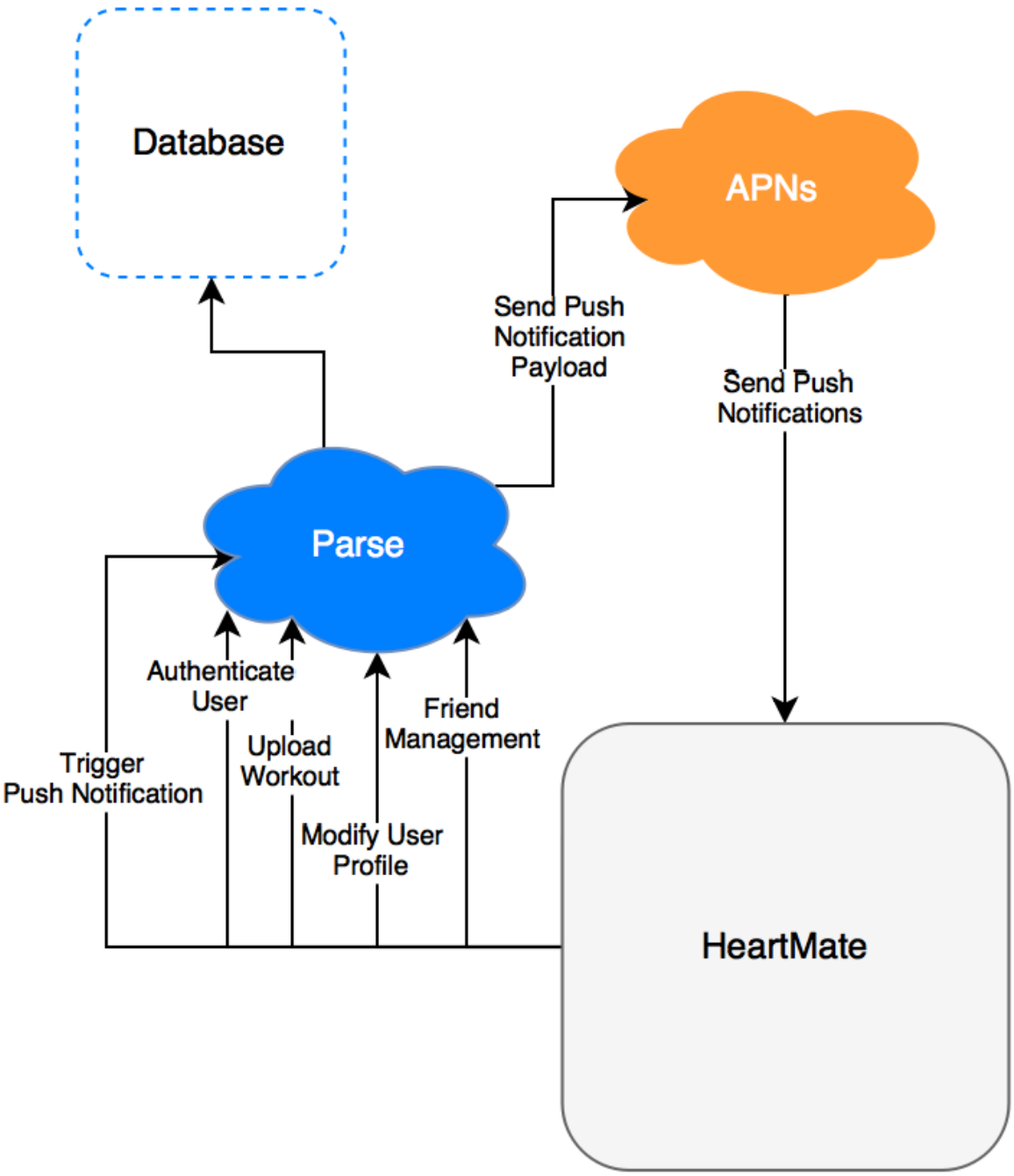


Figure 4.1: HeartMate's interaction with the database, and Apple's Push Notification Service through Parse.

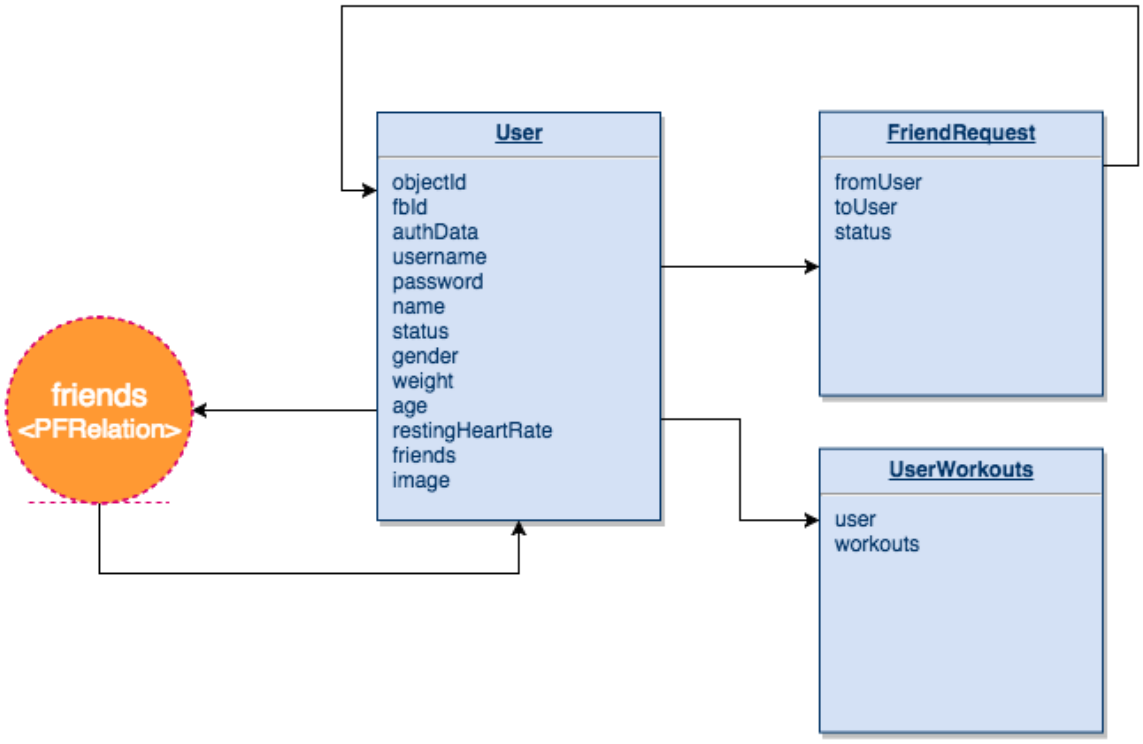


Figure 4.2: The structure of the server side database.

Structure

The database is structured to manage the social component functionality with efficiency. The defined table entities are as follows: *User*, *FriendRequest*, and *UserWorkouts*. A user's friends list, *friends*, is a Parse Relation to a *User*. The structure of the database, and the friends relation can be seen in Figure 4.2. Everything stored on the Parse backend is a *PFObject*, which internally is a JSON object.

User

The *User* entity contains information to users that have signed up through the HeartMate app, as well as information pertaining to their health. The *User* attributes include a unique ID, user login and profile information, health information such as weight, resting heart rate, as well as information pertaining to their linked Facebook account, an image field is also provided for the user's profile image. When a user account is created through HeartMate the user has the option to link their Facebook

```
{
    "objectId": "kmmKG4NmLd",
    "fbId": "125485848521216548545 4",
    "authData": {
                    "facebook": {
                                        "access_token":"
                                            JKLJjklJkl767ghhHGkjjKL",
                                        "expiration_date":"2016-06-23T04
                                            :07:28.487Z",
                                        "id":"125485848521216548545 4"
                                    }
                },
    "username": "jAppleseed",
    "password": "[hidden]",
    "name": "John Appleseed",
    "status": "Ready to challenge everyone",
    "gender": "male",
    "weight": 145,
    "age": 24,
    "restingHeartRate": 65,
    "friends": "[PFRelation]",
    "image": "[PFFile]"

}
```


Figure 4.3: An example of a user stored in the *User* entity.

account which uses a built in Parse API. The login information is stored into the *authData* field, and the Facebook Id, *fbId*, of the user is stored as well. Authentication information is encrypted by the Parse backend. The *PFObject*/JSON structure of the *User* entity can be seen in Figure 4.3. On the device in the HeartMate app, a *User* is represented as a *PFObject* or *PFUser* which is a subclass of *PFObject*. *PFUser* also provides information to the *currentUser* logged into the device.

Friend Request

The *FriendRequest* entity contains information regarding a friend request. For every friend request an entry is made with a pointer to the user that made the request, a pointer to the user requested and the status of the friend request, such as pending, accepted or denied. This information is used by HeartMate as the first step in creating a link between other users in the friends relation. The structure can be seen in

```
{
    "fromUser": kmmKG4NmLd,
    "toUser": o63K1KEZcu,
    "status": "pending"
}
```

Figure 4.4: An example of a friend request stored in the *FriendRequest* entity.

```
{
    "user": kmmKG4NmLd,
    "workouts": [ ],
}
```

Figure 4.5: An example of a workout stored in the *UserWorkouts* entity.

Figure 4.4.

User Workouts

The *UserWorkouts* entity contains uploaded workouts a user wants their friends to challenge. This entity represents a one-to-many relationship where one user can have many workouts stored. The entity consists of a pointer to a user, and an array of uploaded workouts since there is a predefined limit of how many workouts a user can have stored at a time an array is sufficient enough for this data. The array is a collection of JSON or dictionary objects that represent a workout. The workout JSON object provides context to the workout such as date, age, resting heart rate and an array of heart rate data representing the workout. The structure for the *UserWorkouts* entity is shown in Figure 4.5 and for readability reasons a visual representation of a workout containted in the *workouts* array can be seen in Figure 4.6.

```
{
    "date": "2016-02-22T23:25:49.001ZkmmKG4NmLd",
    "age": 25,
    "restingHeartRate": "65",
    "heartRateData": [ ]
}
```

Figure 4.6: An example of a workout object that can be stored in the, *workouts*, array of a *UserWorkouts* entity.

Friends Relation

Parse Relations or a *PFRelation* is used to define relations between entities in the database. Relationships such as one-to-many and many-to-many can be defined using a Parse Relation or other methods, but a Parse Relation is used for efficient scalability. A friends list is a many-to-many relationship as many users can have many friends. The *friends* relation is not an entity, but of a *PFRelation* object, and is used to define the friends list of a user. A *PFRelation* is an attribute in an entity and defines a pointer to a collection of pointers to other entities. As seen in the Figure 4.3, the attribute *friends* exists as a *PFRelation*, and this is simply a pointer to a collection of pointers to an entry in *User* entity. The *friends* relationship can be seen in Figure 4.2.

Parse API

Save The *save* function is used to add entities or modify entities on the database. This is a class function of *PFObject* or *PFUser*, and since they are schemaless if the entity or attribute does not exist on the database then it will be created once saved. This function is used to save any changes to the user's profile information, upload workouts, and create friend requests.

Query The *query* function allows the database to be searched with a *PFQuery* object. The *PFQuery* object is used to define search conditions on the database such as the name of the entity to be searched, as well as any attribute element constraints. This is used to search and retrieve users' uploaded workouts, as well as user information pertinent to defining the context of a workout such as health data. Friend profile information is also retrieved using the query method. Querying also provides a way to update objects as well by searching for the object and retrieving it followed by modification then saving the object through the save function. An example of this would be to modify the *FriendRequest* entity and update the status to pending, denied or accepted.

SignUp The *signup* function is used on a *PFUser* object. The *PFUser* object's properties are set such as username, and password, and is then signed up through the *signup* function. This is used to sign up users, and add them to the *User* entity on the database.

Login The *Login* function checks the credentials of a user and is successful if the credentials provided are validated checking the *User* entity. Once a user is logged in the *currentUser* is cached to the user's profile, and is always logged in until the user logs out. A user can be logged in by linking their Facebook account, and login is called using the *PFFacebookUtils* object, which requests the user to login using their Facebook login information. If the login is successful their *authData* will be added to their user profile in the *User* entity. Facebook information can be requested, such as their profile picture, and Facebook ID. The Facebook ID is used to retrieve the profile picture, and is added to the user's profile in the *User* entity for later retrieval.

Logout The *Logout* function clears the *currentUser* on the app, and will result in the user having to login.

Cloud Define The *Cloud.Define* function is used when creating a cloud function on the server. More detail discussion of a cloud function is discussed in the following section.

Call Function The *callFunction* is used to execute a cloud function defined on the server from the client. Parameters can be passed to the function from the client. This is used to call functions described in the following section.

4.2.3 Parse Cloud Code

Overview

The Parse backend provides customization of server functionality which is known as Cloud Code, and this code provides access to make administrative changes to

the database. This code is written in JavaScript and can be written in the Parse Dashboard. Sometimes there is a need for specific functionality to be integrated into the server such as administrative changes to the database such as adding or removing entries from an entity, or to act as the provider for the Apple Push Notification Service (APNs). A cloud function can be called through the Parse API as mentioned previously. A cloud function takes in a request, and a response with the request containing a JSON object with parameter information, and the response as a way to report if the request was a success. In this section, discussion of what cloud functions were implemented in the assisting HeartMate's functionality.

Add / Remove Friend To Friends Relation

Since, the *User* entity contains the *friends* relation, and only a user can edit any attributes associated with their entry from the HeartMate app using the Parse API, a cloud function was created. The cloud function was necessary to modify the *friends* attribute on any user contained in the *User* entity on the database. When a user accepts a friend request this cloud function is called with the necessary parameters such as the *userId* of the accepted friend, and is used to add the sender and recipient to each others *friends* relation. A function to remove instead of add to a *friends* relation was added as well.

Push Notifications

When a user signs up and logs into their account on HeartMate their unique device token is registered to the Parse database. This token is used to send a push notification payload to APNs, which then sends the payload to the user's device whose matches the device token. The push notifications that are managed by Parse through the use of cloud functions are when a user has requested to be another user's friend, a user has accepted a friend request, a friend is challenged to a competition, or when a challenge was declined/accepted.

From the HeartMate app when any of the conditions are made the Parse Cloud

```
{
    alert: "User: " + user + "\n Sent a friend request.",
    sound: "default",
    tag: "3"
}
```

Figure 4.7: An example of a friend request push notification, where user is the username of the sender.

function is called through the API sending the necessary parameters to the function on the server. The function is then executed on the server sending the corresponding payload to APNs. The parameters sent from the device are typically the user the notification is coming from along with their *objectId* (*userId*), and the user and *userId* of the recipient to build a push notification payload. The structure of a push notification payload is JSON. Since, the payload is of type JSON, it can contain any key-value compatible JSON data, but must contain at least an "alert" key containing the message to be sent for it to be structured for iOS. The "sound" represents the sound the device will make, and the "tag" is to allow HeartMate to identify which notification has been received to process correctly. The identification of a notification is used to present and setup the correct view to the user when the app is opened through a push notification. For example, if a user received a challenge notification the payload will contain information regarding the user, their userId, and the duration of the proposed competition. The *userId* is then used to retrieve the correct *PFUser* from their friends relation, and used to create a real time competition.

When sending a push notification using the Parse API, the recipient of the push notification is queried from the *Installation* entity that provides the association of a user, and their device token on Parse. Once the recipient's device token is retrieved, the Parse API, *Parse.Push.send(...)*, is used with the result of the query to send to the correct user, and the push notification payload, which is sent directly to APNs. The structure of a push notification payload is shown in Figure 4.7, and its use in the cloud function can be seen in Figure 4.8.

```
Parse.Cloud.define("friendRequestPushNotification", function(request,
    response) {

    var user = request.params.user;
    var userId = request.params.userId;

    var userRequested = request.params.userRequested;
    var userRequestedId = request.params.userRequestedId;

    var pushQuery = new Parse.Query(Parse.Installation);
    pushQuery.equalTo("userId", userRequestedId);

    Parse.Push.send({
        where: pushQuery,
        data: {
            alert: "User: " + user + "\n Sent a friend request.",
            sound: "default",
            tag: "3"
        },

        success: function() {
            response.success();
        },
        error: function(error) {
            response.error(error);
        }

    });

});
```

Figure 4.8: An example of a Parse Cloud function to send a push notification to alert a user of a friend request.

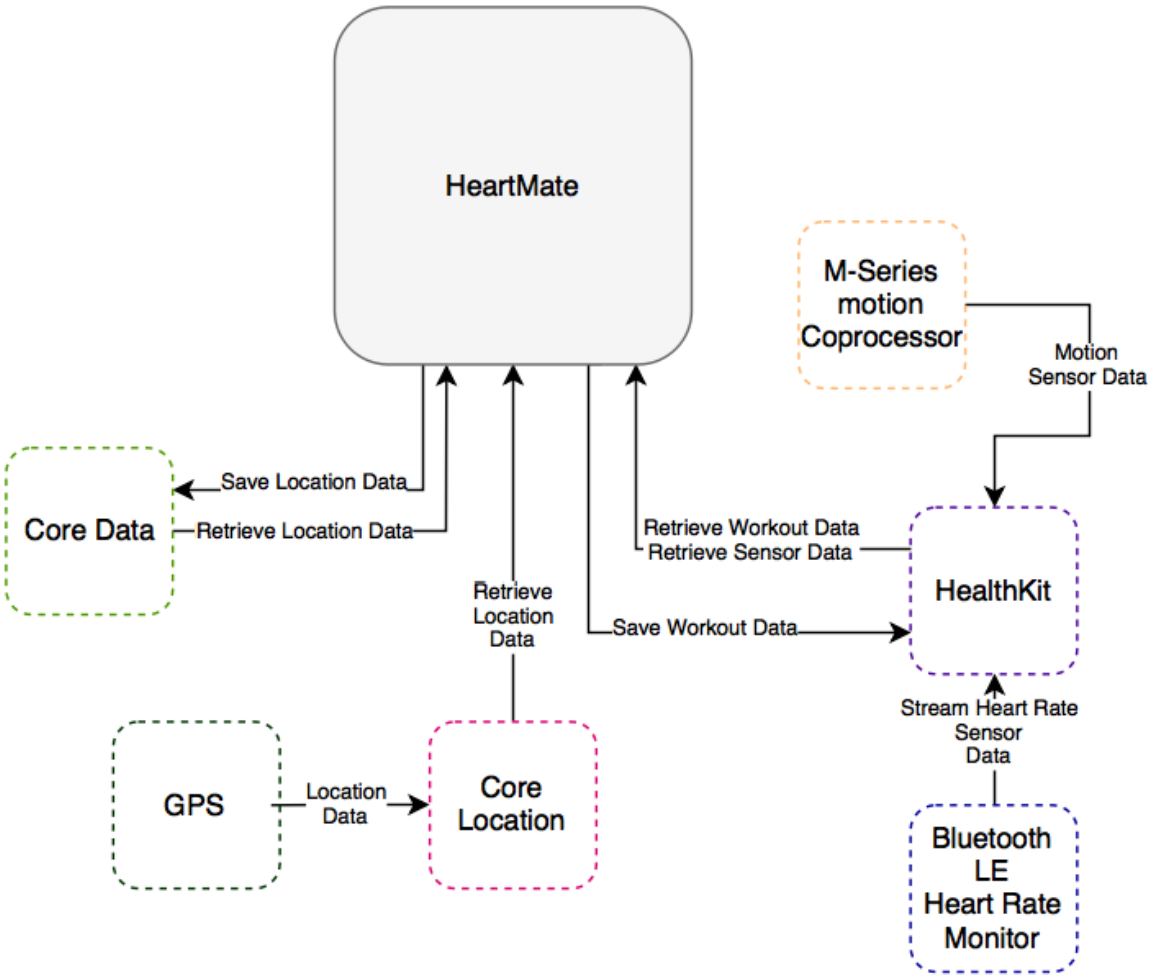


Figure 4.9: HeartMate's interaction with system components to track a run.

4.3 Tracking a Run

4.3.1 Overview

This section will discuss how the activity, running, is being tracked by HeartMate through its implementation using Health Kit, Core Data, Core Location, M-Series Coprocessor and a Bluetooth LE connected heart rate monitor. The system components involved in tracking a run are shown in Figure 4.9.

4.3.2 Workout Session Manager

Overview

A *WorkoutSessionManager* is what defines a current running workout in session. The *WorkoutSessionManager* is comprised of other objects to manage and assist with the collection of running data and information on how hard the user is working out. These class objects are *WorkoutSessionContext*, *HealthDataManager*, *LocationManager*, *WorkoutZoneManager*, and *CompetitionManager*. To provide insight on how these objects work together they will be discussed in this section with the exception of *CompetitionManager* as that will be discussed in a later section.

Workout Session Context

The *WorkoutSessionContext* provides configuration settings to the workout soon to take place. The *WorkoutSessionContext* sets the activity type (running), the location type (indoor or outdoors), duration, age and resting heart rate to correctly configure the workout. When saving workouts locally the *WorkoutSessionContext* is used as the metadata that is saved with the workout data to the device. This is used when retrieving the workout, and recreating workout conditions for competition.

Health Data Manager

The *HealthDataManager* provides HeartMate access to the user's health information from the Health app preinstalled on the iOS device using the Health Kit API. Health Kit also provides sensor data from the M-Series Coprocessor to retrieve distance information, as well as sensor data from any compatible Bluetooth LE heart rate monitor. The heart rate monitor must be connected under the bluetooth settings of the device, which makes it visible to Health Kit. Access to the information located within the Health app is accessible through what is known as an *HKHealthStore* object. For security and privacy reasons, the type of health data be written to or retrieved from the health store must be authorized from the user using the *HKHealthStore*. HeartMate requests access to be able to read and write to the following health information:

workouts, energy data, walking and running distance data, heart rate data, date of birth, and gender, an example of this request is shown in Figure 4.10. Data is retrieved using *HKObjectType* and *HKQuantityType* which are Health Kit objects that encapsulate this information and abstracts the data.

The *HealthDataManager* through the *HKHealthStore* object provides sensor data such as distance traveled reported by the M-Series Coprocessor if a user is performing a run indoors, and sensor data from the heart rate monitor. The information is retrieved from the sensors by the use of an *HKQuery*, *HKAnchoredObjectQuery* performed on the *HKHealthStore* object. An *HKQuery* object provides creation of a predicate to search for specific samples (data) the health store provides. The predicate is what defines some of the search criteria such as start and/or end date as well options that define how dates should be compared in the search. The *HKAnchoredObjectQuery* takes the predicate from the *HKQuery* to create a more specific query such as the type of samples (heart rate, distance), as well as the limit of data obtained, and the anchor previously returned by the last execution of an *HKAnchoredObjectQuery*, which corresponds to the last sample read. *HKAnchoredObjectQuery* is used to create a continuously updating stream of sample data during a user's workout from the health store. Whenever new data is retrieved an *updateHandler* associated with the *HKAnchoredObjectQuery* uses the last anchor as a point of reference to read in the next sample providing a continuous update. Heart rate data obtained through the heart rate monitor is updated every second. Once the workout is completed the queries must be manually stopped by stopping the query on the *HKHealthStore* object. The samples are in the form of a *HKQuantitySample* object, and data not retrieved from sensors can also be added to an *HKQuantitySample* such as energy. Energy expenditure is calculated using the heart rate based formula, Equation 2.4 and Equation 2.5, mentioned previously in Chapter 2. The energy calculation is performed during every heart rate data sample update.

All the data collected from the health store through the *HealthDataManager* is all packaged into a *HKWorkout* object. Heart rate, distance, and energy burned

```
let dateOfBirthType = HKObjectType.characteristicTypeForIdentifier(
    HKCharacteristicTypeIdentifierDateOfBirth)!

let biologicalSexType = HKObjectType.characteristicTypeForIdentifier(
    HKCharacteristicTypeIdentifierBiologicalSex)!

let weightType = HKObjectType.quantityTypeForIdentifier(
    HKQuantityTypeIdentifierBodyMass)!

let energyType = HKObjectType.quantityTypeForIdentifier(
    HKQuantityTypeIdentifierActiveEnergyBurned)!

let heartRateType = HKObjectType.quantityTypeForIdentifier(
    HKQuantityTypeIdentifierHeartRate)!

let walkingRunningDistanceType = HKObjectType.quantityTypeForIdentifier(
    HKQuantityTypeIdentifierDistanceWalkingRunning)!

// types app can create and save to health kit
let writeTypes = Set(
        arrayLiteral: HKObjectType.workoutType(),
        energyType,
        walkingRunningDistanceType,
        heartRateType,
        weightType
    )

// types app can read from health kit
let readTypes = Set(
        arrayLiteral: dateOfBirthType,
        biologicalSexType,
        weightType,
        HKObjectType.workoutType(),
        energyType,
        walkingRunningDistanceType,
        heartRateType
    )

// request authorization
healthStore.requestAuthorizationToShareTypes(writeTypes, readTypes:
    readTypes) { success, error in
    completion(success: success, error: error)
}
```

Figure 4.10: Health information authorization request.

data are all of type *HKQuantitySample*, and are added to the *HKWorkout* object. The *HKWorkout* object saves the *startDate*, *endDate*, *duration*, *totalEnergyBurned*, *totalDistance*, *device*, and *metadata*. The samples retrieved during the workout can also be added to an *HKWorkout* object before it is saved. The metadata saved to the workout is the *WorkoutSessionContext* to provide information used to reconstruct a workout from memory properly for competition challenges. Management of saving, deleting and reading workouts is all done through the *HealthDataManager*. There is the *HeartMateWorkoutManager* that assists in reading and recreating the workout data that is returned from the *HealthDataManager* in the form of a *HeartMateWorkout*. The *HeartMateWorkout* object encapsulates the *HKWorkout* object from *HKHealthStore*, along with the *WorkoutSessionContext*, sample data, location data, and workout statistics. Reading in the workout from the *HeartMateWorkoutManager* provides easily accessible data for displaying the information to the user, as well as for competitions.

Location Manager

If the user has selected to perform the run outdoors the location data is collected during the run. The location data provided is done through the use of the *CLLocationManager* provided by the framework, Core Location. The *CLLocationManager* allows options to set accuracy, as well as the activity type the locations are associated with in the case of HeartMate, the accuracy is set to best, and the activity type is set to fitness. Location updates from *CLLocationManager* is initiated when *startUpdatingLocation()* is called, and location updates are provided by the handler, *didUpdateLocations*. These locations are of type *CLLocation*, which provide coordinate (latitude, longitude) information. Since, the location accuracy is set to best, location information will update at a quick pace. The distance between each point is measured to keep track of distance the user has traveled. When a workout is completed the *stopUpdatingLocation()* function is called on the *CLLocationManager* object.

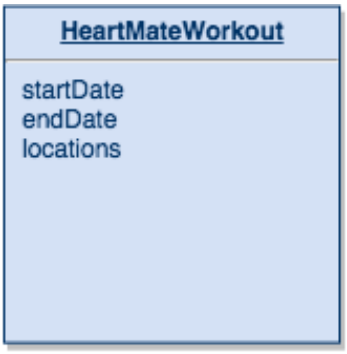


Figure 4.11: The *HeartMateWorkout* entity representation in Core Data.

This data cannot be saved as apart of the *HKWorkout* and is instead saved through the use of Core Data. To save using Core Data, an entity must be created that serves as the structure of the object to be saved. The entity has a title, and contains the attributes associated with the object. The location data is saved to a *HeartMateWorkout* entity which can be seen in Figure 4.11. When the *HKWorkout* is retrieved from *HKHealthStore* and is read in through the *HeartMateWorkoutManager* the workout session context read from the metadata provides information on if the workout was outdoors, and if it was outdoors the location data is retrieved through the Core Data API.

The location data is stored as an array of locations, and in the order of which they were received. When the user wishes to view the map of the route taken, the location data will be reconstructed, and a path is added as an overlay using Map Kit. The location data is fed through the Map Kit API to create a polyline and then overlayed on the map.

Workout Zone Manager

The *WorkoutZoneManager* is where the levels of intensity a user or an opponent to the user, is being tracked and managed. Also, managed by the *WorkoutZoneManager* are the voice updates to a user of the current zone they or their opponent reside in during a workout session. The *WorkoutZoneManager* calculates zone intensity by using techniques and calculations discussed in Chapter 2. There are five zones a user

and an opponent could possibly be in during a workout such as none, low, moderate, high, and red line. These zones should look familiar as they correlate to the ones shown in Table 2.1 from Chapter 2.

The *WorkoutZoneManager* is initialized to the user's health information either provided by Health Kit or entered by the user directly into the application. Utilizing the user's health information such as resting heart rate (HR_{rest}), and age to calculate the max heart rate (HR_{max}) using Equation 2.2. The heart rate reserve (HRR) is then calculated to calculate each of the aforementioned zone thresholds. This is done by using the Karvonen method, Equation 2.7. If the user has an opponent the same process is applied creating zones specifically matching the opponent's health data. The current heart rate value for either the user or opponent is processed each second to determine which zone the heart rate value falls under, using the calculated zones for the specific users.

In order for a user to be considered in a specific zone the heart rate values must consecutively fall within the same zone for at least 15 seconds, otherwise it will not trigger a zone switch or an update. When a zone switch or update does occur the user will be notified by voice with the current zone they switched to. If they are in competition mode the user will be notified when their opponent switches zones along with their current zone at that time. The voice update system is using AVFoundation and MeidaPlayer to create a text to speech for the update. The MediaPlayer is used to determine if a user's music is playing in the background and through AVFoundation lower the audio to provide a more audible update while music is playing. Progress is also provided to the user and their opponent through the graphical interface by using DPMeterView, where each zone increase is a 20% increase to their progress meter.

Since heart rate values are being processed every second the amount of time spent in a zone is calculated and a counter for the specific zone is incremented. At the end of the workout session the zone counters will be used to determine how long a user spent in each specific zone. These time spent counters are also used to create a score between the user and the opponent. If the user is in competition mode the

WorkoutZoneManager also tracks how long the opponent and user were in zones higher than each other. This score is also created through the use of a counter as it equates to the time spent in a higher zone. This is so a user can be shown by how many seconds they were beaten.

4.4 Competitions

4.4.1 Overview

Competitions are the foundation of HeartMate, and provide users ways to challenge themselves from a previous workout or a real time challenge between a friend. This section will discuss the *CompetitionManager*, which handles the competition between a user's or a friend's previously stored workout, or the connection between two devices for a real time competition using PubNub. The system components for tracking a real time workout can be seen in Figure 4.12.

4.4.2 Competition Manager

The *CompetitionManager* contains a *HeartMateWorkoutManager*, *HeartMateOpponentWorkout*, *WorkoutZoneManager*, *pubConfiguration*, *pubClient*, *channel*, *friendChannel*, and *channelGroup* object. The *HeartMateWorkoutManager* assists in preparing a previously stored workout for competition such as gathering the workout session context, and heart rate sample data. The workout session context is important because it provides information required by the *WorkoutZoneManager* to initialize the opponent zones properly, and the heart rate sample data is used as the performance measure. The *HeartMateOpponentWorkout* provides the context to the type of competition such as a real time workout, or a competition to a previous workout. In the *HeartMateOpponentWorkout* the opponent is represented in the form of a parse *PFUser* which corresponds to a real time competition. *HeartMateWorkout* signifying a previous workout of the user's, or *dbWorkoutData* representing a friend's previously stored workout. The *pubConfiguration*, *pubClient*, *channel*, *friendChannel*, *channel-*

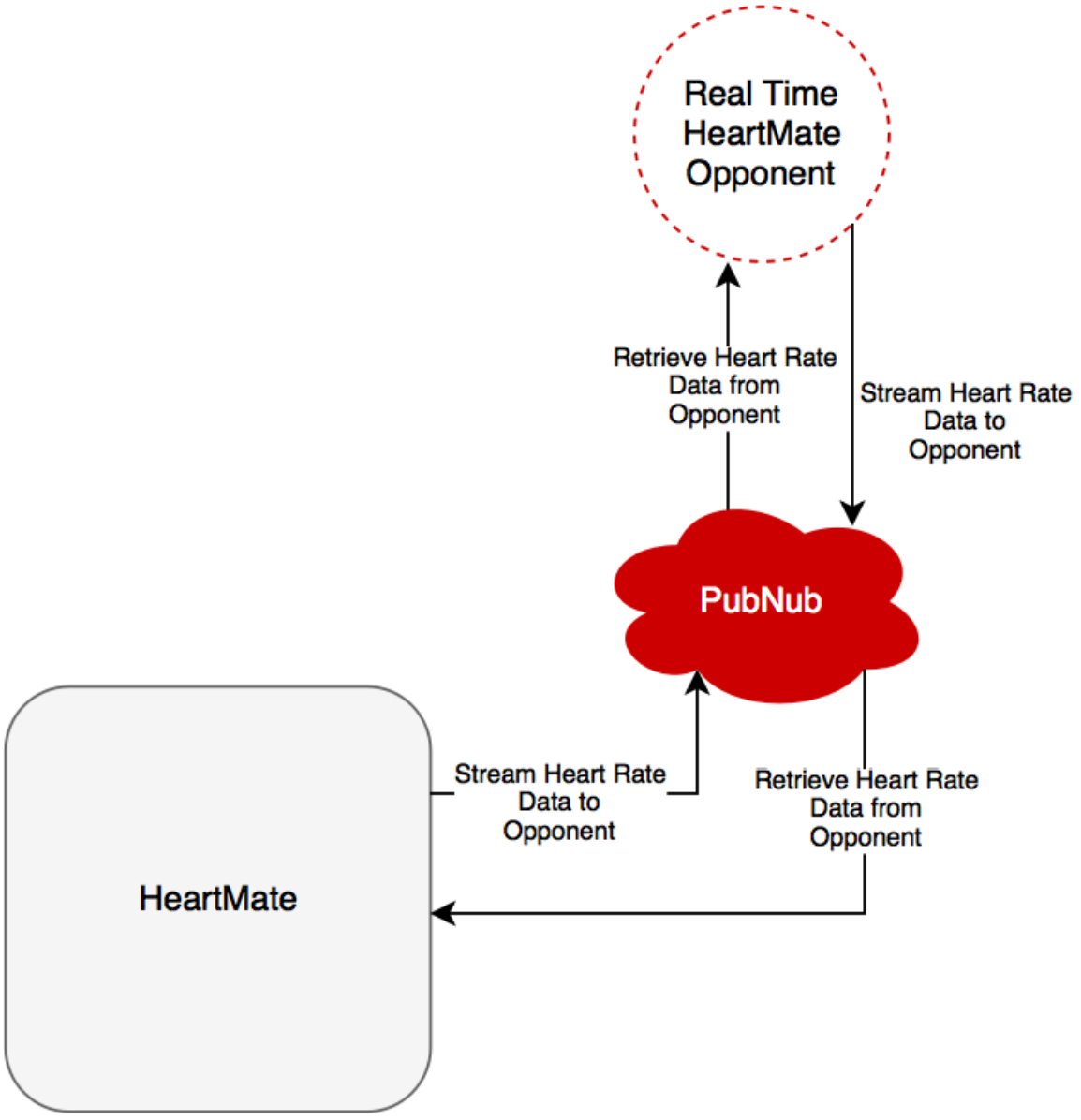


Figure 4.12: HeartMate's interaction with system components to track a real time competition.

Group object are part of the real time opponent competition and will be discussed in the real time opponent section.

Previous Workout Opponent

The competition structure of a previous workout is shown in Figure 4.13. Where **User A** is competing against **User B's** previously stored workout. **User B's** heart rate samples from the previous workout is stored in the heart rate array and is used as the performance measure during the competition. The age and resting heart rate of **User B** is also used to provide context for the heart rate samples stored in the heart rate array to calculate **User B's** zones (opponent zones) at the time of that particular workout. **User A's** heart rate data is collected from the connected Bluetooth LE heart rate monitor and their zones are calculated based off their current resting heart rate and age. Each second, a heart rate is collected from both **User A** and **User B**, which is then processed to determine which zone the heart rate falls under. Once the zone is determined the time spent for that particular zone is then incremented as each heart rate reading corresponds to one second. A user must be in a zone for at least 15 consecutive seconds similar to that of a regular workout being tracked. A user is scored by using the time spent in a higher zone than their opponent where each second is a point added to the user's score. A user will be updated when the opponent either increased or decreased zones along with the current user's zone.

When competing against a previously stored workout the *HeartMateOpponent-Workout* object is set using either a *HeartMateWorkout* or *dbWorkoutData*. If the *HeartMateWorkout* is set as the opponent the workout session context is used as the current user's workout configuration such as duration, but the location of the workout is still configurable. The duration is not configurable because competitions rely on this information as the heart rate data is produced every one second to match the duration of the workout. Heart rate data from a heart rate monitor is read every one second and every same second a heart rate value is read from the opponent (*Heart-MateWorkout* or *dbWorkoutData*). The workout session context is also used to match

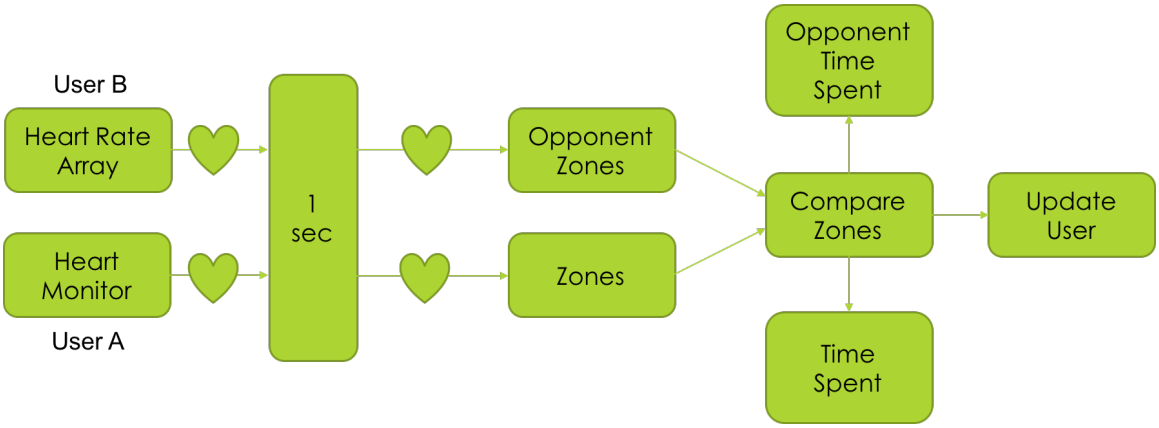


Figure 4.13: The competition structure when using a pervious workout as an opponent.

the fitness level of the user's previous session as it contains the resting heart rate, and age of the opponent at the time of the previous workout. This information is used to set the *WorkoutZoneManager* of the *CompetitionManager* for accurate zone tracking. If the opponent is from a friend's previous workout retrieved from the database, then *dbWorkoutData* will be in the form of Figure 4.6, which provides the friend's resting heart rate, and age at the time of the stored workout. In either case, the heart rate data is read from an array where each heart rate value is indexed by the current second in the workout, and is passed through the *WorkoutZoneManager* for zone processing.

Real Time Opponent

The structure of a real time opponent is similar to that of a previous workout opponent, but the opponent's heart rates are not stored in an array and is streamed directly from the opponent as shown in Figure 4.14. PubNub is used for a real time competition and creates a publish / subscribe connection between two devices. The Parse *PFUser* object in the *HeartMateWorkoutOpponent* object is only set as it represents the friend being challenged, and provides the necessary information to create a competition. Since a *PFUser* is in the exact structure as shown in Figure 4.3, it provides their current health data information such as resting heart rate and age

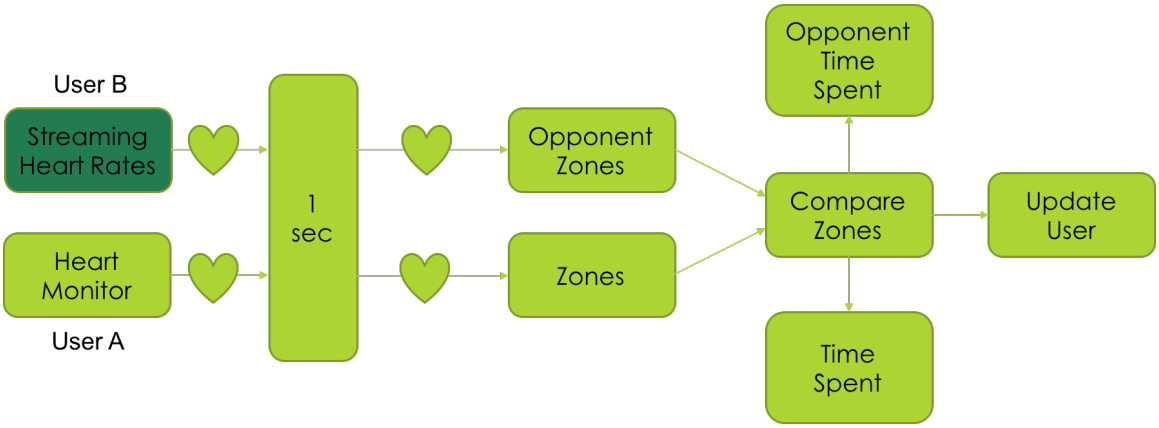


Figure 4.14: Real time opponent structure.

to provide an accurate comparison. The *PFUser* object also provides their unique *objectId* which is used to create a channel.

To setup the connection between devices the *pubConfiguration*, *pubClient*, *channel*, *friendChannel*, *channelGroup* objects are used. The *pubConfiguration* contains app specific information to create a connection between HeartMate apps, and the *pubClient* manages the publisher / subscriber connection between devices. The channel is set to the current user's *objectId*, and *channelGroup* is set to "competition" and through the *pubClient* the *currentUser* is subscribed to the *channel* and added to the *channelGroup*. The *friendChannel* is set to the friend's *objectId*, and this allows the current user to publish their heart rate data to the *friendChannel* all through the *pubClient*. The friend will receive the current user's heart rate data on their *channel*, and will publish their heart rate data on their *friendChannel* which the current user will receive on their *channel* they are subscribed to. The friend is also added to the *channelGroup*. The *pubClient* object has a handler for listening to the channels a user is subscribed to, and the heart rate data is retrieved from the *didReceiveMessage(...)*, handler.

Synchronization is important to maintain that one-to-one ratio when comparing heart rates on each second the two devices need to be in sync. PubNub's presence feature is utilized, and when a user is added to the *channelGroup* the state is updated for all users apart of the *channelGroup*. This information is provided by the *pubClient*

handler, *didReceivePresenceEvent(...)*, when a user joins the *presenceEvent* state is set to "join", and the occupancy total is updated. Through the use of this information a competition is synchronized when the second person added to the *channelGroup* and the *occupancy* is updated to the number 2. Once this occurs a countdown to the workout begins in sync on both devices. This allows for the workout to be started together on both devices, thus making *channelGroup* act as a lobby. The competition tracking is managed and setup on each device the same way. The connection PubNub provides is a path for each user's heart rate data to be shared across the network no matter the location of each user. The heart rate is then processed locally by the *WorkoutZoneManager* and provides updates as if they were competing side by side. If a person leaves the competition, thus setting the *presenceEvent* state to "leave", during a competition the other user is notified through push notification or through the "leave" state.

4.5 Summary of Software Technology and Characteristics

HeartMate is an iOS application developed using Apple's Xcode IDE with the programming language, Swift. Various third party libraries and frameworks along with Apple's own frameworks were used to assist and provide the functionality HeartMate required. Health Kit played a major role in developing HeartMate as it was the source for most of the health data tracking such as gathering heart rates from a Bluetooth LE connected heart rate monitor, and tracking distance for indoor runs. Health Kit was used to store and retrieve workouts locally with the exception of location data which was done through Core Location. Core Location was also used to gather location data during an outdoor workout. Parse provided the server side functionality to HeartMate, while PubNub provided the real time streaming of heart rate data between devices. Libraries such as BEMSimpleLineGraph and Charts were used to display the workout information to the user. During the development of HeartMate a total of 56 classes were created along with approximately 10,000 lines of code to

provide perspective. Currently, HeartMate is not open source as the intention is to release it as a commercial application.

Chapter 5

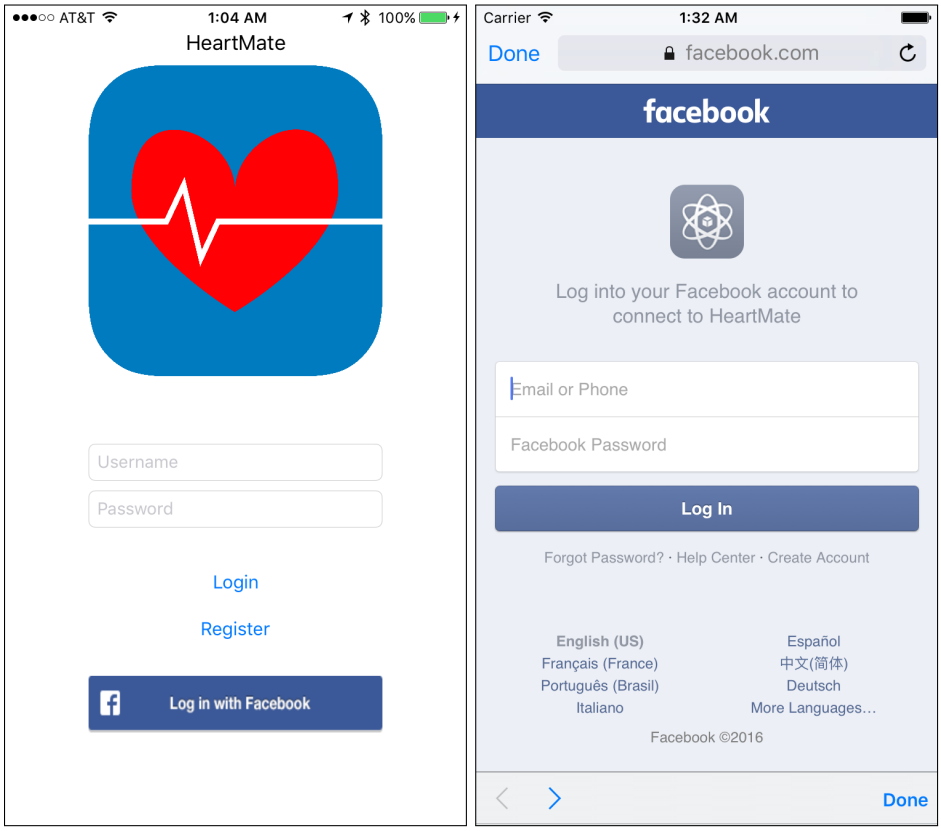
Application Walkthrough

This chapter will serve as a walkthrough of the HeartMate application, by showing the app's interface and describing its functionality.

5.1 Login and Sign Up

When a user launches the application they will be presented with the login and sign up screen as shown in Figure 5.1a. The user is presented with two methods of registration, through the app or linking their Facebook account with HeartMate. If Facebook is chosen, the user will be redirected to the web browser to login to Facebook, and grant permission to HeartMate as seen in Figure 5.1b and Figure 5.2a. When linking a Facebook account their profile picture will be visible showing a successful link, and a username must be supplied by the user to complete the registration process which can be seen in Figure 5.2b. Once the user has completed a successful login they are brought to their main user profile as seen in Figure 5.3, which is also the main menu.

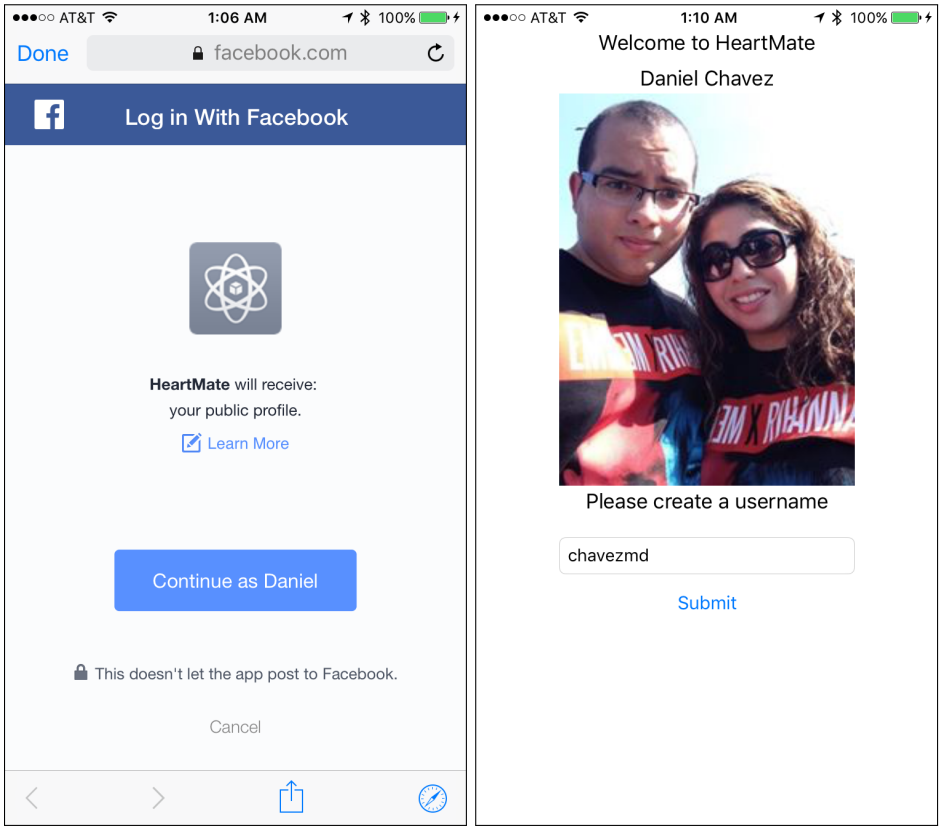
The second option is to create a HeartMate account, and the user's full name, username, and password is requested. Figure 5.4a and Figure 5.4b show the complete process of making a HeartMate account, and once the user is created they are brought to their main profile view, as seen in Figure 5.5. Their picture is represented by a person's silhouette.



(a) HeartMate's initial starting view, login and sign up.

(b) A user must sign into Facebook to link their account to HeartMate.

Figure 5.1: Initial view and user signup through Facebook using the Parse API.



(a) Grant HeartMate access to the user's Facebook account.

(b) Last step is to create a HeartMate username.

Figure 5.2: Final step of linking a user's Facebook account is granting permission, and creating a username for HeartMate.

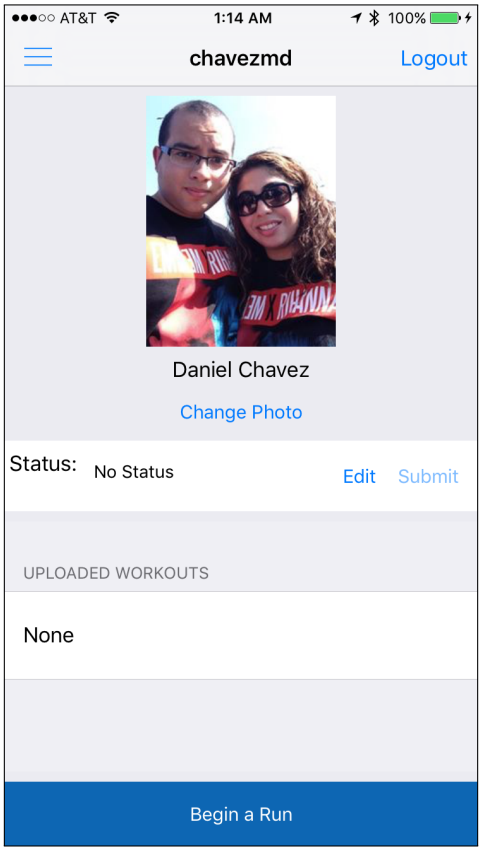
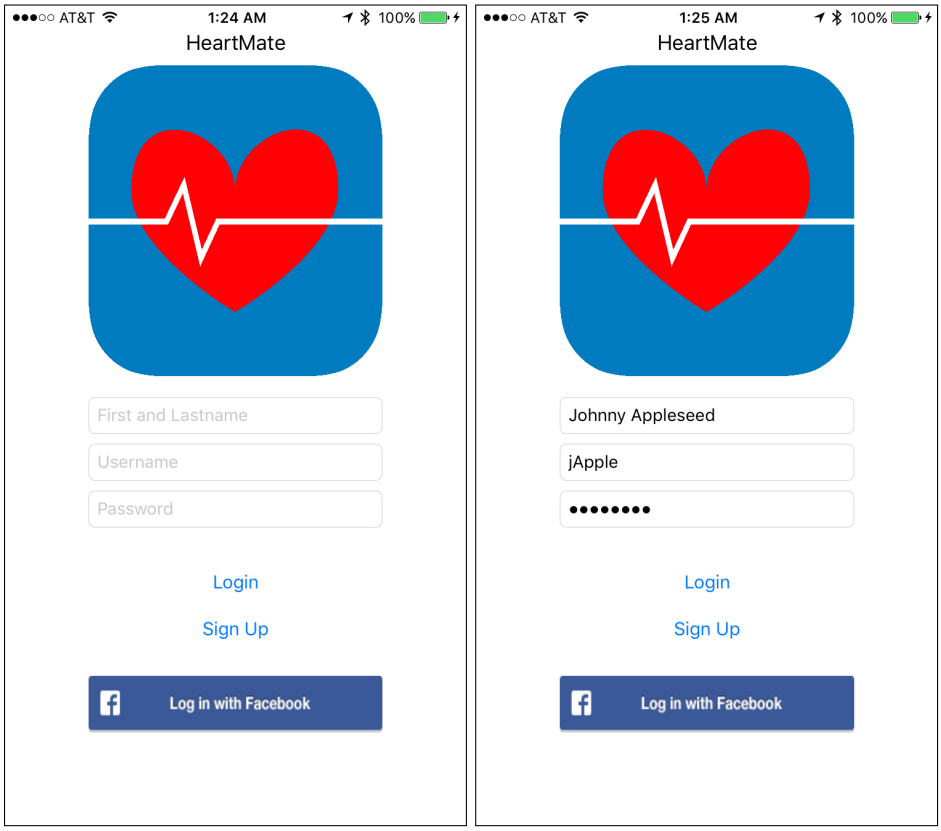


Figure 5.3: The main user profile view, also the view once a user is authenticated.



(a) Account creation requires a name, username and password.

(b) The "Sign Up" button completes the process.

Figure 5.4: HeartMate's normal sign up process.

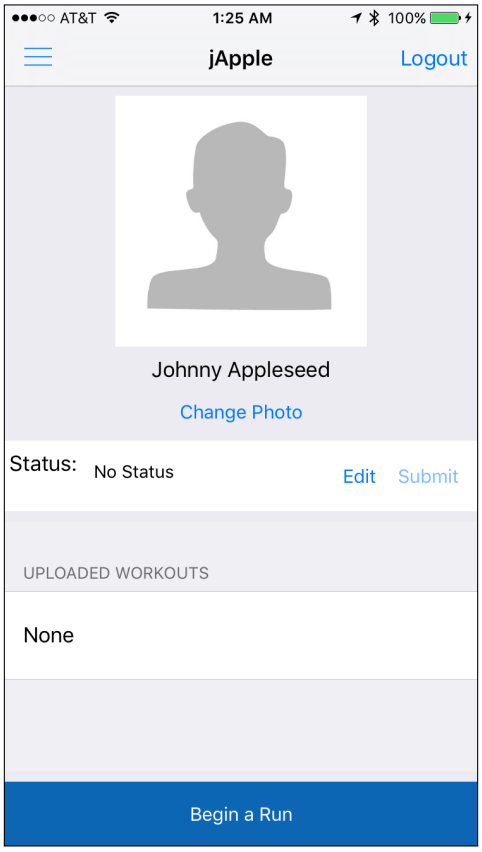
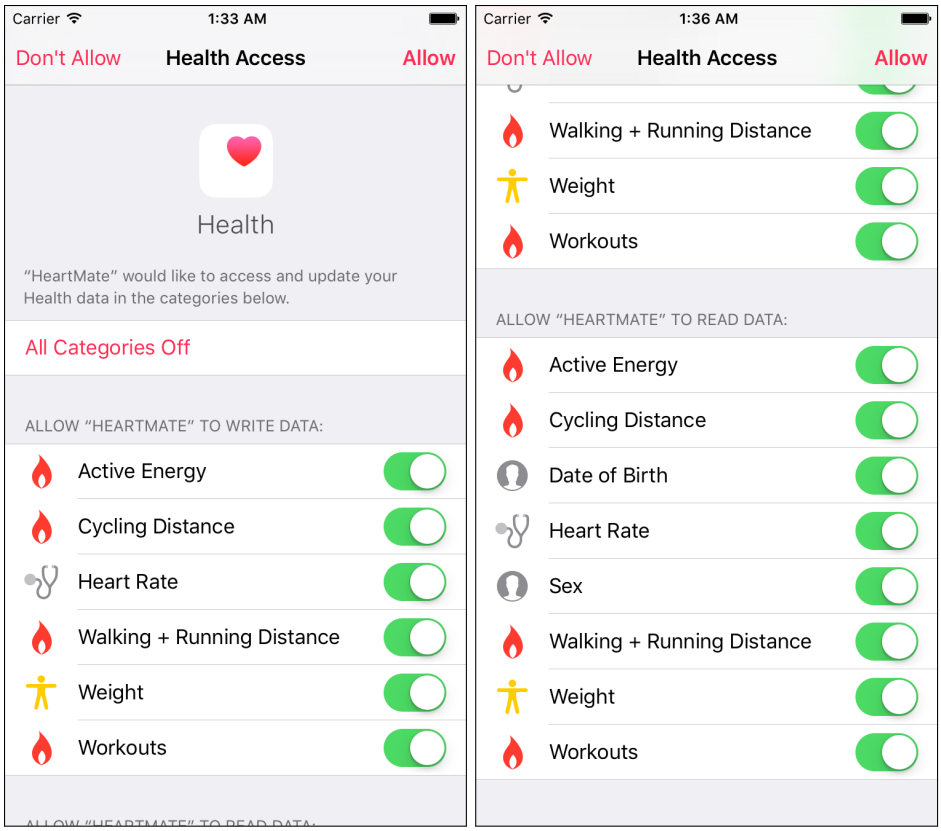


Figure 5.5: The user's main profile view.



(a) The requested data HeartMate can write to the Health app.

(b) Requested data to be read from the Health app.

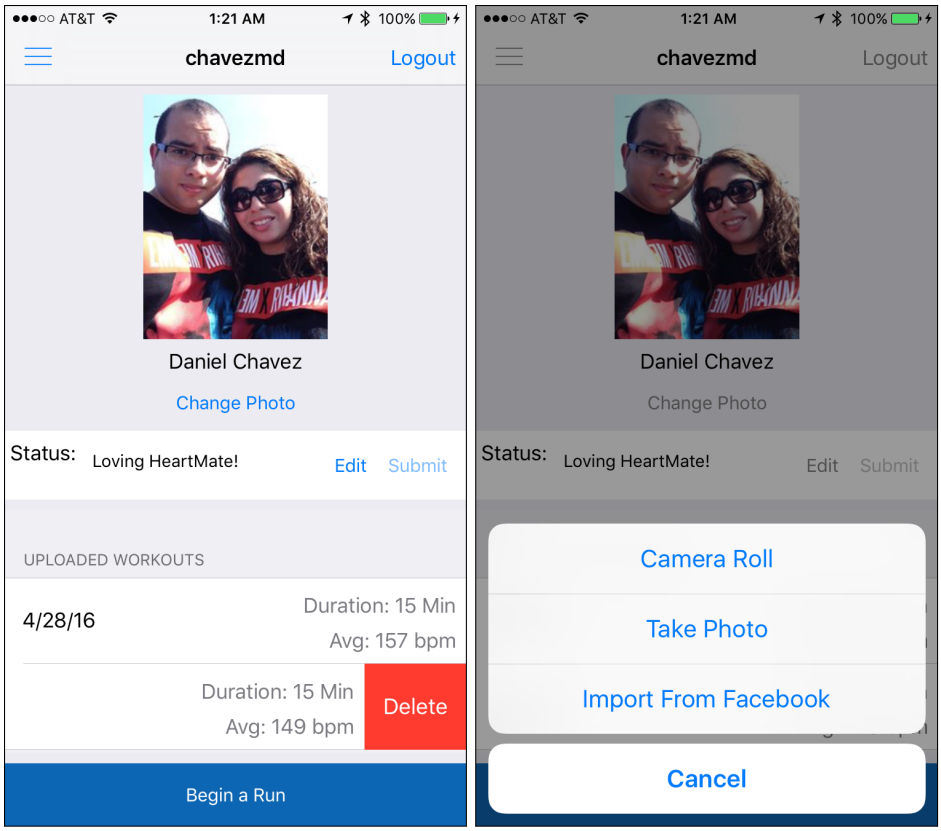
Figure 5.6: Health Kit authorization request.

5.2 Health Authorization

Since HeartMate uses Health Kit for a variety of operations such as reading and writing health data to and from the Health app the user has to authorize HeartMate to access this information. HeartMate requests Active Energy, Heart Rate, walking and running distance, weight and workouts. Figure 5.6 shows how the health authorization request is presented to the user, and is requested after the first time the user logs in.

5.3 Edit Profile

When the user is at the main menu they are shown their HeartMate profile. Their HeartMate profile is visible to their friends, and consists of their status, image, and



(a) Removal of uploaded workouts through a swipe.

(b) Importing images is possible from Facebook, camera, and device.

Figure 5.7: An example of editing a user's HeartMate profile.

uploaded workouts their friends can challenge. The user can change their status, and manage their uploaded workouts as well as logout from the app which is shown in Figure 5.7a. The user can change their photo by importing one from their local device (camera roll), take a photo using their device, and if they have linked their account with Facebook they can import their current profile picture all of which can be seen in Figure 5.7b.

5.4 Menu Navigation

The menu navigation is accessed through the use of a three bar horizontal line icon located at the top left of the application. This menu icon appears on each view in the same location. Once the user presses this button they are able to access the

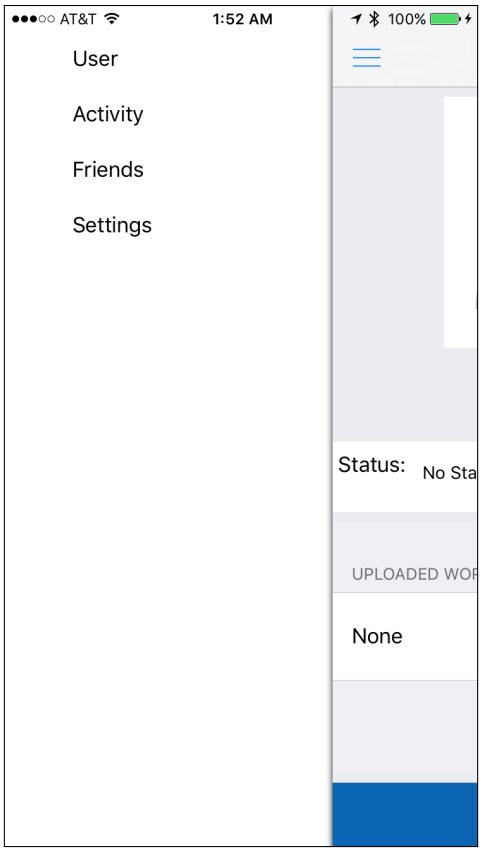
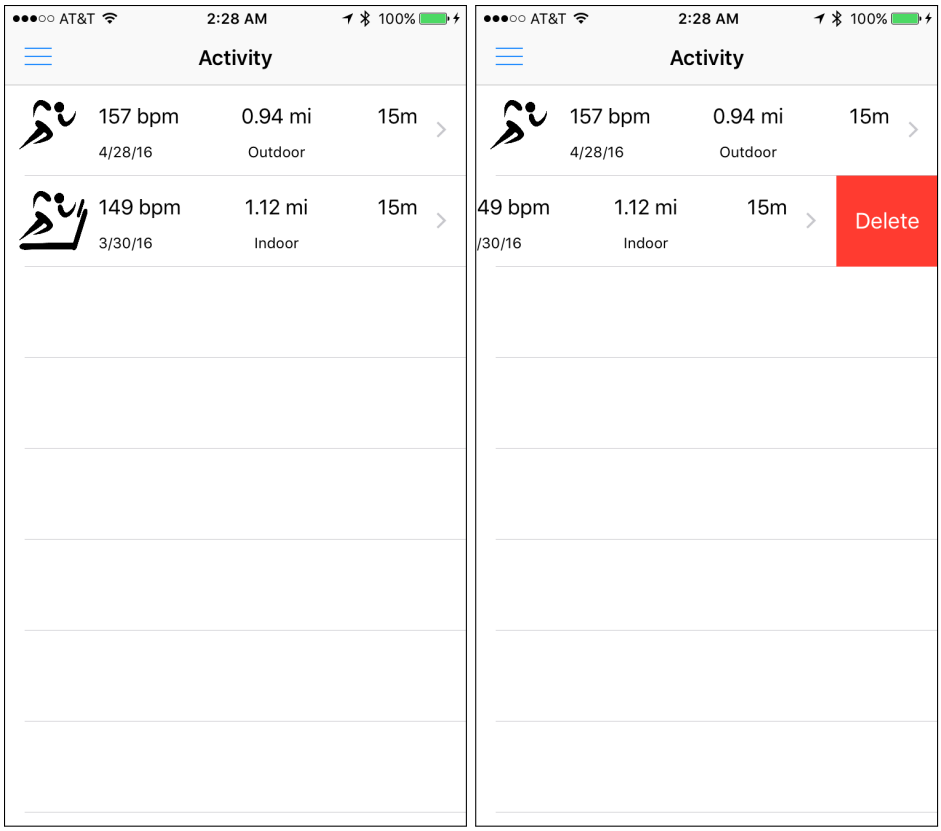


Figure 5.8: The menu options of HeartMate implemented using SWReveal.

different menu options to navigate HeartMate. The menu slides out from underneath the current view, and was created this way to maximize the space of each view. Figure 5.8 shows how the HeartMate navigation menu behaves.

5.5 Viewing Workouts

HeartMate allows a user to view their previously saved workouts stored on the device in the form of a list as seen in Figure 5.9a. The list provides a quick view of some of the workout stats. Since the HeartMate app only supports the exercise running, the icons that denote the activity differentiate the location by showing a runner with a treadmill for indoors, and only a runner for outdoors. A user can manage their workouts by adding or removing a workout from their device by swiping their finger across revealing a delete button as seen in Figure 5.9b.



(a) List of local workouts stored using Health Kit.

(b) Removal of a local workout through a swipe.

Figure 5.9: Viewing locally saved HeartMate workouts, and their details.

When a user selects a workout from the list they are taken to a new view which shows the workout details for that particular workout. A graph of the heart rates are shown, and the user can drag their finger across the data to view what heart rate value and zone they were in at a particular time. An example of this is in Figure 5.10a. The user can also view a pie chart of their intensity levels during the workout, and is shown in Figure 5.10b. Information such as duration, distance and calories is also provided.

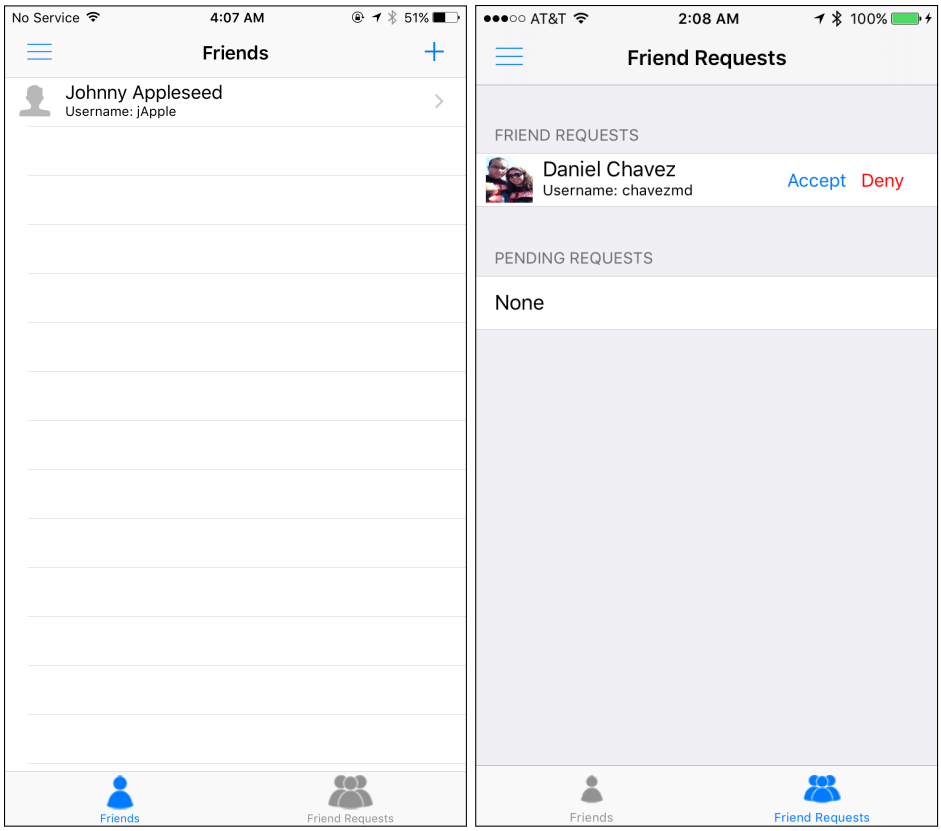
If the workout was performed outside, an arrow indicator will appear next to the location name. Once pressed, the user is taken to an embedded map view with a path of the route. This was implemented with Map Kit with an added polyline overlay to represent the route. The map view can be seen in Figure 5.11.

(a) Heart rate graph, and workout statistics.

(b) Zone intensity pie chart in the same view as the heart rate graph.

Figure 5.10: Viewing locally saved HeartMate workouts, and their details.

Figure 5.11: Map view of the user's running route.



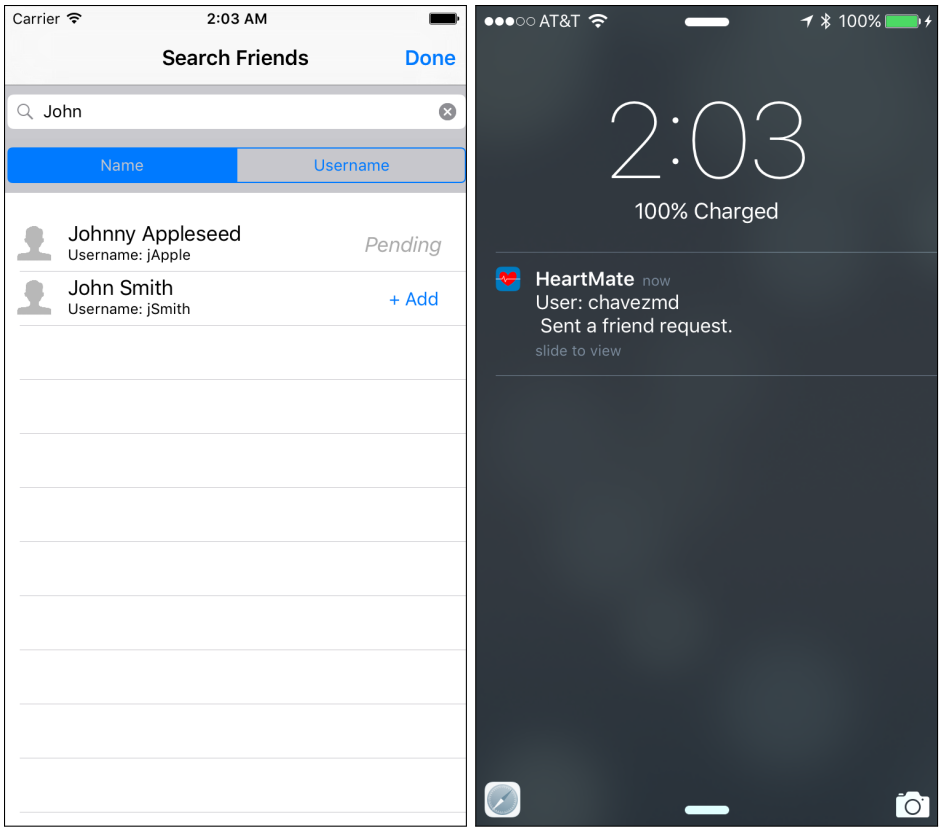
(a) Friends list view. (b) Friend requests view.

Figure 5.12: Viewing HeartMate friends and requests.

5.6 Friends

Friends located under the "Friends" menu option provides a list of friends and a way to manage friend requests. The first presented view is their friends list, where they can view their friends in the form of a list as well as add friends. The second option is the friend requests view and allows management of those requests. Both options can be viewed in Figure 5.12a and Figure 5.12b.

A user can search and add friends by either their username or full name. If the user they are searching has already been requested then they will not be able to add them again, which is indicated by a "Pending" status. A check mark indicates the user is already friends with searched user. HeartMate's handling of searching of and adding friends can be seen in Figure 5.13a. Once the user requests to be friends



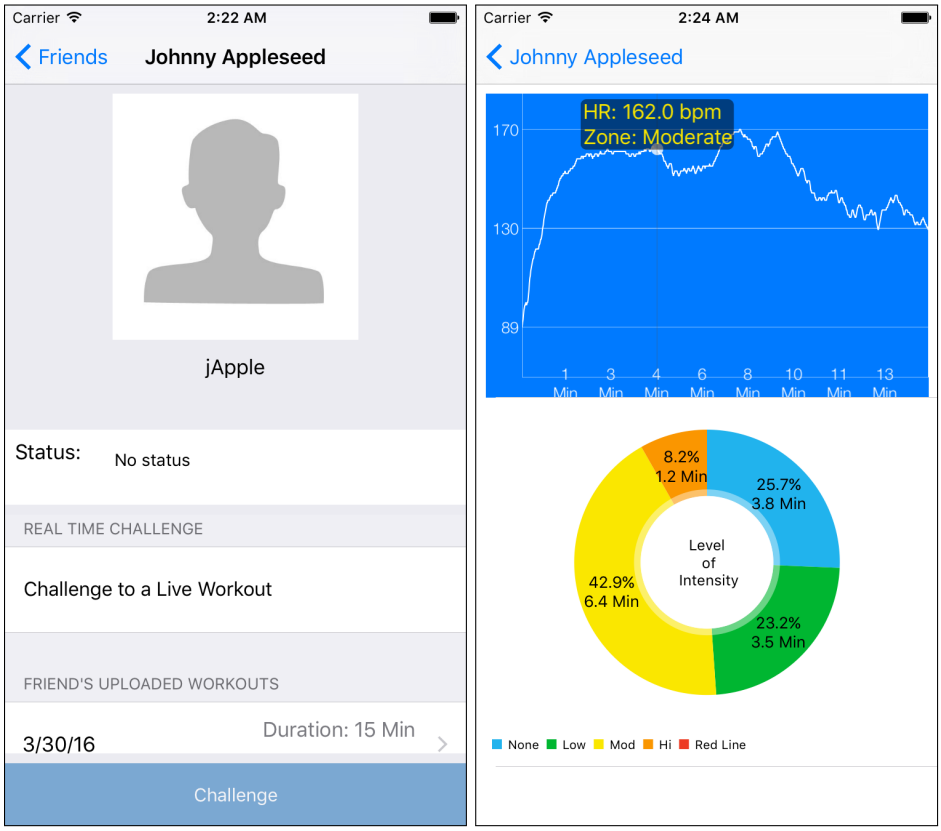
(a) Search and add friends by a name or username.

(b) A push notification is sent to the user requested.

Figure 5.13: HeartMate searching and adding friends.

with a particular user a push notification is sent to the requested user as seen in Figure 5.13b.

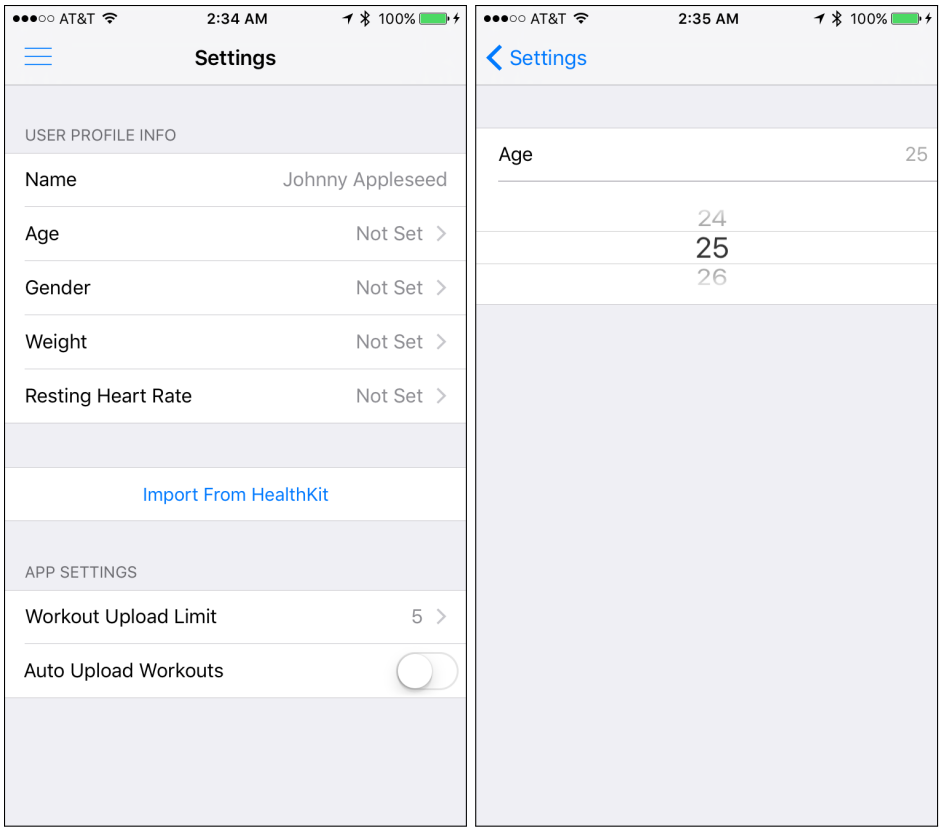
Through the friends list view the user can view their friend's HeartMate profile. The friend profile view is similar to the main profile view of the user, but is not editable. Options to challenge the user's uploaded workouts and to a real time workout is available as seen in Figure 5.14a. Details to a particular uploaded workout can also be viewed before challenging, and provides insight to the user's performance measure. Graphs of heart rate, and zone intensities are shown. An example of a friend's uploaded workout can be seen in Figure 5.14b.



(a) Status, profile picture, previous workouts can be viewed.

(b) Friend's workout information can also be viewed.

Figure 5.14: Viewing a friend's profile.



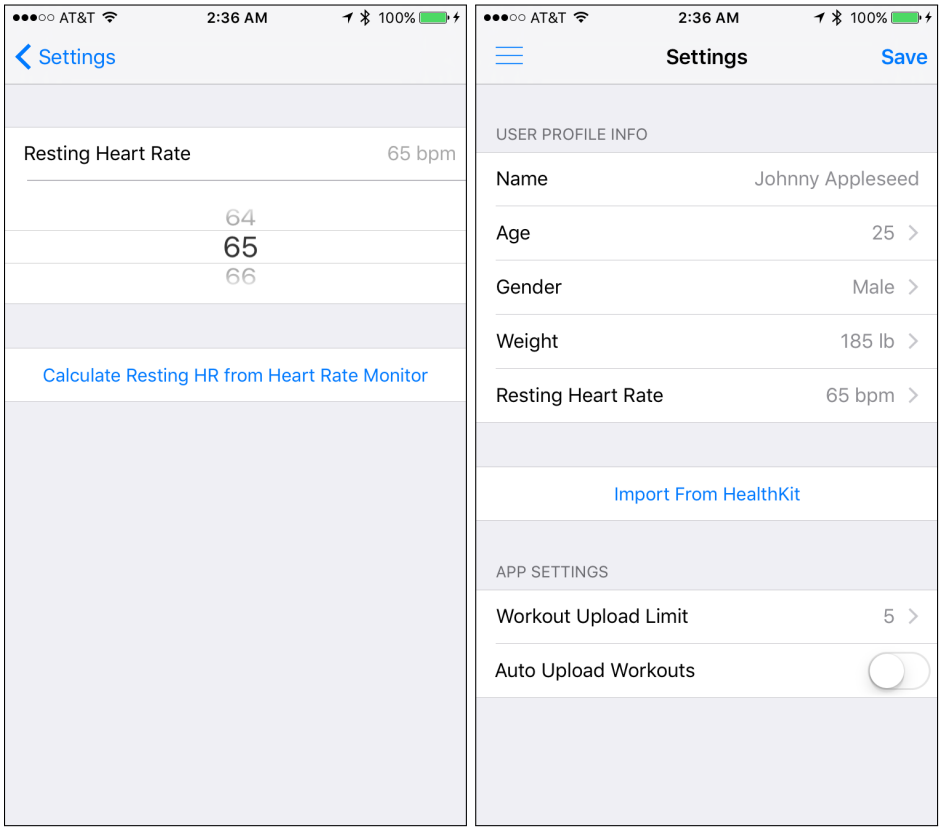
(a) Health information used for zone information.

(b) Age, gender, weight, and resting heart rate are inputed similarly.

Figure 5.15: Profile settings with a method to import information from Health Kit.

5.7 HeartMate Settings

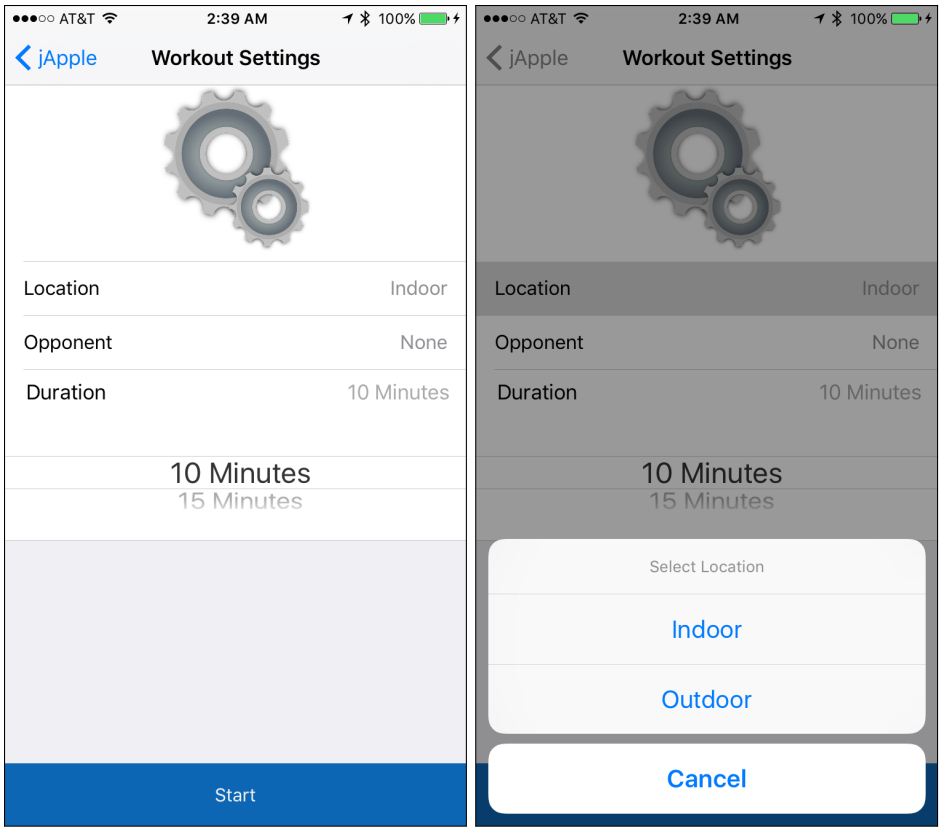
Settings for HeartMate is a critical aspect in accurately tracking a user's workout and their zones. Under the "Settings" menu option, a user can input their age, gender, weight, and resting heart rate. The user also has the option to import the information from the Health app. Automatic upload of workouts can also be set under the "App Settings" category. The settings view can be seen in Figure 5.15a, and how a setting is set in Figure 5.15b. Importing of a user's resting heart rate can be done through the use of a connected Bluetooth monitor. The heart rate is averaged for twenty seconds, and set to the correct value as seen in Figure 5.16a. Once the information is inputted the user must save this information to their profile as seen in Figure 5.16b.



(a) Resting heart rate has the option to calculate from a heart monitor.

(b) A "Save" button appears if the information has changed.

Figure 5.16: Profile settings with a method to import information from Health Kit.



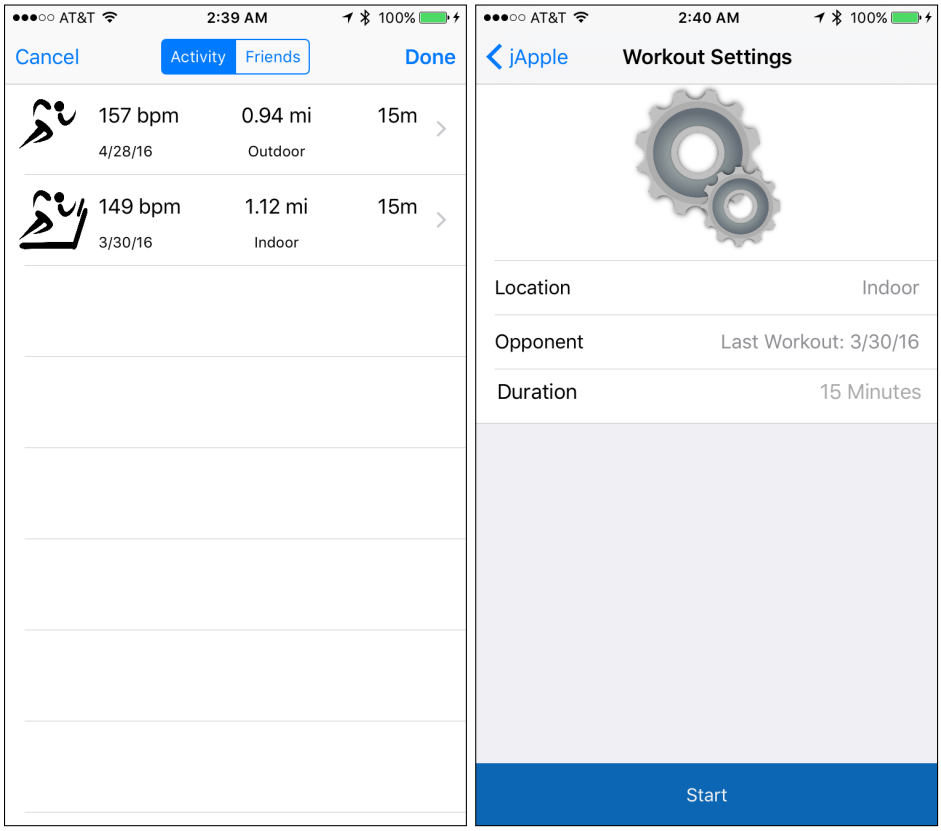
(a) A user's inputted workout settings.

(b) User has the choice between an indoor or outdoor run.

Figure 5.17: The different workout configurations of HeartMate.

5.8 Workout Settings

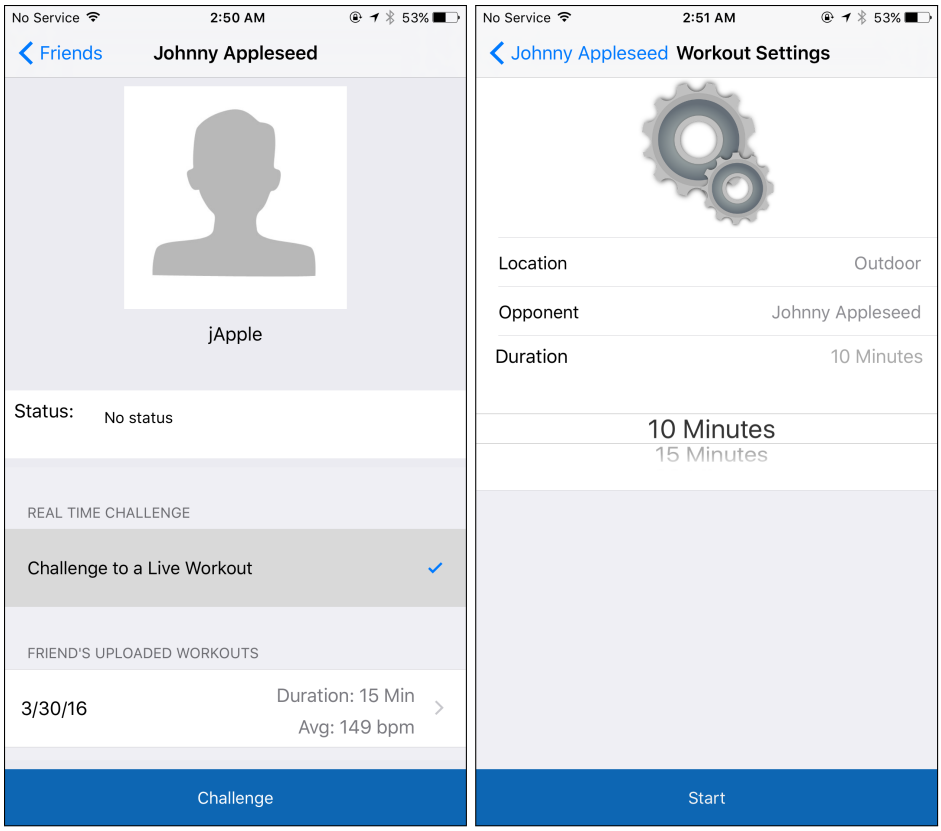
The workout settings define the configuration of the run or competition. A user must select a location, opponent, and duration. The settings view is shown in Figure 5.17a, and input of a location in Figure 5.17b. If a user selects an opponent, an opponent selection view will be shown to the user to select a previous workout of their own or friend's, and the ability to challenge a friend live is as shown in Figure 5.18a. If a previous workout is selected the duration is restricted to the previous workout's duration to maintain the one-to-one comparison of heart rates as shown in Figure 5.18b.



(a) Challenge previous workout or a friend.

(b) Challenging a workout limits duration configuration.

Figure 5.18: The different workout configurations of HeartMate.



(a) Choose the live workout option under a friend's profile.

(b) Define the workout conditions of the competition.

Figure 5.19: Starting a real time competition with HeartMate.

5.9 Real Time Competitions

A real time competition is created either through the friend's view by selecting the option, "Challenge to a Live Workout" as shown in Figure 5.19a, or through the workout configuration view. Once a user selects an opponent they are the ones to define the competition parameters as seen in Figure 5.19b. Once the user is satisfied with the settings they may start the workout. Once the user has pressed start the user must wait for the opponent to accept and start the workout as shown in Figure 5.20.

The chosen opponent will receive a push notification as an indication they are being challenged. The push notification alert can appear anywhere on the device, and alert the user they have been challenged and can accept or decline, which can be seen

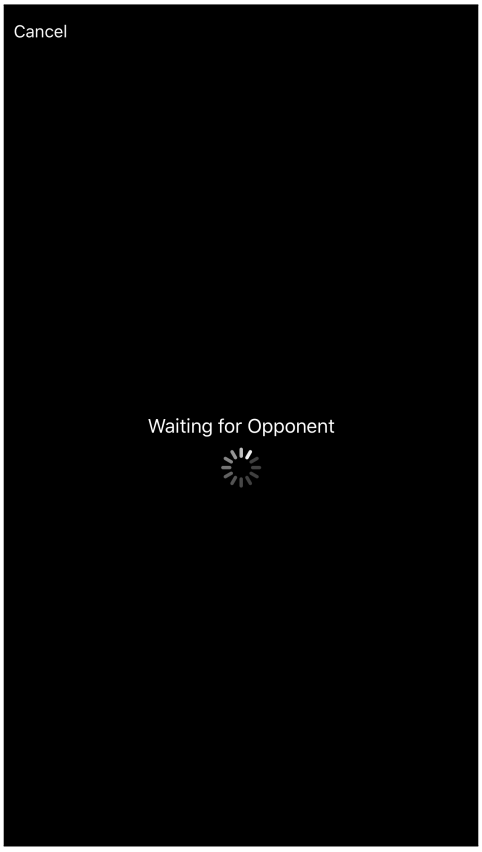
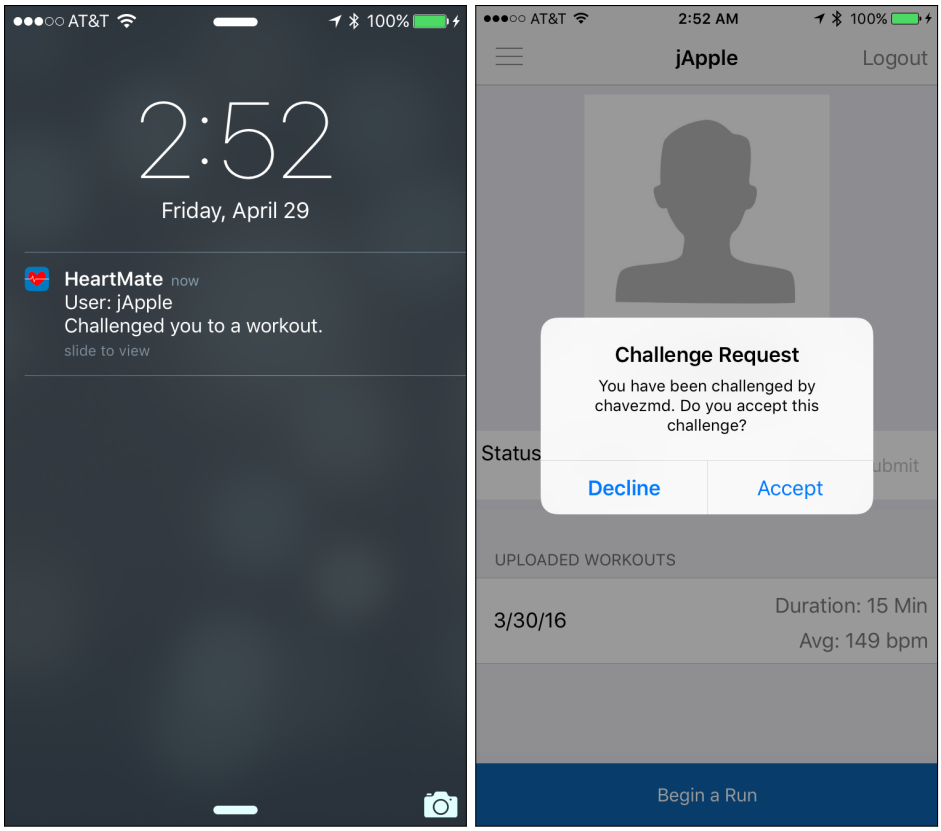


Figure 5.20: Waiting until the other user joins or declines.



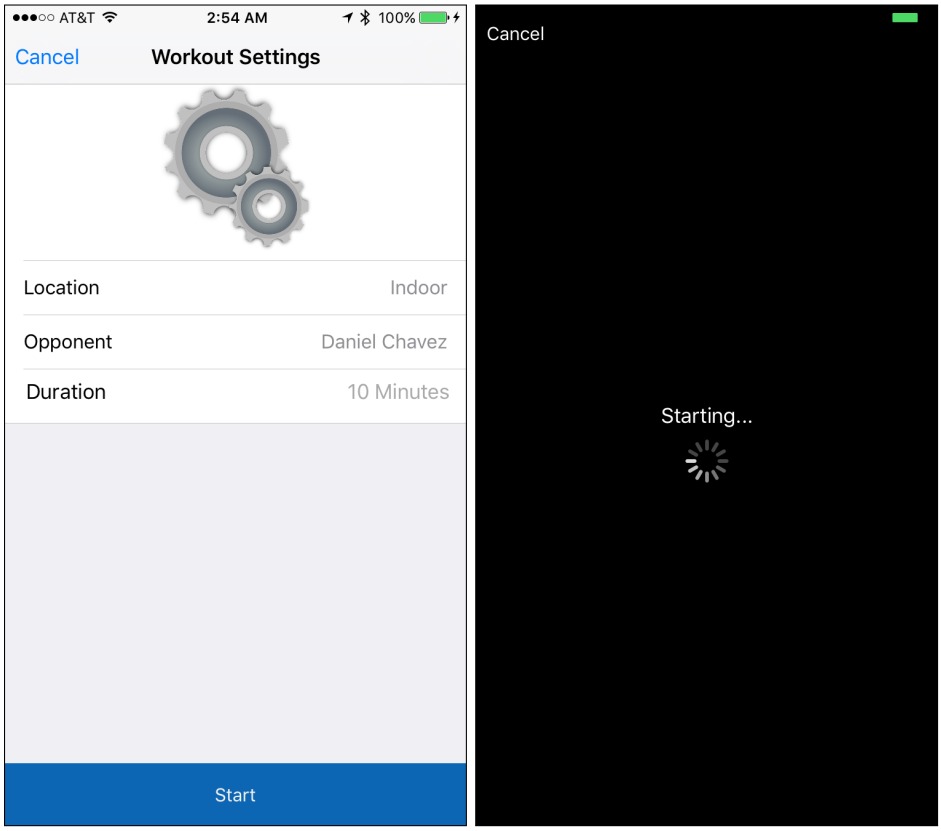
(a) The challenged user receives a push notification.

(b) Once opened the user must accept or decline.

Figure 5.21: The user's perspective of receiving a request for a real time workout.

in Figure 5.21a and Figure 5.21b. Once the user has accepted the challenge they will be shown the competition settings the challenger has chosen with duration restricted as shown in Figure 5.22a. Once the user starts the workout the user is presented with a starting screen as shown in Figure 5.22b. When both users are connected and are notified by the "join" state provided by PubNub the devices will start at the same time and the countdown will start as shown in Figure 5.23.

During the workout the user is presented with their heart rate, zone, calories and miles ran. A progress view of the zone they are currently in as well as their opponent's is shown. The progress view utilizes DPMeterView, and animates during the run. Threshold zones are indicated to provide the user a target heart rate to aim for as well as the current status of their opponent. When the opponent switches



(a) Configuration preset by challenger.

(b) Joining lobby, and will start when both users are in group.

Figure 5.22: The user's perspective of receiving a request for a real time workout.

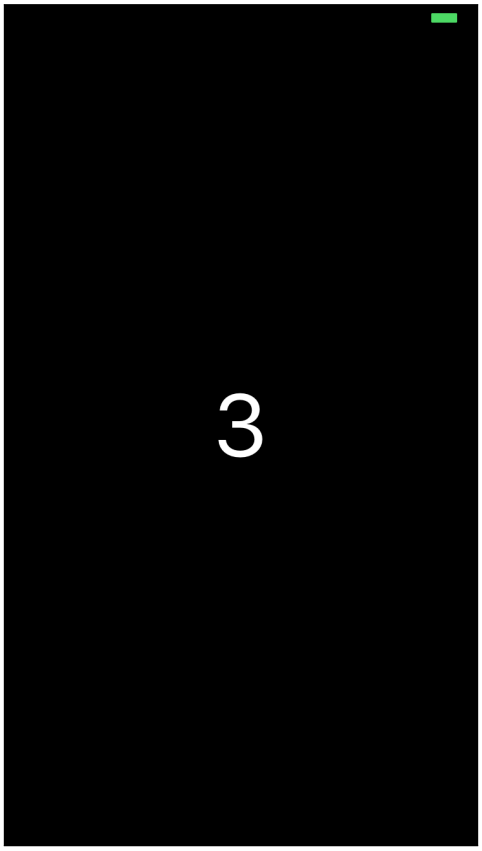
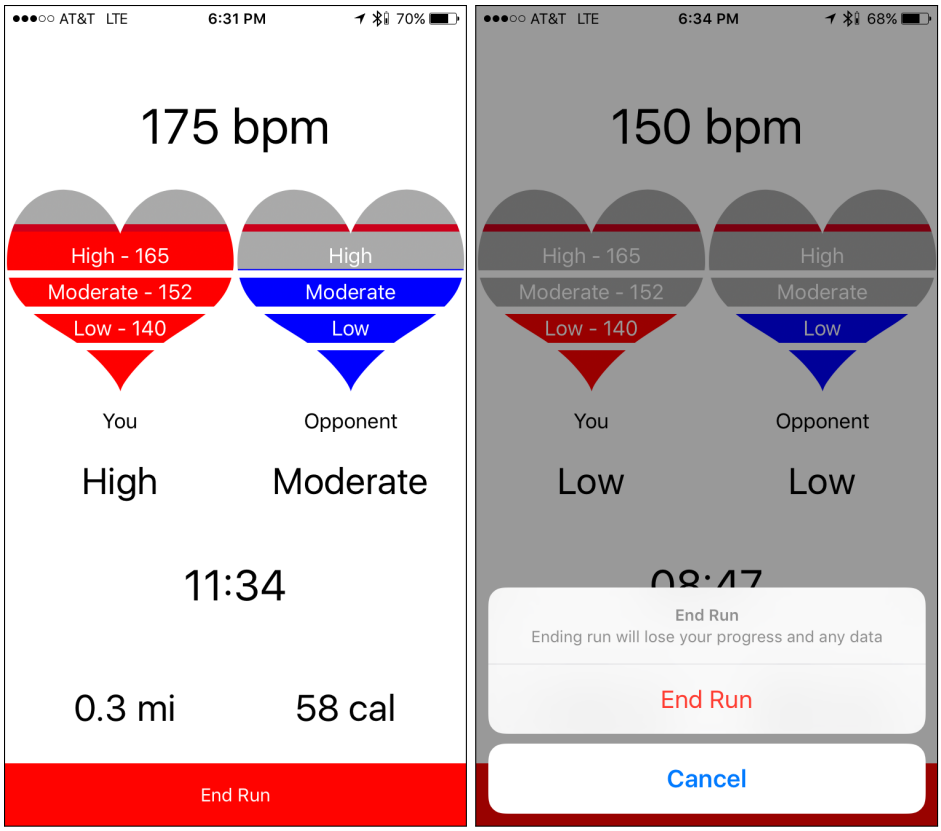


Figure 5.23: The real time workout then begins counting down on both devices at the same time.



(a) Comparison of a workout being challenged.

(b) Warning the user of lost data for ending a run early.

Figure 5.24: Active workout view during a HeartMate competition.

zones the user is notified through a voice update and alerts them of the zone they have decreased or increased to, and the current zone they are currently in. The active workout view can be seen in Figure 5.24a. The active view is similar to a regular workout session with no opponent. If a user ends a competition early they will lose their workout data and is warned as shown in Figure 5.24b. If the user does end the workout early the user will either receive a push notification or a state change through PubNub and will receive an alert as shown in Figure 5.25, resulting in their workout ending.

Once the workout has completed the results of the workout will be shown along with the map of the route taken as shown in Figure 5.26 and Figure 5.27. The graphs shown are the same graphs when viewing a stored workout, and provide both the

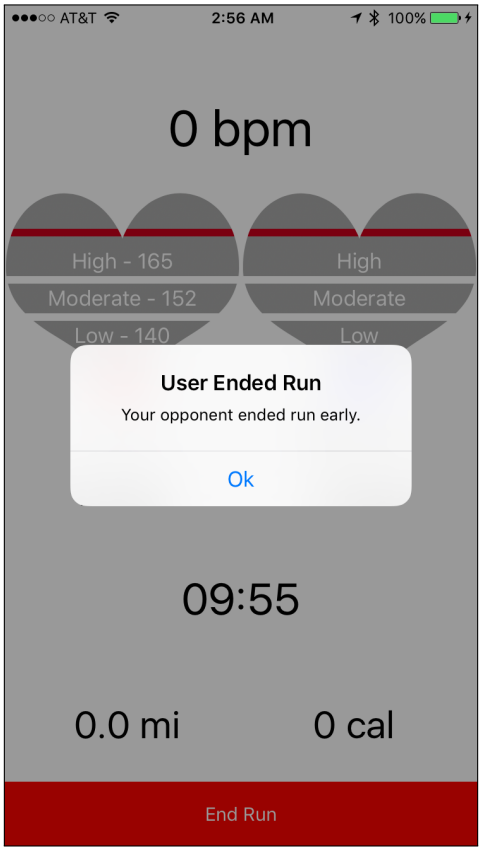
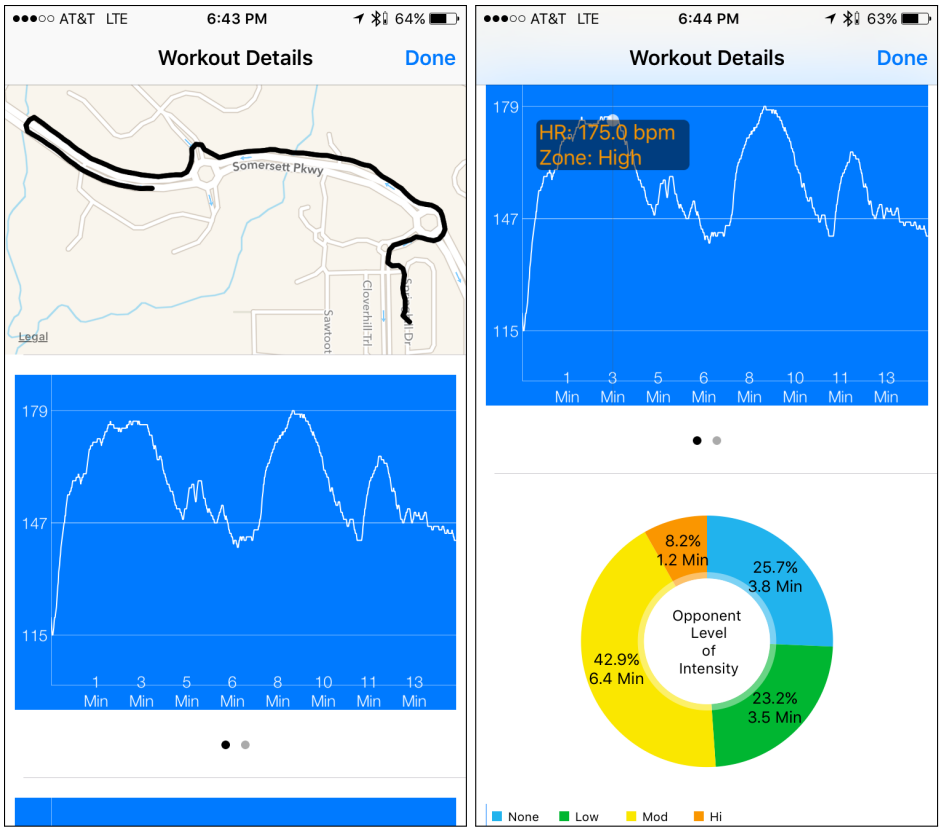


Figure 5.25: State change or Push notification of the competition ending early.



(a) Map view is shown with route; user's heart rate data is displayed.

(b) The opponent's data is displayed underneath the heart rate graph.

Figure 5.26: The view of a completed competition workout.

user's and their oppoonent's data to compare. The user's graphs are shown at the top with the opponent's underneath for side by side comparison as seen in Figure 5.26b. A score is also shown of the opponent and user providing how long each user was in a higher zone than the other. When the user is in a regular run only the normal statistics are shown along with the graph data.

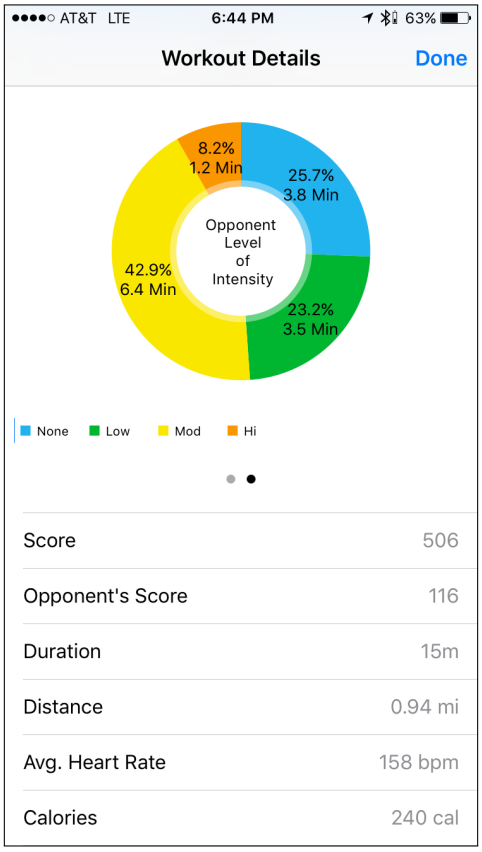


Figure 5.27: The statistics and scoring of the competition.

Chapter 6

Comparison with Related Work

In this section, we compare our HeartMate application with the other fitness applications (Nike+ Move, Nike+ Running, and Fitbit) mentioned in Chapter 2. The competitive and social features of each application is listed along with advantages and disadvantages.

The Nike+ Move application tracks a user's movement continuously throughout the day using NikeFuel as the performance measure. Users do not need to start an activity tracking session as the application will always be tracking the users' movement. NikeFuel is calculated using a proprietary algorithm that generalizes activities converting movement into NikeFuel without the need to specify the particular activity beforehand. Since NikeFuel is the performance measure the user is able to compare their NikeFuel throughout the week, against friends as well as users near their location. Nike+ Move does track the user's location to provide a map of where NikeFuel was earned at specific locations. Nike+ Move does not provide any real time feedback or any other type of motivational feedback to do better in comparison to a user's friends [37].

The Nike+ Running application is dedicated to tracking the activity, running. NikeFuel is calculated with the Nike+ Running application, but is only used to provide a quantified measurement for how much energy the user has expended instead of using it as a performance measure. The amount of miles is instead used as the motivator and performance measure. The user can then compare the total amount of miles ran during the month with their friends instead of comparing NikeFuel. Once again there

is no real time feedback or motivator other than a number comparison. Although, Nike+ Running does have a feature where a user can connect their Nike+ account to Facebook and post they are going for a run to receive "cheers" whenever a friend interacts with the post. This provides another method of motivation pushing users to continue their run knowing their friends are cheering for them. Through the use of "cheers" Nike+ Running does provide that connection between others during a workout to provide a motivational experience. The problem with "cheers" is that someone has to interact with post, and if no one interacts with the post there will be no cheers especially if the run takes place during odd hours of the day. Nike+ Running does have another competitive aspect called, Nike+ Challenges. Nike+ Challenges allow a user to invite their friends to a race in which the user defines a set distance and the first to reach that distance wins. This the closest to a real time competition, but the performance measure is speed. There is a chance the user has friends who are naturally faster, and speed is not a determinate of who is working out harder. Some may also be in better shape, and reach the distance without breaking a sweat. Nike+ Running does utilize the use of a heart rate monitor, but is solely used for the user to track their heart rate during a run [38].

The Fitbit application tracks a user's movement such as walks or runs, but focuses on the amount of steps as the performance measure. Using steps is the equivalent of using NikeFuel in the Nike+ Move app. The amount of steps can be compared between friends and used in Fitbit Challenges. Fitbit Challenges are predefined goals used to compete against friends such as who can take the most steps during the weekend. Fitbit does provide updates to the user such as when they are approaching or have met a goal and is notified through push notifications. The use of push notifications allow users to stay up to date with their movement goals, and motivate them to continue working towards the specific goal [23].

HeartMate similar to Nike+ Running as it is only dedicated to tracking the activity, running. The usual workout statistics are provided similar to the other applications mentioned. The competitive aspect of HeartMate does create a social

Table 6.1: Comparisons Between HeartMate and Other Fitness Applications.

	Nike+ Move	Nike+ Running	Fitbit	HeartMate
User Accounts	YES	YES	YES	YES
Track Multiple Activities	YES	NO	YES	NO
Social Network	YES	YES	YES	YES
Challenge Based Goals	YES	YES	YES	NO
Real Time Competitions	NO	NO	NO	YES
Challenge Previous Workouts	NO	NO	NO	YES
Challenge Friends' Previous Workouts	NO	NO	NO	YES
Factors Fitness Level for Challenges Between Friends	NO	NO	NO	YES
Track Location	YES	YES	YES	YES

environment the other applications provide such as the ability to add friends and compete against them. The competition and motivation aspect is where HeartMate and the previously mentioned applications diverge. Nike+ Move, Nike+ Running, and Fitbit all have similar comparisons of performance measures where the user had to gain more NikeFuel, miles, or steps than their friends. HeartMate provides the ability to determine who is working out harder during a real time workout, through the use of a previous workout stored locally or from a friend's profile. In both cases, the competition is as if the user is beside the opponent as voice updates are provided during the workout to push the user to work their hardest. Zones calibrated to each user is utilized to determine who is working harder than the other. Scoring is not done through the use of steps or some form of NikeFuel, but is done through how long a user can stay in a higher zone than their opponent. Zones are relative to each person's fitness level, which none of the previous apps mentioned take into account. Table 6.1 concludes the comparisons between HeartMate and other fitness applications.

HeartMate is a prototype and is no way claiming to be better than any of the applications used for comparisons. HeartMate provides a different method for creating a social, motivational and competitive environment through the use of heart rates as

the performance measure. Where the performance of users is calibrated specifically to their fitness level.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Smartphones are becoming more useful every day with the addition of third party applications. They provide utility, and even provide the user ways to maintain a healthy lifestyle through the use of third party fitness applications. The technology that is found in smartphones is becoming more advanced in the way a user's health can be tracked. Accelerometers, gyroscopes, GPS, Bluetooth LE, and even dedicated chips to assist with the tracking of some of those sensors such as the M-Series Motion Coprocessors, are becoming widely available for health tracking. With the help of those sensors this thesis presented HeartMate, an iOS mobile running fitness application that utilizes a bluetooth connected heart rate monitor to provide a new competitive and motivational environment through a social component using heart rates as the performance measure. Specifically, HeartMate utilized a client-server architecture to allow users to create a profile and use that profile to connect with friends all over, as well as create competitions between friends using their heart rate data as a performance metric. Users can challenge theirs or their friends' previous workouts, or even compete real time through the use of the publish / subscribe model to stream heart rate data. HeartMate is different from other fitness apps because it allows comparison of heart rates relative to the user's fitness level through the use of the Karvonen method which factors in a user's resting heart rate (HR_{rest}) and maximal heart rate (HR_{max}), to calculate a percentage of the heart rate reserve ($\%HRR$),

which correlates to a user's volume of oxygen consumption reserve ($\%VO_2R$) to create intensity zones relative to the user. Doing so provides accurate comparison between competitors relative to their own fitness level, and not that of the user in which they are competing against. HeartMate uses the tracking of these zones to create a competition as well as provide updates to theirs and their opponent's current level of intensity throughout the workout for motivation. HeartMate adds to the different types of fitness apps already available, but in a new way.

7.2 Future Work

7.2.1 Different Cardio Tracking Types

Currently, the workout activity being tracked is running. The addition to other cardiorespiratory exercises such as cycling could be added as a workout and utilize the M-series Coprocessor chip as it supports the tracking of other exercises. Sensor data for different activities would have to be captured and abstracted through the Core Motion framework instead of the Health Kit framework, which HeartMate currently does to track a run.

7.2.2 User Interface Enhancements

Live Graphing of Heart Rates During Challenges

HeartMate currently provides a progress meter of the user and their opponent's intensity level, but it would be more interesting to see a live graph of the heart rate data as time progressed. The current libraries used for graphing do not support live data. A more extensive customizable graphing library could be implemented to add this feature.

Heart Rate Graph Intensities

The displaying of heart rate data with the zone information is currently done through the use of a popup view. Gradient thresholds applied to the line graph would make

it easily more readable to the user to see what zone they are in at a glance. Once again, a custom graphing library for iOS could be utilized.

7.2.3 Apple Watch Support

The Apple Watch has essentially all the sensors required to integrate HeartMate. The Apple Watch has Wifi, sensors to track motion, and a built in heart rate monitor. Originally, the plan was for HeartMate to be developed for both the Apple Watch and iPhone, but the hardware for the Apple Watch was limited for real time tracking of heart rates. The heart rate monitor data was inaccessible during a workout until the user woke the screen of the device, and then would be flooded with the heart rate data tracked. HeartMate uses a one-to-one measure of heart rates per second during the workout and processes the information as it is received. Zone updates, and motivation would not be possible in real time. Once the Apple Watch hardware improves, an Apple Watch version of HeartMate will be possible.

7.2.4 Android Support

Using the same backend, Parse, for an Android version of HeartMate will allow cross device competition. Only the client application would need to be developed as the backend already exists, and could manage the Android accounts as the Parse framework supports Android.

7.2.5 Integrate Other Health Sensors

HeartMate provides the foundation for accessing other sensors through Health Kit, and can easily be integrated to work with other Bluetooth LE devices that are designed to work with Health Kit. The game component can be stripped from HeartMate and provide a way for users to create health profiles, and monitor data provided through third party sensors. These health profiles can be created and stored locally or through the use of the Parse server.

Bibliography

[1] Apple, Inc. About the ios technologies. https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html, 2014. [Online; accessed 20-April-2016].

[2] Apple, Inc. About Swift. https://swift.org/about/, 2016. [Online; accessed 21-April-2016].

[3] Apple, Inc. Apple push notification service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html, 2016. [Online; accessed 20-April-2016].

[4] Apple, Inc. Av foundation programming guide. https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/00_Introduction.html, 2016. [Online; accessed 21-April-2016].

[5] Apple, Inc. Core data programming guide. https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreData/, 2016. [Online; accessed 21-April-2016].

[6] Apple, Inc. Core location framework reference. https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/, 2016. [Online; accessed 21-April-2016].

[7] Apple, Inc. The healthkit framework. https://developer.apple.com/library/ios/documentation/HealthKit/Reference/HealthKit_Framework/index.html, 2016. [Online; accessed 21-April-2016].

[8] Apple, Inc. The mapkit framework reference. https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference/, 2016. [Online; accessed 21-April-2016].

[9] Apple, Inc. Media player framework reference. https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MediaPlayer_Framework/index.html, 2016. [Online; accessed 21-April-2016].

[10] Apple, Inc. Uikit framework reference. https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/, 2016. [Online; accessed 21-April-2016].

[11] American Heart Association. All about heart rate (pulse). http://www.heart.org/HEARTORG/Conditions/More/MyHeartandStrokeNews/All-About-Heart-Rate-Pulse_UCM_438850_Article.jsp#.VxwC0pMrLa5, 2016. [Online; accessed 21-April-2016].

[12] Owen Barder. *Running for fitness*. A. & C. Black, 2002.

[13] Jakub Berlinski, Cameron Rowe, Marlon D Chavez, Nathan M Jordan, Devyani Tanna, Roger V Hoang, Sergiu M Dascalu, Laurence C Jayet Bray, and Frederick C Harris Jr. Neocortical builder: A web based front end for ncs. In *Proceedings of the 27th International Conference on Computer Applications in Industry and Engineering (CAINE-2014)*, 2014.

[14] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. IETF, 2014.

[15] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, 2009.

[16] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[17] BrianMac Sports Coach. Heart rate training zones. http://www.brianmac.co.uk/hrm1.htm. [Online; accessed 21-April-2016].

[18] CocoaPods. What is cocoapods. https://cocoapods.org/. [Online; accessed 20-April-2016].

[19] Joe Decuir. Bluetooth 4.0: low energy. *Cambridge, UK: Cambridge Silicon Radio SR plc*, 16, 2010.

[20] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification. using game-design elements in non-gaming contexts. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pages 2425–2428. ACM, 2011.

[21] Pierre Dulac. DPMeterView. https://github.com/dulaccc/DPMeterView, 2016. [Online; accessed 21-April-2016].

[22] Boris Emorine. BEMSimpleLineGraph. https://github.com/Boris-Em/BEMSimpleLineGraph, 2016. [Online; accessed 21-April-2016].

[23] Fitbit, Inc. Fitbit. https://itunes.apple.com/us/app/fitbit/id462638897?mt=8 and https://www.fitbit.com/app, 2016. [Online; accessed 18-April-2016].

[24] Kent German. A brief history of android phones. http://www.cnet.com/news/a-brief-history-of-android-phones/, 2016. [Online; accessed 21-April-2016].

[25] Arielle S Gillman and Angela D Bryan. Effects of performance versus game-based mobile applications on response to exercise. *Annals of Behavioral Medicine*, pages 1–6,157–162, 2015.

[26] Daniel Cohen Gindi. BEMSimpleLineGraph. `https://github.com/danielgindi/Charts`, 2016. [Online; accessed 21-April-2016].

[27] Liu Guo-Cheng and Yu Hong-Yang. Design and implementation of a bluetooth 4.0-based heart rate monitor system on ios platform. In *Communications, Circuits and Systems (ICCCAS), 2013 International Conference on*, volume 2, pages 112–115. IEEE, 2013.

[28] James L Hargrove. History of the calorie in nutrition. *The Journal of nutrition*, 136(12):2957–2961, 2006.

[29] N. Hutchings. Nike+ fuelband se review. `http://www.coachmag.co.uk/fitness-technology/3469/nike-fuelband-se-review`, November 2014. [Online; accessed 18-April-2016].

[30] Andrew Imm. The new parse developer experience. `http://blog.parse.com/announcements/the-new-parse-developer-experience/`, 2016. [Online; accessed 21-April-2016].

[31] LR Keytel, JH Goedecke, TD Noakes, H Hiiloskorpi, Raija Laukkanen, L Van Der Merwe, and EV Lambert. Prediction of energy expenditure from heart rate monitoring during submaximal exercise. *Journal of sports sciences*, 23(3):289–297, 2005.

[32] Matthias Kranz, Andreas Möller, Nils Hammerla, Stefan Diewald, Thomas Plötz, Patrick Olivier, and Luis Roalter. The mobile fitness coach: Towards individualized skill assessment using personalized mobile devices. *Pervasive and Mobile Computing*, 9(2):203–215, 2013.

[33] Cameron Lister, Joshua H West, Ben Cannon, Tyler Sax, and David Brodegard. Just a fad? gamification in health and fitness apps. *JMIR serious games*, 2(2), 2014.

[34] William D McArdle, Frank I Katch, and Victor L Katch. *Exercise physiology: nutrition, energy, and human performance*. Lippincott Williams & Wilkins, 2010.

[35] Network Computing. Client/Server Fundamentals. `http://digital.networkcomputing.com/netdesign/1005part1a.html`, February 1999. [Online; accessed 20-April-2016].

[36] Nike, Inc. Explore the power of nikefuel. `http://www.nike.com/us/en_us/c/nikeplus/nikefuel`, 2016. [Online; accessed 18-April-2016].

[37] Nike, Inc. Nike+ move. `https://itunes.apple.com/us/app/nike+-move/id712498492?mt=8` and `https://secure-nikeplus.nike.com/plus/support#answers/detail/article/nikeplus-move`, 2016. [Online; accessed 18-April-2016].

[38] Nike, Inc. Nike+ running. https://itunes.apple.com/us/app/nike+-running/id387771637?mt=8 and https://support-en-us.nikeplus.com/app/answers/list/p/4220,4241, 2016. [Online; accessed 18-April-2016].

[39] Brett Ohland and Jayant Varma. *Xcode 7 Essentials*. Packt Publishing, second edition, 2016.

[40] Parse. ios guide. https://parse.com/docs/ios/guide, 2016. [Online; accessed 21-April-2016].

[41] Scott Powers and Edward Howley. *Exercise Physiology: Theory and Application to Fitness and Performance*. McGraw-Hil Publishing Company, seventh edition, 2008.

[42] PubNub, Inc. The pubnub data stream network. https://www.pubnub.com/products/global-data-stream-network/, 2016. [Online; accessed 21-April-2016].

[43] T Scott Saponas, Jonathan Lester, Carl Hartung, and Tadayoshi Kohno. Devices that tell on you: The nike+ ipod sport kit. *Dept. of Computer Science and Engineering, University of Washington, Tech. Rep*, 2006. [Available at https://www.cs.washington.edu/research/systems/nikeipod/tracker-paper.pdf, accessed 18-April-2016].

[44] DAVID P Swain and BARRY A Franklin. Vo˜ 2 reserve and the minimal intensity for improving cardiorespiratory fitness. *Medicine and Science in Sports and Exercise*, 34(1):152–157, 2002.

[45] David P Swain and Brian C Leutholtz. Heart rate reserve is equivalent to% vo2 reserve, not to% vo2max. *Medicine and science in sports and exercise*, 29(3):410–414, 1997.

[46] Hirofumi Tanaka, Kevin D Monahan, and Douglas R Seals. Age-predicted maximal heart rate revisited. *Journal of the American College of Cardiology*, 37(1):153–156, 2001.

[47] Emmanuel Munguia Tapia, Stephen S Intille, William Haskell, Kent Larson, Julie Wright, Abby King, and Robert Friedman. Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor. In *Wearable Computers, 2007 11th IEEE International Symposium on*, pages 37–40. IEEE, 2007.

[48] Time. 8 years of the iphone: An interactive timeline. http://time.com/2934526/apple-iphone-timeline/, 2016. [Online; accessed 21-April-2016].

[49] James Turner. *Encyclopedia of Behavioral Medicine*, chapter Maximal Exercise Heart Rate, pages 1200–1200. Springer New York, New York, NY, 2013.

[50] Ted Vickey, John Breslin, and Antonio Williams. Fitness–there's an app for that: Review of mobile fitness apps. *International Journal of Sport & Society*, 3(4), 2012.

[51] Roy Want, Bill Schilit, and Dominik Laskowski. Bluetooth LE Finds its Niche. *IEEE Pervasive Computing*, (4):12–16, 2013.

[52] Mitchell H Whaley, Leonard A Kaminsky, Gregory B Dwyer, Leroy H Getchell, and JAMES A Norton. Predictors of over-and underachievement of age-predicted maximal heart rate. *Medicine and science in sports and exercise*, 24(10):1173–1179, 1992.

[53] Yue Wu, Atreyi Kankanhalli, and Ke-wei Huang. Gamification in fitness apps: How do leaderboards influence exercise? ICIS, 2015.

[54] John Lluch Zorrilla. SWRevealViewController. `https://github.com/John-Lluch/SWRevealViewController`, 2016. [Online; accessed 21-April-2016].