TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Sergii Spivakov 157382

IMPLEMENTING LIFE-TRACKER MOBILE APPLICATION

Master's thesis

Supervisor: Vladimir Viies

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Sergii Spivakov

Abstract

Most of the users are interested in measuring their health or some other activity. For example, many years ago, if you need to count your foot steps you would start counting each step and at the end you would probably miss something or would be distracted with something or somebody and your counting shall start again from the beginning. Nowadays, it is hard to imagine our life without smartphone or some activity tracker device. You can wear a special wristband or watches, pair it with your smartphone or even without pairing and all that you need to get your steps amount is just look onto the screen of your phone or other device. Therefore, it is important to say that mobile devices and technologies are our future and we have to follow and use all of the benefits that it brings to us. If you need to count your footsteps, use your mobile phone, if you need to count your heart rate use your wristband or smart-watches, if you need to count calories or track your health during the exercise again you can use a special workout manager device, watches or some other technology that gives you an opportunity to do this.

The main purpose of this thesis is to investigate various life-tracking applications and show how to implement your own self-tracking app. This paper sets out to explore the particular types of life-tracking applications, software and hardware that can be used to measure user's data. The research topic is novel and therefore and exploratory approach is taken for the analysis. For the conceptual framework, self-quantified literature is reviewed to depict the common life-tracker methods and describe the principles of tracking application implementation. The empirical analysis focuses on the already implemented and known self-tracking applications. Additionally, there was implemented a prototype application to understand and present how to build up a life-tracker from the scratch.

Application uses modern frameworks and methods in iOS development that includes CoreData usage, WatchOS, Cocoa Pods dependency manager, Dependency Injection, SOLID principles of OOD and so on.

As a conclusion, you can use information and source code provided to create your own applications. Even though that prototype project is oriented on iOS development the methods and technologies can be used in different platforms such as: Android, Windows.

Keywords: life-tracking, Quantified Self, software development, mobile devices, wristband, sensors, tracking cues, triggers, visualization, user interface(UI), iOS, WatchOS, accelerometer, gyroscope, CoreMotion, HealthKit.

This thesis is written in English and is 89 pages long, including 5 chapters, 16 figures and 1 table.

Annotatsioon

Mobiilse elutegevuse monitooringu rakenduse loomine

Enamus kasutajaid soovivad mõõta oma tervist või mõnda muud liikumisega seotud tegevust. Näiteks mõni aasta tagasi tuli oma sammude loendamiseks lugeda igat sammu eraldi ja tõenäoliselt kadus lugemise ajal tähelepanu kellegi või millegi tõttu ning sammude loendamist tuli alustada uuesti. Tänapäeval on raske kujutada ette elu ilma nutitelefonita või seadmeta, mis jälgib kasutaja aktiivsust. Võimalik on kanda randmele kinnitatavat aktiivsusmonitori või nutikella, ühildada seade soovi korral oma nutitelefoniga ja sammude loendamiseks tuleb vaadata ainult telefoni või muu nutiseadme ekraanile. Seetõttu on oluline märkida, et mobiiliseadmed ja tehnoloogia on meie tulevik ning meil tuleb kaasa minna eelistega, mida see meile pakub. Kui kasutaja soovib loendada samme, on seda võimalik teha mobiiltelefoniga. Südamerütmi on võimalik jälgida aktiivsusmonitoriga või nutikellaga. Ka kulutatud kaloreid või üldiselt kogu trenni kulgemist on võimalik jälgida spetsiaalse seadmega - nutikell või mõni muu tehnoloogia, mis annab võimaluse seda kõike teha.

Käesoleva lõputöö peamine eesmärk on uurida erinevaid *life-tracking* rakendusi ja näidata, kuidas ise sellist rakendust implementeerida. Töös uuritakse teatud tüüpi *life-tracking* rakendusi, tarkvara ja riistvara, mis võimaldavad mõõta kasutaja andmeid. Uurimisteema on uudne ja seega on ka analüüsimisel kasutatud uurimuslikku lähenemist. Töös on antud ülevaade *self-quantified* kirjandusest, et saada ettekujutus *life-trackeri* tavalisematest meetoditest ja kirjeldada jälgimisrakenduse implementeerimise põhimõtteid. Empiiriline analüüs keskendub juba olemasolevatele ja tuntud *self-tracking* rakendustele. Lisaks on töös implementeeritud ka rakenduse prototüüp, et aru saada ja näidata, kuidas luua *life-tracker* nullist.

Rakenduses kasutatakse kaaseaegseid iOS arenduse raamistikke ja meetodeid - CoreData, WatchOS, Cocoa Pods dependency manager, Dependency Injection, OOD SOLID põhimõtteid jne.

Töös esitatud lähtekoodi ja informatsiooni võib kasutada oma rakenduste loomiseks. Kuigi prototüüp on mõeldud arendamiseks iOS platvormil, on meetodid ja tehnoloogia sobilikud arendamiseks erinevates platvormides – näiteks Android, Windows.

Võtmesõnad: *life-tracking, Quantified Self*, tarkvaraarendus, nutiseadmed, aktiivsusmonitor, sensorid, jälgimise põhimõtted, *triggers*, visualiseerimine, kasutajaliides, iOS, WatchOS, akselomeeter, güroskoop, CoreMotion, HealthKit.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 89 leheküljel, 5 peatükki, 16 joonist ja 1 tabelit.

List of figures

List of tables

List of abbreviations and terms

QS	Quantified Self
UI	User Interface
MVC	Model-View-Controller design pattern
App	Application
MVVM	Model-View-ViewModel design pattern
DNA	Deoxyribonucleic acid
EEG	Electroencephalography
ECG	Electrocardiography
TED	Technology, Entertainment, Design
HCI	Human–computer interaction
URL	Uniform Resource Locator
IDE	Integrated Development Environment
SQL	Structured Query Language
TV	Television
CD	Compact Disc
GPS	Global Positioning System
DIY	Do It Yourself
SMS	Short Message Service
PDF	Portable Document Format
OS	Operating System
API	Application Programming Interface
OOD	Object Oriented Design

Table of Contents

1. Introduction

Time – it is something that we wish to have more and wish to have a possibility to save it somehow. It is a commonsense and one of the biggest observations: there is a wide disconnect between our awareness of problems, our knowledge of answers, even our professed priorities in life, and the ways we actually live.

Scientologists have marked a disconnect, but it could be the modern foible for life-tracking which is making it clear to understand. Actually investigation what amount of time we spend on some useless activities can be a shock for us.

We understand and perceive that we must sleep and it would be better to sleep more. Too much work, too many problems, worries. We have to work, this is normal and there are, of course, many studies on how much time at work is actually wasted on talk, coffee, tea and/or cigarette breaks, dreaming in Facebook or other social network, if allowed and even more discussions of the productivity and profit supposedly wasted, but who consider that probably human beings need to daydream and might be more creative and motivated by it? We have to cook, procure and prepare food, to eat, to follow other bodily and psychological needs, whether to the restroom or into relaxation. People have to socialize and want to spend time with their partner, let alone be there for our children. It is normal that we understand our values and find the time for the things we are fascinated of.

But there's a huge problem: with such busy lives, we don't feel we have the time to explore and follow any passions, if we are even aware of any such thing. We don't even feel we have the time to talk with people, to maintain our living spaces well. Such a basic feature and structural element of life as cooking and eating together has fallen by the wayside, packaged and sold as something to be avoided, in favor of convenience "food" and time at a job or with daily life's busyness. At the same time, the average person clearly has time enough in a day for social networks, watching TV and so on.

Once looking at these things more closely, measuring them or even just being made more aware of them, realization strikes.

Maybe, we should be appreciating the decent performance of everyday parts of life more, but we typically don't even spend much time with those.

We say we want to be with our families, we want to take good care of ourselves, do more for health and fitness or gym, learn more and grow, but all too much of our time, our very existence, is all too often spent making money that's immediately spent itself, never quite turns out to be and beget enough, never really quite gives the freedom it was supposed to

and the pursuit of which is so exhausting that we feel able only of plopping down in front of the TV or excited only about shopping – and we feel that we've earned a right to do nothing more than relax like that, to boot.

You can say how important health and good sport are for you however much you want, but they'll not be nearly as good as they could be if you only buy a few organic or functional foods, spend a little time each week in a gym, but if you don't see any results you become lazy and you start to skip gym, eat unhealthy food and so on.

As we know our eyes make us the most confident because we believe in what we see. Of course, we can use paper to write down our results, performance, time we slept and so on, but if we are going to do it every day it becomes repetitive and routine, and the most important it also takes our time. To satisfy all our needs and safe time efficiently we have to consider digital life-tracking which uses modern technologies to track everything even without your perception that you are being tracked. Thus, we are saving our time for other parts of our daily life.

2. Life tracker application meaning

First of all, it is important to define what a life tracker application really means. From a technical perspective it is a binary file that is embedded into your mobile device or digital watches (Apple watch for an example), that is under Operating System control and uses all privileges of higher OS API level. But from the user perspective it is much bigger than a file, it is their life storage, where they can track their budget, schedule time, trainings, calculate amount of time they spent for some activity, get their heart rate, create a diet and the most important they can observe results, make conclusions and change some behavior and most of the work is performed automatically using predefined algorithms. At the end of this paper you will see a prototype application where you can observe said above.

More scientific definition of lifelogging, is a movement to incorporate technology into data acquisition on aspects of a person's daily life in terms of inputs (food consumed, quality of surrounding air), states (mood, arousal, blood oxygen levels), and performance, whether mental or physical. In short, life-tracking is self-knowledge through self-tracking with technology.

Data collection through self-monitoring and self-sensing combines wearable sensors (e.g. EEG, ECG) and wearable computing. Among the specific biometrics one can track are insulin and cortisol levels, sequence DNA, and the microbial cells which inhabit one's body.

Other names for using self-tracking data to improve daily functioning are self-tracking, auto-analytics, body hacking, self-quantifying, self-surveillance, and personal informatics.

2.1. History of life-tracking

According to Riphagen et al., the history of the quant metric self-tracking using wearable computers began in the 1970s:

"The history of self-tracking using wearable sensors in combination with wearable computing and wireless communication already exists for many years, and also appeared, in the form of surveillance back in the 1970s " [1]

Quant metric self-sensing was proposed for the use of wearable computers to automatically sense and measure exercise and dietary intake in 2002:

"Sensors that measure biological signals, ... a personal data recorder that records ... Lifelong video capture together with blood-sugar levels, ... correlate blood-sugar levels with activities such as eating, by capturing a food record of intake." [2]

The “life-tracking”, “life-logging”, “fitness/health tracking”, “quantified self” or “self-tracking” are modern labels. They show the wider trend of the progressions for organization and meaning-making in human history. There has been a use of self-taken measurements and data collection and analysis that attempted the same goals that the quantified movement has [3]. Scientisation plays a vital role in legitimizing self-knowledge through self-tracking.

The word and term "quantified self" was proposed in San Francisco, CA, by Wired Magazine editors Gary Wolf [4] and Kevin Kelly [5] in 2007 as a collaboration of users and tool makers who share an interest in self-knowledge through self-tracking. In 2010, Wolf has been speaking about the movement at TED [6], and in May 2011, the first international conference was held in Mountain View, California [7]. There are meetings and conferences in America and Europe. Gary Wolf said "Almost everything we do generates data." Wolf says that companies target advertising or recommend use data from phones, tablets, computers, watches, wristbands other technology, and credit cards. However, using the data they make can give people new ways to deal with medical problems, help sleep patterns, improve diet and so on.

Scientologists like Michel Foucault are determined as being a part of the foundations in the ideas of the quantified movement. Foucault and other scientologists focus on the idea of “care of the self”, in which they emphasize the importance of life-tracking for personal improvement and development. Foucault explains that it involves looking inside oneself and emphasizes self-reflection, which is also associated with the quantified self-movement, where self-tracking participants can attend “show-and-tell” style conventions to share their experiences and skills with the technology.

Nowadays the global community has over a hundred groups in 40 countries around the world, with the largest groups in San Francisco, New York, London, and Boston having more than 1600 members each.

2.2. Life-tracker methodologies

Like any empirical study, the main method is the collection and analysis of data [8]. In many cases, data are collected automatically using wearable sensors, often worn on the

wrist (it can be a wristband or watches) [9]. In other cases, data may be logged manually or with some other technology.

The data are collected and analyzed using techniques such as linear regression to establish correlations among the variables under investigation. To understand a potentially high-dimensional data, visualization techniques can suggest hypotheses that may be tested more rigorously using formal methods. One example of a visualization method is to view the change in some variable – say height in centimeters – over time.

The idea of tracking is not new, but the technology is and each year we receive some updates. A lot of people would track what they would eat, drink or how much physical activity they got within a week. Technology has made it easier and simpler to gather and analyze personal data and you can do it with visual, "eye" experience. Since these technologies have become smaller and cheaper to be put in smart phones or tablets, it is easier to take the quantitative methods used in science and business and apply them to the personal sphere.

Stories constitute a symbiotic relationship with bodies of large data. Therefore, quantified self-participants are encouraged to share their experiences of self-tracking at various conferences and meetings. [9]

2.3. Quantified Self

People consider a new kind of lifelogging approaches coming from the recently emerged Quantified Self. QS community promotes "self-knowledge through numbers." Lifelogging or life-tracking is the process of tracking personal data generated by our own activities. Personal data like food, sleep, exercise (fourth chapter shows this), mood, location, alertness, productivity, and even spiritual well-being may be collected and measured to be part of this log. Their lifelogging experiments and their tools have the intention of gaining knowledge about their own behaviors, habits and thoughts by collecting relevant information related to them. The starting point of the QS initiative is not a set of scientific theories, but it is based on empirical self-experimentation. Apart from Quantified Self, all these approaches and tools can also be found under a variety of names including personal informatics, living by numbers, self-surveillance, self-tracking and personal analytics.

The Quantified Self Community

We can go deeper about QS community. Quantified Self (QS) has emerged as a community pursuing different lifelogging methods and approaches. QS is a collaboration of users and developers who share an interest in self-knowledge through self-tracking. They are a diverse group of "life hackers, data analysts, computer scientists, early adopters, health enthusiasts, productivity gurus, and patients" – this message can be found on the official website. The experiments that they perform and the tools they use have the intention of gaining knowledge about their own behaviors, habits and thoughts by collecting relevant information related to them. Tools used to track data about themselves range from pen and paper to user-developed tracking applications or commercially available tools.

This community of self-trackers was born with the eagerness to experimentally know more about human life. They were motivated by the potential experiments that they observed around: People experiencing some change in their lives, going on or off a diet, kicking an old habit, making a vow or a promise, going on vacation, switching from incandescent to fluorescent lighting, getting into a fight. All of these experiments are fake, not real experiments, because typically no data was collected and no hypotheses are formed. But with the abundance of self-tracking tools now on offer, everyday changes can become the material of careful study. Its starting point was the creation of the quantifiedself.com blog by Gary Wolf and Kevin Kelly in 2007. Their initiative was extended in 2008 with a regular Meetup to discuss and share their practices with personal data tracking.

One of the pioneering projects in lifelogging was **MyLifeBits**, which was inspired by Vannevar Bush's article "As We May Think". In this project, Gordon Bell captured a lifetime's worth of articles, books, cards, CDs, letters, memos, papers, photos, pictures, presentations, home movies, videotaped lectures, and voice recordings and stored them digitally. Afterwards, since the QS community was founded, people have seen a wide variety of approaches where people track; e.g., more than 60 different categories of information about the own health, the power usage of a thatched cottage, Vitamin D consumption or sleep analysis.

Although the main purpose of the QS community is to know how they are affected by diverse factors, these self-trackers not only use the captured data for themselves, but they make their data available and report about the insights that they gain.

2.3.1. Blogs, Meetups and QS Conferences

Since its creation, the blog entries posted in the QS community have served as the main platform for the sharing of self-tracking practices, applications, experiences and other related materials. Both videos from their meetups as well as traditional blog articles are used as content for these entries. As of November 10, 2014, over 300 video posts in their QS post and more than 700 "Show&Tell" videos in the QS online video blog have been posted. Participants also share their practices, experiences, mistakes and lessons learned in regular QS meetups and annual conferences. In their QS meetups they promote a specific type of format called "Show & Tell", conceived to explain firsthand experiences with self-tracking and help with the narrative. This format organizes the talk by answering the Three Prime Questions [47]:

1. What did you do?
2. How did you do it?
3. What did you learn?

By answering these three questions, speakers talk about their basic approaches including problems, motivations and goals, the tools and methods that they used, and the insights and outcomes gained from the data and tracking process. Recently, the videos originated from these meetups and conferences have been of interest to HCI research and were used to analyze what motivates them to keep tracking data, what tools they use, what insights they gain, and what challenges they face [30].

2.3.2. The Quantified Self Approaches

Apart from all these experiments, a lot of tools are already available, which facilitate the tracking of different aspects of our lives. Hundreds of applications and tools were found during the survey conducted. Therefore, the following review contains only the most important and significant approaches to current thesis work.

Projects

MylifeBits is a project of Microsoft Research based on a lifelong storage of everything you can imagine. It is the fulfillment of Vannevar Bushs 1945 Memex vision including **FTS(**full-text search), text annotations, audio annotations, and hyperlinks. There are two parts of MyLifeBits: the first one is an experiment in lifetime storage, and the second one is a software research effort [48]. In the experiment, Bell has collected a lifetime's worth

of articles, books, cards, CDs, letters, memos, papers, photos, pictures, presentations, home movies, videotaped lectures, and voice recordings and stored them digitally. According to Gemmell [49], Bell is now paperless, and is beginning to capture phone calls, IM transcripts, television, and radio. The implemented MyLifeBits software holds SQL servers to support hyperlinks, annotations, reports, saved queries, pivoting, clustering, and fast search [49]. SenseCam [50] is a wearable digital camera that takes photographs passively. SenseCam contains different electronic sensors (light sensors, an infrared detector, a temperature sensor and accelerometers) whose measurements are used to automatically trigger photographs to be taken. Originally, the main goal of SenseCam was to provide an aid for people with memory loss. However, it has been used in a variety of settings. For instance, it has been used to automatically generate images from landmarks, to capture a person's environment in order to analyze the amount of exercise people take, to enable teachers to reflect on logs about their days, or to study how office workers work simultaneously on different tasks.

The SenseWear Armband developed by Bodymedia was motivated by the lack of an easy-to-use, reliable and cost efficient way to accurately assess metabolic physical activity and energy expenditure by consumers, clinicians and researchers. It allows users to track energy expenditure, physical activity durations and levels, and lifestyle information. The SenseWear Armband has been validated in several in the field studies in the field and in numerous clinical and scientific studies and patient programs. Another example of projects associated with the QS community is Mappiness.

This research project aims at gaining insights about how people's happiness is affected by their local environment; e.g., pollution, noise, location, or people around. Users enter how they feel by rating several moods (happy, relaxed, awake, etc.) within a certain scale (not at all–extremely). Users receive feedback about their mood and the factors that affect it. Aggregated values of users from the UK and London are also visualized in their online platform.

Tools

Below, there are some of the tools found in the Quantified Self community. This is not an exhaustive list, but it gives the reader a good perception of the kind of tools that can be helpful dealing with.

Web-based and Desktop Applications

- daytum: a platform consisting of a web-based application and a complementary iPhone App to collect, categorize and communicate users' personal and everyday data in form of items count; e.g., number of bought bananas or something else.
- mycrocosm: web service that uses the visualization of statistics to share chunks of personal information and allows users to track several aspects of their daily lives ranging from hours of everyday sleep to what color of shoes a person wears.
- moodscope: web-based application to measure user's mood through a card game, track its evolution on a graph and share the scores with friends, hoping to receive support from them.
- Dagaz: is an application to train meditation level through a game experience. The application makes use of mandala shapes, generated by the user's mind. Mandala shapes are symmetrical shapes used in many cultures and almost every religion.
- Microsoft HealthVault: online personal health-record service that is compatible with a growing number of home health monitors. Data from these devices can be uploaded directly into a patient's HealthVault record, where users can then create a handy graph of their blood pressure, weight, blood sugar, or other data, and share it with their doctors or family.
- Yawnlog: a website that lets users track how much sleep they are getting, note how good the sleep was, record their dreams and compare all of that information with their friends.
- Time Sink: activity tracking application that logs user activity on a Mac laptop or computer by keeping track of windows and programs used.

Physiological or Environmental Sensors

- Nike+: tracking of running activity. It includes measurement of physical constants (calories, heart beats), time, distances, goals to achieve and challenges to fulfill with other users.
- Fitbit: sensor that tracks a person's movement to produce a record of steps taken, calories burned, and sleep quality. Data are uploaded to the Web so that users can monitor their activity and compare it with that of their friends.
- Philips DirectLife: Activity Monitor tracks your body motion every time you move up, down, forwards, backwards and sideways. By measuring the acceleration of these movements, it calculates how much energy you used to make

them. With DirectLife, a program is offered to increase your activity levels and they provide you a personal coach who can help you stay motivated.

- SenseCam: wearable camera with a wide-angle lens that periodically takes photos without user intervention.
- SenseWear: is a physical activity monitor to track user movement. It includes an accelerometer to measure motion events and steps. Additionally, there is a special a galvanic skin response monitor to collect data about the electrical conductivity of the skin, temperature, and heat flux (the amount of heat dissipating from the body).
- MIO: watches and bands to track different physiological measures.

Mobile Applications

- Sleep Cycle: Sleep Cycle is an iPhone app that allows users to track their sleep. It creates a nightly record of users' sleep by analyzing the measurements from the device accelerometer.
- Runtastic: This company has set of applications and devices for life-tracking. Fitness tracker for an example.
- Health Application on iOS(iPhone, not iPad)
- Trixie Tracker: is an application conceived to keep tracking of a baby's life; e.g., naps taken or bottles fed. Its goal is to discover patterns and detect sleep schedules.
- oneLog: application for iPhone that allows tracking of any sort of information or activity.
- My Tracks: is an application that uses GPS position to track several parameters (path, speed, elevation) while the user walks, runs, or does any other outdoors activity. Data are available live, announcements notify users about their progress, and annotation features are provided.
- Memolane: this application collects and connects content from other sites such as Facebook or Twitter. Its goal is to create an easy way of exploring past social network content.

Table 1 shows a categorization of the reviewed applications and tools. The categorization is made according to their topic/field of application (columns) and the composition of the tool (rows).

	Private Life Loggers	Health and Sport Trackers	General Activity Trackers
App/Website	Memolane MyLifeBits Time Sink	Sleep Cycle Microsoft HealthVault	Tixie Traker YawnLog OneLog moodscope mappiness
Device		MIO BodyCare Telcare Vitality's GlowCaps	SenseCam
Device + App/Web		Nike+ Fitbit Philips DirectLife Cogito MedApps BodyMedia	Affectiva

Table 1. Application categorization

The reviewed apps are only a representative set of applications that are being developed or used in the QS community. On their QS blog a list of more than 500 available tools and apps can be found. It must also be taken into consideration that the survey conducted as basis for the model was completed in 2012. Since then, new tools and applications have proliferated, especially in the context of health and sport tracking. These new tools are not included in the presented survey in order to be consistent with the background that served as basis for the created model.

A major application of quantified self has been in health and wellness improvement [10]. There exists lots of devices and services which help with tracking physical activity, can make a sleep analysis and other things that we have in personal well-being. Corporate wellness programs, for example, will often encourage some form of tracking. Genetic testing and other services have also become popular.

Life-tracking is also being used to improve personal or professional productivity [11], with devices, applications and services being used to help people keep track of what they do during the day, where they spend their time and perform some activities.

There is one application in the education field, with devices that can be weared being used in schools so that students can learn more about their own life activities and related math and science [12].

The Nike+ FuelBand is one of the devices that people use as life-tracking tools. Many start-up companies occupy the market right now. Most of them help track data for some type of health or fitness pattern, you can find thousands of apps in App Store or Google Market. However, there are monster companies such as Nike, Jawbone, FitBit that have occupied current markets and it is really hard to overcome them.

A recent movement in quantified self is gamification. There are a wide variety of self-tracking technologies that allow everyday activities to be turned into games by awarding points or monetary value to encourage people to compete with their friends. The success of connected sport is part of the gamification movement.

Most of the self-tracking applications are working and can share with each other (actually iOS provided special storage called HealthStorage where each self-tracking app can save some sample and share with other applicaitons) and other websites so people can share information with one another. Each device or technology may connect with other devices, applications to achieve bigger purpose and goals (Chapter 4 show a clear example were iPhone is connected with Watches and what can be achieved when they working together in pair). For example, one may figure out that migraines were more likely to have painful side effects when using a particular migraine drug. Or other can measure your mood when you are performing some exercise.

The quantified self is also demonstrating to be a major component of “big data science", due to the amount of data that users are collecting on a daily basis. Although these data set streams are not conventional big data, they become interesting sites for data analysis projects, that could potentially be used in medical-related fields to predict health patterns or aide in genomic studies. Examples of studies that have been done using QS data include projects such as the DIYgenomics studies, the Harvard’s Personal Genome Project, and the American Gut microbiome project.

Quantified Baby

Quantified Baby or in short QB is a branch of the QS community movement with main goal to collect comprehensive data on a baby's activities (from playing some game to eating food), and using this data to make some predictions, advices about baby’s behavior, health and so on. There are a huge number of applications and devices to assist with data

collection or collect data automatically for some other analysis purpose. Reactions to "Quantified Baby" are mixed [13][14].

Doctors often say to parent to collect and record data about their babies in the first year. It can be a feeding time, sleep changes and time and other activities that might be critical for the baby's health. This is useful for both the parent (used to maintain a schedule and ensure they remain organized) and for the health professional (to make sure the baby is on target and occasionally to assist in diagnosis). The main power of QS is knowledge, and self-knowledge helps us to improve ourselves. The aim for many is to use this tracking to ultimately become better parents. Some parents use sleep trackers because they worry about sudden infant death syndrome [15].

A number of apps exist that have been made for parents wanting to track their baby's daily activities. Usually, metrics that are being tracked are sleeping, feeding, changes in mood and other unpredictable behaviour. Mood, activity, medical appointments and milestones are also sometimes covered. There are applications that are specifically made for breastfeeding mothers, or other milk-related events.

Quantified baby, as in quantified self, is associated with a combination of wearable sensors and wearable computing.

Summary

In this chapter life-tracking meaning and Quantified Self community were considered. As a conclusion it can be said that life-tracking is everywhere and it is really helpful nowadays. We discovered that there is a huge community called Quantified Self. It has many blogs, meetups, conferences and developed applications. Actually, most of the information in this thesis got from their website and scientific papers. There are a lot of software and hardware for life-tracking. You can find applications for desktop, smartphone or even smart watches. Also, there is a special branch is QS community called Quantified Baby. Quantified Baby is concerned with collecting extensive data on a baby's daily activities, and using this data to make inferences about behaviour and health.

To sum up, life-tracking helps us to avoid many problems and have overview of our life and activities.

3. Life tracker methods

There are a lot of different possibilities and methods to track human activity and I would like to mention some of them:

- Tracking cues: capturing and keeping track of certain data
- Triggering: gathering data and analysis
- Recalling and revisiting experiences: recalling and revisiting past experiences through the enrichment and presentation of data in order to make sense of past experiences.

3.1. Tracking cues

Tracking means the observation of a person. Tracking strives to collect data about person's life in order to give some understanding and higher picture. We further characterize tracking by the means that are used, the object that is tracked, and the goal that is being strived for.

Tracking Means

There are 2 main ways to track something: the first one is tracking through often specialized software (with available hardware) and the second one is to develop your own hardware/software pair with sensors that directly track behavior.

Software Sensors

Software sensors are applications (desktop-based, web-based or mobile-based) that aid the user in capturing experiences. Software sensors are particularly important for experiences that cannot (currently) be directly measured (such as feelings and/or ideas) and are often much simpler, more flexible and cheaper to produce than hardware sensors. Software sensors are currently used in a broad variety of QS applications.

Daytum [16] is one example of a QS application that relies exclusively on software sensors (it means that only higher level of API is used). It consists of a web application for desktop computers and mobile devices as well as an iPhone app. The application enables users to track any number of arbitrary things about their lives, to categorize them and to view infographics based on this data. Another example is the more specialized

Trixie Tracker [17] that supports tracking data about your child. It can be a diaper changes, naps and bottles fed and so on.

Hardware Sensors

Hardware sensors are devices that automatically capture data that can be used to deduce experiences or collect contextual information. Common categories of sensors are: environmental sensors (e.g., light sensors, thermometers or microphone) and physiological sensors (accelerometers, heart rate sensors, sphygmomanometers, etc.).
One example is the application **Sleep Cycle** [18] that makes use of an iPhone's acceleration sensor to track sleep states in order to help people better understand their sleep. Another well-known application is Nike+ [19], an application built around an accelerometer that is attached to a shoe. The goal of the application is to measure and track the distance of a walk or run.

Tracked Aspects

The tracked aspects for a QS application can be very broad—as, for example, in "Total Capture" applications like the SenseCam [20] project, which attempts to use a camera to track all aspects of daily life. However, research so far has found little evidence of such systems being effective in serving as a digital memory or supporting reflection processes [21]. Situation specific tracking applications—as, for example, the above mentioned Trixie Tracker - concentrate on only a small class of experiences (in this case some experiences related to child-rearing).
The tracked aspects found in Quantified Self applications can be classified in the following four categories:

- Emotional aspects. It can be an anxiety, habits, stress, mood, etc.
- Private and work data. Data from work processes and our lives such as photos, the browser's history, digital documents, music, or use of a particular software etc.
- Physiological data. Physical indicators and biological signals that describe a person's state of health. The main approaches comprise the measurement of physical activity (for applications focusing on sport) and factors indicating health and sickness (e.g., glucose level).

- General activity. Data about a user's general activity, such as the number of cigarettes, cups of coffee, hours spent in a certain activity or number of times that something is done.

Purposes

Another important classification dimension is that of the purpose of a QS application, the goal which the user tries to achieve by using this application. This target directs and guides which measures are tracked. Within their framework, [22] Li have identified six different purposes or questions people have in mind when tracking data.

Firstly, these are to get to know their current status for determining if goals are met or if correction in behavior is needed. Secondly, the log of their data for determining trends and progress made. Thirdly, other goals worth to achieve and drive. Fourthly, discrepancies between their set goal(s) and current behavior either to correct or maintain it. On more purpose is that the context that may influence their current status. Lastly, long-term influencing factors in order to monitor trends. Seen within the reflective learning model, the purpose is the outcome that the Moodscope application that encourages users to track and share their mood. For the already mentioned Daytum application the purpose is a general self-improvement, as the user is largely free to determine what he or she wants to track.

3.2. Triggering

Within the reflective learning process, triggers are responsible for starting the actual reflection process. The role of triggers is to raise awareness and detect discrepancy. We differentiate between active and passive triggering.

Active Triggering

Active triggering consists of the tool sending a notification or catching the attention of the user explicitly. Active triggering is supported if an application performs data analysis to detect experiences that are suitable for initiating reflection. Such an experience or situation may be a mismatch between users' goals and their current achieved level, a comparison to a global threshold or other persons, or a deviation from personal patterns. An example for active triggering are alerts on **RescueTime** [23]: the user can specify

goals such as expending a maximum amount of time on ‘distracting’ web sites or being a minimum
number of hours doing productive work per day. The system alerts the user with a notification when the goal has been achieved or the certain amount of time has passed by.

Passive Triggering

It can be said that application supports merely a passive triggering if it does not identify experiences suitable for fostering reflection or it does not actively interact with the user. This kind of system mainly displays the collected data in a suitable way. It relies on the user to be triggered by something outside of the system or on the user regularly visiting the site and then detecting something that starts a reflection process. Daytum (described above) is one application that relies purely on passive triggering.

3.3. Recalling and Revisiting Experiences

Different aspects affect the recalling and revisiting of past experiences, when analyzing the benefits that QS approaches could offer. Enrichment and presentation of the data may facilitate the revisiting of the data to analyze past experiences and reflect on them, and therefore enhance the learning process of the user. Support of Quantified Self applications can exist along multiple dimensions: Contextualization, Data Fusion, Data Analysis, and Visualization.

Contextualizing

The data that one sensor or even app collected can be more certain and accurate with other sensor/device/app data. A great example is a prototype application, where two sensors are used to make calculations more accurate and reliable. This contextualization of the data with other sources of information may be performed by the same tool or result from the interaction between tools (e.g., two mobile applications or a sensor with a desktop application). An example for the contextualization for reflection is Garmin Connect [24]. With the right hardware, this application can track heart rate, bicycle speed, cadence, altitude and location and display this in the context of a map—thereby facilitating a better understanding of fitness data.

Adapting the context definition from Dey [25], we define context within this model as: "any information that can be used to characterize the situation of a tracked entity and that can aid the reflection process." For the types of context that can be of relevance, we have to go beyond usual classifications of context in computer science (e.g., location, identity, activity and time [26]) since these focus on the context of a concrete interaction with a context-aware application. For this model we need to consider as context everything that can aid the understanding of sequences of (data about) experiences and outcomes. Based on a review of existing Quantified Self applications several classes of context can be identified:

- **Social context**. Data can be augmented with information about the social context of the user. This can be a comparison to social network friends (VKontakte, Facebook) or a comparison to all users. This enables a possibility to compare own performance with the others. Sharing in a social context provides additional data to others in expectation to retrieve more data in exchange and ultimately see one's own experiences in relation to others' experiences. Additionally, it can support the active processing of difficulties; e.g., by talking about problems and finding peers with similar problems. Sharing is a way of contacting with others and retrieving additional feedback on data. An aggregation of data over multiple users may provide new perspectives on experiences and offer new abstraction levels.
- **Spatial Context.** The location in terms of country, street or even the room. As context information, this data can aid life-tracking by helping the user to understand the relation between a place and his behavior—such as understanding the effects of high pressure on his or her heart rate, the calming effect of visits to specific places or the identification of the places where most time is lost in traffic.
- **Historical context.** Historical data is an additional type of context data that can aid in the self-learning process. Comparing current values to historic ones allows seeing upward or downward trends or to identify deviations from a historic norm that may indicate a problem. Historic data may also help to identify the difference between periodic fluctuations (such as variations in weight or fitness according to the seasons) and other deviations from the norm that may indicate progress or a problem.
- **Item Metadata.** Any metadata available about the things a user interacts with - such as the information that a particular website's user is accessing is not work

related but rather distracting, or data that this biscuit contains too much sugar and other harmful adding.

- **Context From Other Datasets**. In addition, there are numerous datasets (e.g., train or school schedule) that can also be used in contextualizing.

Data Fusion: Objective, Self, Peer and Group Assessment

One important aid to the tracking process can be the fusion and comparison of objective (i.e., measured by sensors), self-reported data from the user, peer and group assessment (reported data from others about a user). There may be differences and discrepancies between these views that can foster tracking, can help to bridge the gap from subjective to objective experiences and in this way yield new insights and lead to learning.

Data Analysis: Aggregation, Averages

Different forms of data processing help to present the user useful measurements (e.g., amount of lunch breaks per day, number of cups of tea per day/week, etc.).

Many books suggest three types of aggregation: mathematically; e.g., into averages and correlations, graphically; e.g., in charts and plots, or formally; e.g., by relation networks or tag clouds. The selected type of aggregation depends on the kind and amount of available data as well as on the purpose of the aggregation. For instance, graphs and plots require quantitative data and can show trends or help to identify high level patterns. Aggregation in tag clouds may need large amounts of data to become valuable but can be applied to semi or unstructured data like texts. Further, it might be desirable to hide the source of the underlying data through aggregation and in this way create anonymity and privacy.

Visualization

Presentation and visualization that are attractive and intuitive for the users should be chosen because it helps to foster the analysis of the data.

3.4. Exemplary Applications

In order to illustrate the application of the QS model, 2 QS tools are shown below and how they are classified. These two exemplary applications have been chosen with the intention of covering several of the main presented points: having one hardware sensor

and one software sensor; dealing with data about physical activity and data related to emotions; and finally having a goal-oriented approach and an approach with an unclear general goal.

Philips DirectLife

Philips DirectLife [27] is an activity monitor that tracks the body motion of the user and converts it into calories burned. By measuring the acceleration of movement, it calculates how much energy the user used to make them. DirectLife is based on a motion sensor together with a website. In this personalized website, the user can upload the data from the sensor, review how active he or she has been and monitor his or her progress against the personal plan. In addition, an activity program is offered to increase activity levels and a personal coach is provided, with the role of keeping the user motivated.

- **Tracking.** DirectLife consists of an accelerometer. It is a sensor that can measure a motion with acceleration. Therefore, the tracking of the user activity is done through a physiological sensor. In this case, the measurement and tracking of the activity is based on the of health and sport related measures. The main motivations for users to use this system include staying active, being healthier or losing weight. With DirectLife, goals are set based on the current activity levels of the user. It is something similar to activity rings in iOS native Activity application. Some example goals that the user can explicitly choose are being healthy, fit, active or sporty; and the difference between them is the level of activity and caloric targets.
- **Triggering.** Although DirectLife performs analysis on the data, namely aggregation of data, average calculation, goal comparison and social ranking, we would identify it as a passive triggering tool. With this system, the data are shown to the user in order to take a decision by him or herself and the tool does not explicitly arise the attention of the user to certain discrepancies or particularities of the data. The function of triggering the user to change her behavior and being more active might be in this case attributed to the personal coach (which is actually a person and not the tool itself).
- **Recalling and Revisiting.** In DirectLife, the main factor used to contextualize the data is time; i.e., it allows the user to see the data according to a certain day, week or month. Users can see their history, their daily average activity, their weekly activity and their personal plan. Moreover, they are ranked among other

participants based on their age and gender and they can compare themselves and feel more motivated with this social physiological data provided by the sensor and therefore this tool supports the tracking comparison. The burned calories calculated according to the body motion of the user is an objective measure, as it is not the perception of the user, but the data provided by the sensor. DirectLife provides the user with different bar graphics which show the data in a daily, weekly, monthly and yearly basis. Users can also see their progress in their website by comparing their activity with the ongoing target.

Moodscope

Moodscope consists of a web-based application that allows users to track their mood through a card game, see their mood evolution on a graph and share their scores with friends. The purpose of this sharing is the support that the user can receive from his or her friends, when they are aware of the user's emotional state. According to the website of this tool, "measuring your own mood is a daily must-do, just like cleaning your teeth or washing your face."

- **Tracking.** Moodscope is based on a web application that supports the capturing and tracking of emotional state of a person, concretely a person's mood. The measurement of the mood is done through a card game based on a psychological mood questionnaire called PANAS [28]. User have to choose between 20 double-sided cards every day. They can do it once a day and it is suggested to do it best at the same time each day. Having chosen a card, the user receives a score, which is a percentage scale from 0 which means "sad" and 100 which means "happy". The reason to choose this card game is the fact that cards engage the more thoughtful and reflective side of the human brain. The general goal that any user of Moodscope may have is being happier and thereby feeling better, which is a rather undefined, unspecific and unclear goal.
- **Triggering.** Moodscope follows the particular phenomenon called "The Hawthorne Effect" with the theory that when we believe that we are being observed, our behavior can change for the better. Therefore, Moodscope sends an email with the user's daily score and a progress graph to the people that act as a user's "buddy." These buddies are chosen by the users themselves and may be friends, colleagues or relatives or any other user chosen person. This sharing of the data acts as a motivation for users and affects the progress of their emotions,

but it is not a trigger itself. This tool may then support a kind of passive triggering, when presenting to user the magnitude in his/her mood.

- **Recalling and Revisiting.** The mood data tracked in Moodscope can be contextualized through comments made in the progress graph. Apart from presenting the evolution of the user's mood in a line graph, this tool does not apply any other data analysis. As explained before, users can share their moods with the people that they choose as buddies and these receive a daily email with their mood status and progress. In this case, although data are shared, there is no comparison established with the mood of anyone else. Due to the fact that the capturing of data is user-driven and chosen by themselves with a card, we can say that Moodscope deals with subjective data of the user.

3.5. Related Work

Overall the proposed combination of self-learning and QS applications in this thesis materializes the vision of learning analytics for a particular model of learning and a specific class of support tools. In the following, we present a review of related theoretical and application-oriented approaches found in the literature.

3.5.1. The Quantified Self and Personal Informatics

Recent studies have explored the motivation behind the Quantified Self [29] conducted a survey to understand the underlying motivations of self-triggered health monitoring in the QS community focused on patient-driven approaches and the implications for healthcare information systems. They show that self-tracking is a voluntary activity driven by both intrinsic and extrinsic motivations [29] that requires time and effort to first collect the data, then review and reflect on the data subsequently. In [30], main challenges as well as pitfalls are identified and analyzed using the experiences of QS members. They report that their "ultimate goal is to reflect upon one's data, extract meaningful insights, and make positive changes, which are the hardest part of QS" [30]. Rooksby [31] investigates what "people are making of personal trackers for themselves" and report an interview study with current users. These studies provide valuable insights about personal tracker users, but we lack empirical evidence whether these insights also apply in a work setting.

Previous research in self-tracking has predominantly focused on improving physical activity, sustainable living, health, and well-being [32]. Additionally, applications typically target individuals rather than teams. Issues under investigation are often tracking effort [33] and support for behavior change [34]. While social features are common in QS tools (e.g. sharing activity levels in minutes or distance run), Rooksby [35] argue that these features are not necessarily put to use, but users rather share data to announce their achievements to friends or to compete with other users. Hence, sharing typically serves to increase user motivation via competition or peer recognition [36] rather than to support team and communication processes, as it is necessary in work settings. In the study of Li [37], the authors surveyed and interviewed people who track and reflect on personally relevant information. Based on this, they derived a five-stage model of personal informatics systems. The stages comprise preparation, collection, integration, reflection, and action. The model focuses especially on barriers in each stage. In comparison to our framework, Li [37] addresses how to design personal informatics systems from a technical perspective, whereas we focus on how to use such systems especially for reflective learning. This aspect is missing in their work. Reflection is only considered as short- and long-term reflection. Nevertheless, we may extend the model by Li [37] and provide a more detailed view with the QS characteristics of our framework; for instance, regarding the collection of information with our tracking means and tracking objects. Within their framework, Li [33] have identified six different purposes or questions people have in mind when tracking data (see also Section 3.2.3). These are to get to know (1) their current status for determining if goals are met or correction in behavior is needed, (2) the history of their data for determining trends and progress making, (3) (new) goals worth pursuing, (4) discrepancies between their set goal(s) and current behavior either to correct or maintain it, (5) the context that may influence their current status, and (6) long-term influencing factors in order to monitor trends. They further identified two different phases of reflection, maintenance and discovery, related to the questions and provide suggestions on how to support these with technology. However, these insights are disconnected from their model. Self-tracking has been also investigated from the perspective of human-computer interaction systems. However, these research works have focused on approaches related to well-being, sport, mental health, or multiple types of disease [38]. There are few related works that deal with structuring QS approaches towards the purpose of reflection, but this is mainly from an HCI design perspective.

Summary

In the third chapter many things were considered, from tracking methods to exemplary applications.

There are a lot of different methods, but generalized view is that we have the following three:

- Tracking cues
- Triggering (active and passive triggering)
- Recalling and revisiting experiences

You can observe how these methods used in two exemplary application provided above.

To sum up, you can use any method you like depending on the application you need to build.

4. Life tracker application implementation

This chapter shows how to implement a basic type of Life-Tracker application. Here you can see a Fitness tracker type which helps you to track different exercises such us "Biceps" and "Hammer-Biceps" [39].

4.1. Used technologies

To implement life-tracker application many different technologies can be used, but for this project I have used the following:

Hardware:

- iPhone 7 paired with Apple Watch first generation 42mm
- Gyroscope
- Accelerometer

Software:

- iOS 10 and WatchOS 3 operating systems
- xCode IDE
- Swift 3 programming language
- Core Motion Framework
- WatchConnectivity Framework
- CoreData Framework
- UIKit framework
- WatchKit framework
- HealthKit framework

OOD Patterns:

- Singleton
- Strategy
- MVC
- Dependency Injection in its Pure form unfortunately.

I have chosen iOS with WatchOS platform because it is one of the best platforms for the implementation of Life-Tracker applications. It is highly documented and there are lots of tutorials how to use its hardware sensors.

Mainly, to detect users motion data Apple Watches sensors have been used. They have been used through Apples provided framework called CoreMotion.

The Core Motion framework lets your application receive motion data from device hardware and process that data. The framework supports accessing both raw and processed accelerometer data using Block object interfaces. For devices with a built-in gyroscope, you can retrieve the raw gyro data as well as processed data reflecting the attitude and rotation rates of the device. You can use both the accelerometer and gyro-based data for games or other apps that use motion as input or as a way to enhance the overall user experience [40].

To enable data interaction between WatchOS and iOS a WatchConnectivity framework has been used. Before WatchOS 2 other method could be used called "Data Sharing using App Groups", but since the release of WatchOS 2 Apple totally changed Apple Watches workflow. Therefore, to exchange data it is mandatory to use WatchConnectivity framework.

The Watch Connectivity framework (*WatchConnectivity.framework*) provides a two-way communications conduit between an iOS app and a WatchKit extension on a paired Apple Watch. Apps use this framework to pass files and data back and forth. Most transfers happen in the background when the receiving app is inactive. When the app wakes up, it is notified of any data that arrived while it was inactive. Live communication is also possible when both apps are active [41].

To save data SqlLite database has been used with special framework wrapper called CoreData.

UIKit and WatchKit are two special frameworks that helps developers to provide user interface for iOS and WatchOS respectively.

Core Data is a framework that you use to manage the model layer objects in your application. It provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence.

Core Data typically decreases by 50 to 70 percent the amount of code you write to support the model layer. This is primarily due to the following built-in features that you do not have to implement, test, or optimize:

- Change tracking and built-in management of undo and redo beyond basic text editing.
- Maintenance of change propagation, including maintaining the consistency of relationships among objects.

- Lazy loading of objects, partially materialized futures (faulting), and copy-on-write data sharing to reduce overhead.
- Automatic validation of property values. Managed objects extend the standard key-value coding validation methods to ensure that individual values lie within acceptable ranges, so that combinations of values make sense.
- Schema migration tools that simplify schema changes and allow you to perform efficient in-place schema migration.
- Optional integration with the application's controller layer to support user interface synchronization.
- Grouping, filtering, and organizing data in memory and in the user interface.
- Automatic support for storing objects in external data repositories.
- Sophisticated query compilation. Instead of writing SQL, you can create complex queries by associating an NSPredicate object with a fetch request.
- Version tracking and optimistic locking to support automatic multi-writer conflict resolution.
- Effective integration with the macOS and iOS tool chains [42].

The UIKit framework (*UIKit.framework*) provides the crucial infrastructure needed to construct and manage iOS and tvOS apps. This framework provides the window and view architecture needed to manage an app's user interface, the event handling infrastructure needed to respond to user input, and the app model needed to drive the main run loop and interact with the system.

Additional UIKit Features

In addition to the core app behaviors, UIKit provides support for the following features:

- A view controller model to encapsulate the contents of your user interface
- Handling touch- and motion-based events
- A document model that includes iCloud integration
- Graphics and windowing, including support for external displays
- Managing the app's foreground and background execution
- Printing
- Customizing the appearance of standard UIKit controls
- Animating user-interface content
- Integration with other apps on the system through URL schemes and framework interfaces
- Working with various accessibility settings and preferences

- PDF creation
- The user's photo library

In iOS, UIKit also supports the following features, some of which are device specific:

- Cut, copy, and paste actions
- The Apple Push Notification service
- Local notification scheduling and delivery
- Using custom input views that behave like the system keyboard
- Creating custom text views that interact with the system keyboard
- Sharing content through email, Twitter, Facebook, and other services
- The built-in camera (where present)
- Device name and model information
- Battery state information
- Proximity sensor information
- Remote control information from attached headsets [43]

HealthKit is a really huge and helpful framework, but here it is used only for the background mode support.

4.2. Project structure and implementation

First of all, it is important to understand a project architecture, how everything is connected and working together. Figure 1 represents project architecture for iOS projects, but you can follow the same scenario for any other mobile platform.

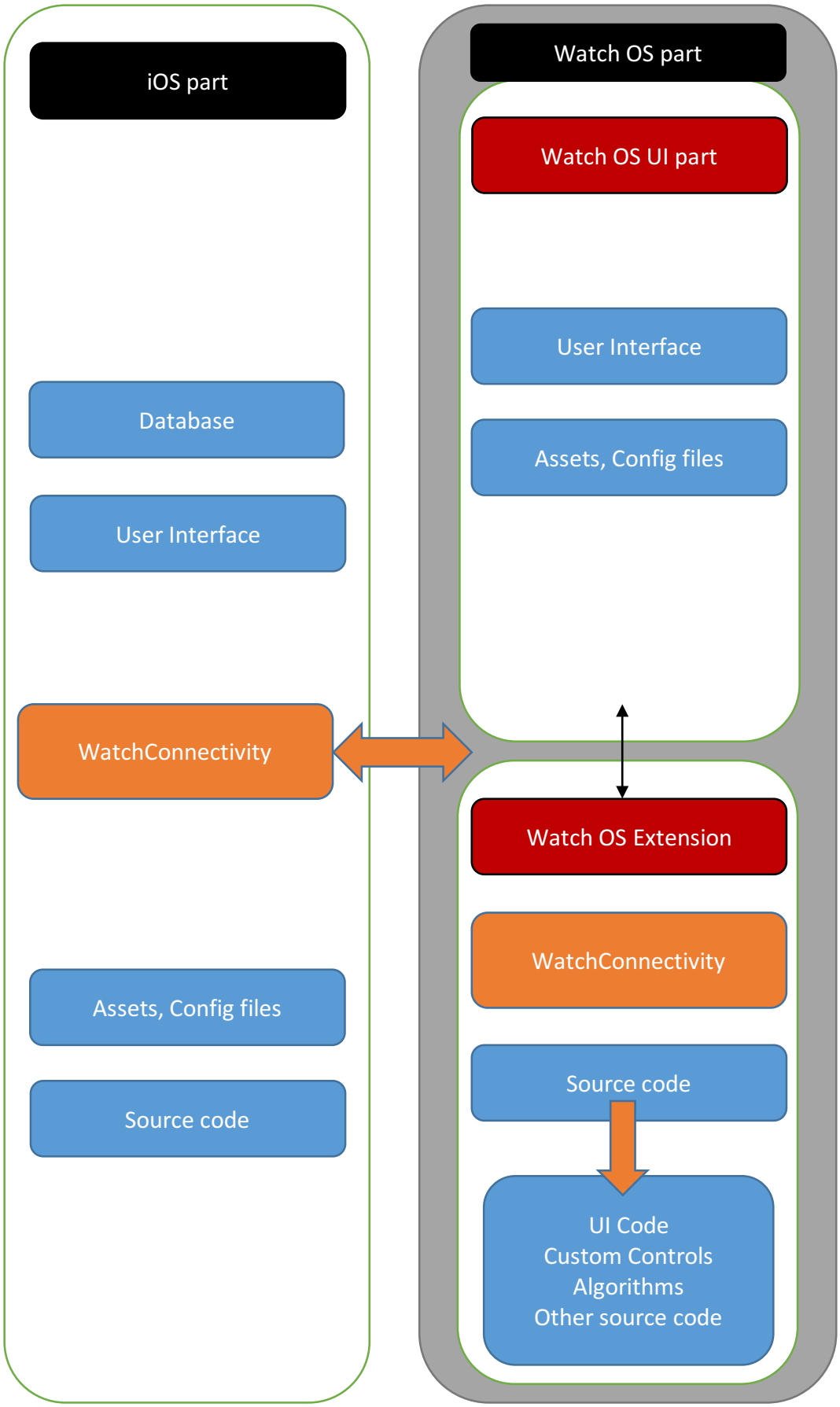

Figure 1. Application Architecture

As it can be seen from the Figure 1 we have an application that is separated into two parts – iOS and Watch OS. These are two separate operating systems, they have its own hardware, software and source code. Communication between them is done using WatchConnectivity Framework provided by Apple.

Project is written using a Model-View-Controller design pattern. It is quite easy and useful pattern in iOS development. As a substitution, MVVM can be used, but for the prototype application MVC is enough.

The basic structure of MVC pattern is the following:

- Model
- View
- Controller

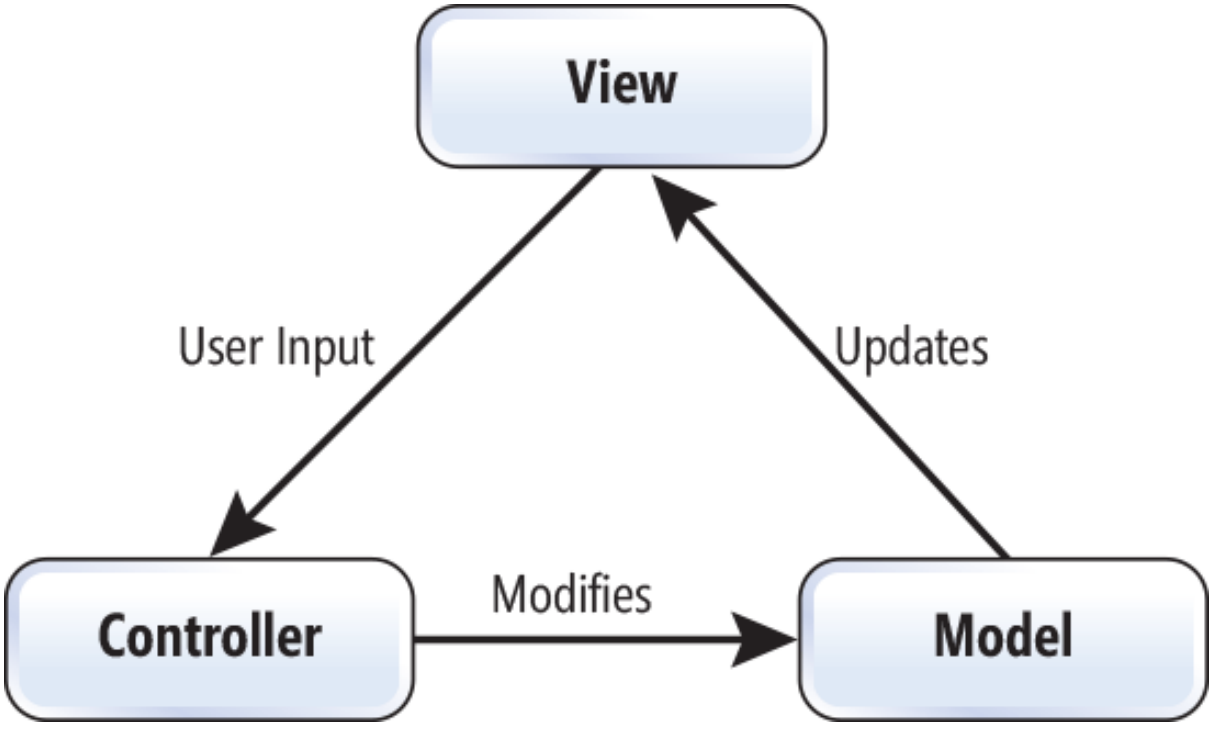

Figure 2. MVC Pattern

As it can be seen from the Figure 2 the structure is really easy and powerful. We have a controller that works like adapter between view and model. Additionally, view is connected with model using controller and vice versa.

Life tracker MVC pattern can be seen from the Figure 3.

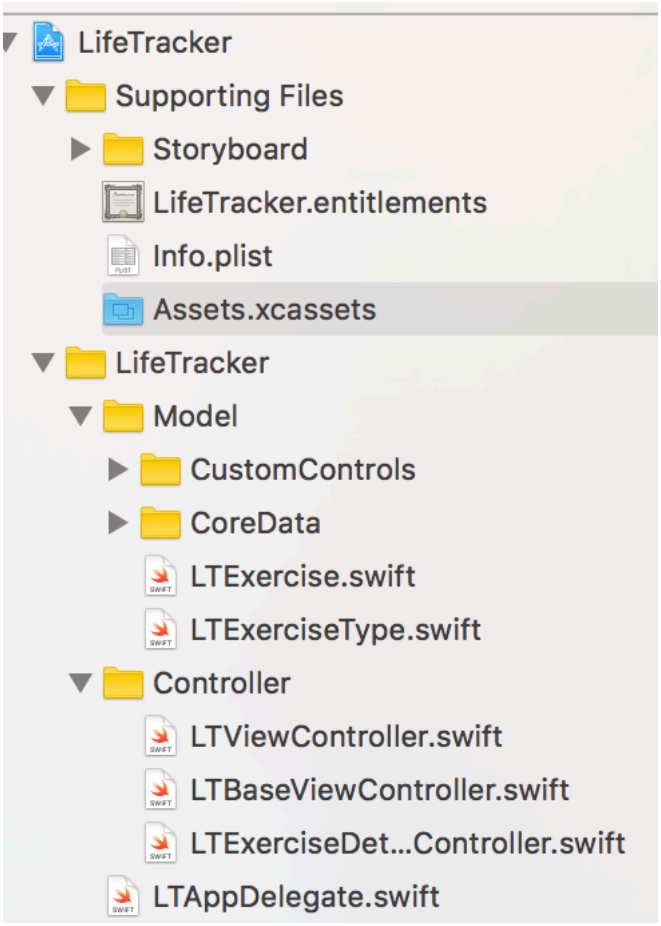

Figure 3. iOS MVC Structure

Generally, each exercise represents its own Class and calculation algorithm respectively. That is why a "Strategy" design pattern is used in this project to supply a specific calculation algorithm for each chosen exercise.

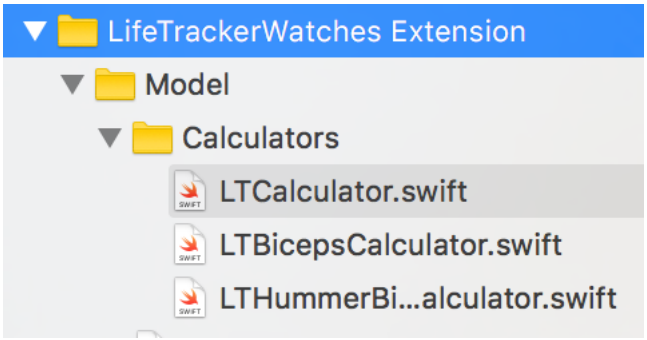

Figure 4. Strategy pattern

From the Figure 4 it can be seen that Strategy pattern is used. There is a basic protocol called LTCalculator(LT stands for Life Tracker), then there are two

classes(LTBicepsCalculator and LTHammerBicepsCalculator) that conforms to this protocol. This structure enables a light-weight algorithm supply.

LTBicepsCalculator hides detection algorithm for biceps exercise movement.
Figure 5 shows a basic biceps movement.

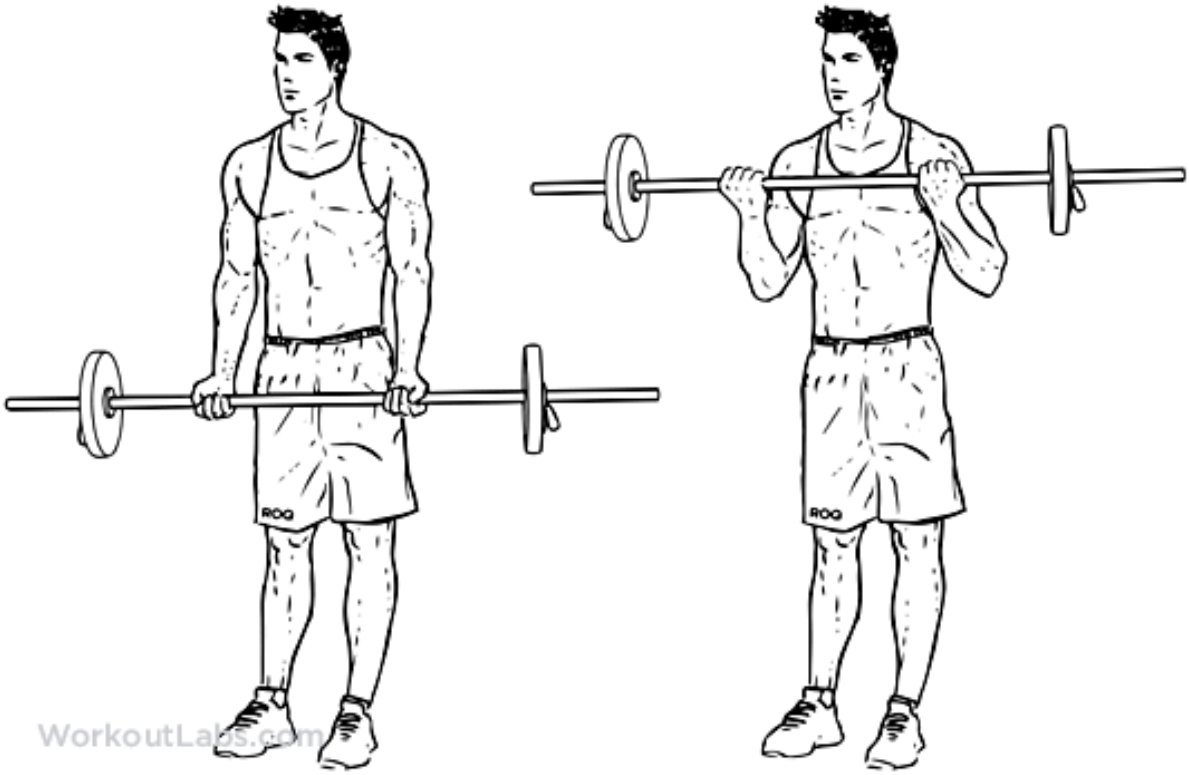

Figure 5. Biceps movement(http://workoutlabs.com)

To detect when user lifts bar or barbell we need to understand rotation magnitude.
Using empirical approach and Figure 6 provided from Apples documentation it is easy to understand what axis we need to follow to detect rotation change. To get data CoreMotion framework can helps us. Figure represents iPhone rotation axes, but the same applies to Apple Watches. Actually, it is quite confusing to determine right axes, but as I said empirical approach it a life vest for us.

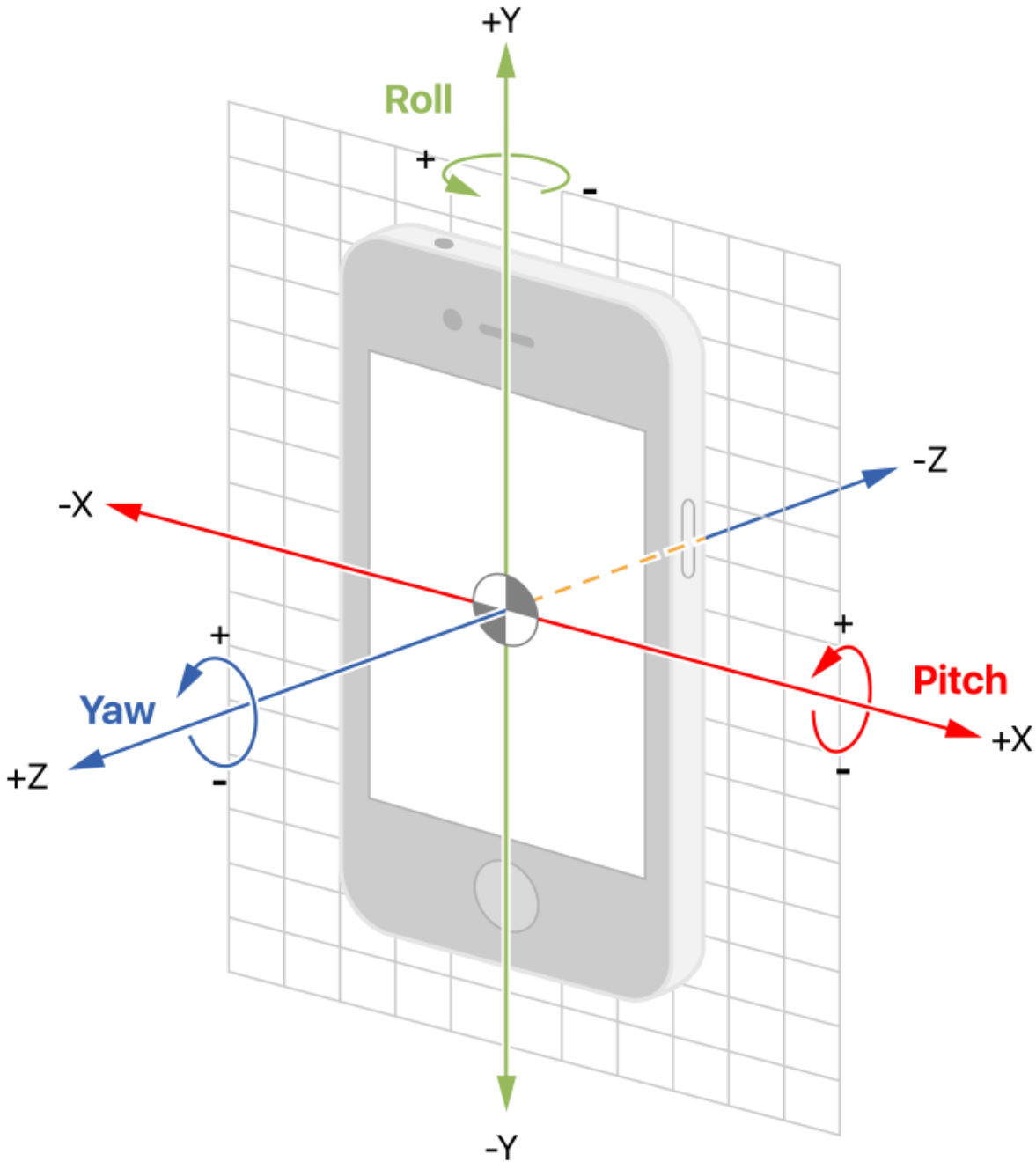

Figure 6. Pitch, roll, and yaw axes (https://developer.apple.com/library/content/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/Art/attitude_rotation_2x.png).

As you can see from the Figure 6 to detect a biceps movement it is enough to know Z and X axes. Because we need to determine actually simple biceps exercise, not hammer or other type, we need Z axis to determine wrist rotation. After that, we can use X axe to determine an angle in elbow fold. Of course there are few constraints that we need to have:

- User has to wear Watches and it has to be tight to his wrist.

- User has to understand that he is doing biceps, not other exercises. Because it is clear to understand that you can perform such rotation not only with biceps exercise. Therefore, this can be called as a leak and disadvantage in this algorithm, but unfortunately neither phone nor watches cannot detect their position change. On the other hand, it can be called as an advantage, because this algorithm can be reused for other exercises.

Additionally, to add some realism to this algorithm, we cannot determine everything only with just angle change. To understand that person actually did exercise correctly, he or she has to have some delay and that is why in algorithm we have to user some kind of buffer to collect gyro samples and once they reach some predefined point we can say that user performed it correctly.

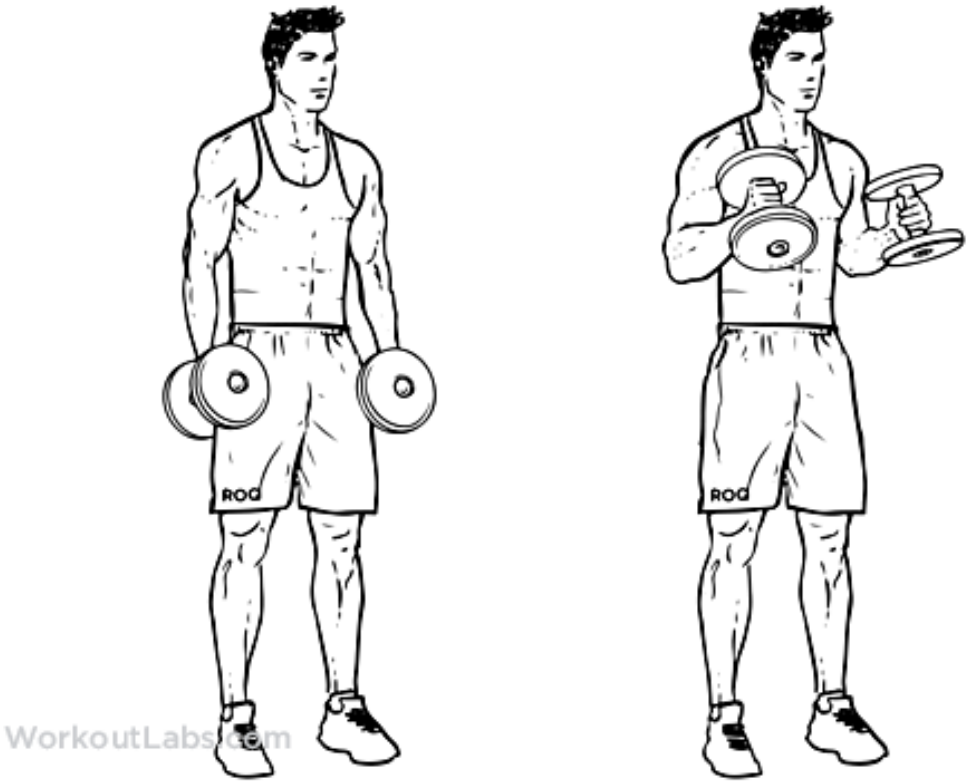

Figure 7. Hammer biceps exercise(http://workoutlabs.com)

Figure 7 represents Hammer biceps exercise. It can be seen that it is the same biceps only with wrist rotation change.

That is why a Hammer biceps has exactly the same algorithm with only one change with Z axes threshold. To be more certain, for Biceps Z axes minimal value is 10.0 degrees,

maximum value is 50.0 degrees and for Hammer Biceps type minimal value is -10.0 degrees, maximum value is 15.0 degrees. Of course, it cannot be measured with simple math, because these sensors have some faults and errors and we must take them to the calculation as well.

4.3. Life-tracking methods analysis

- **Tracking.** Prototype consists of an accelerometer and gyroscope usage. Therefore, the tracking of the user activity is done through a physiological sensor. In this case, the measurement and tracking of the activity is based on the of sport related measures.
- **Triggering.** Active triggering is used here because we trigger user when exercise is complete.
- **Recalling and Revisiting.** In current application prototype we contextualize the data on the iPhone part of the application where we have a table of performed results. Additionally, we count how much work is done (count of sets). And of course visualization is present. We have a UI for both Apple Watches and iPhone.

4.4. Application workflow

To start using application it is mandatory to have an iPhone paired with Apple Watches. After that you can install application. Generally, you can use App Store to download and install it or as in the case if you are a developer you can use XCode to install it directly. Figure 8 represents iOS menu screen with “Life Tracker” application with yellow sun icon already installed. As you can see nothing is impossible, our application has the same excellent appearance as other applications from different vendors.

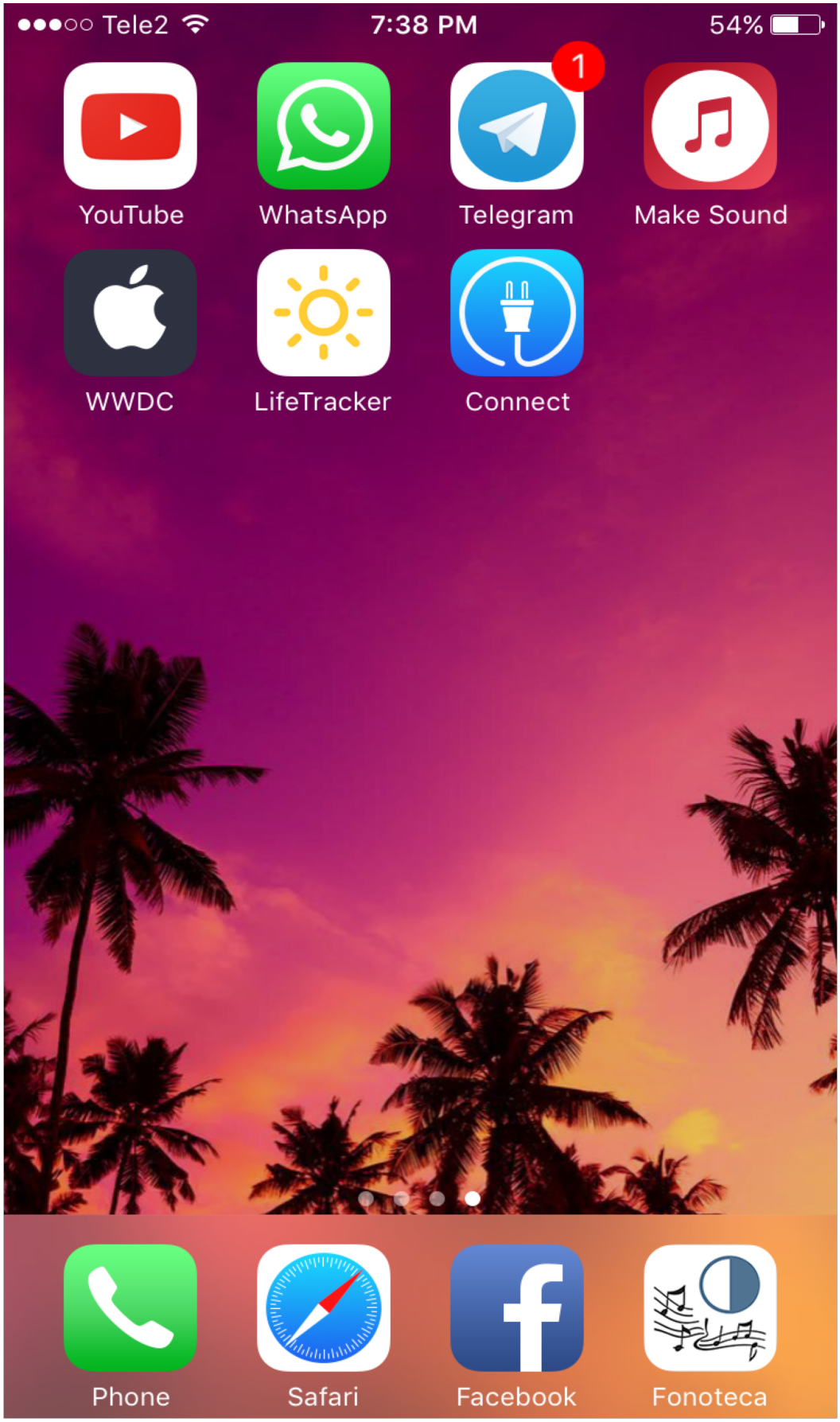

Figure 8. iOS menu screen with LifeTracker application

After application installation, we need to install it on the paired Watches. It is easy to do with iOS native Watches application. Figure 9 shows that the only thing you need to do is to toggle a switch control.

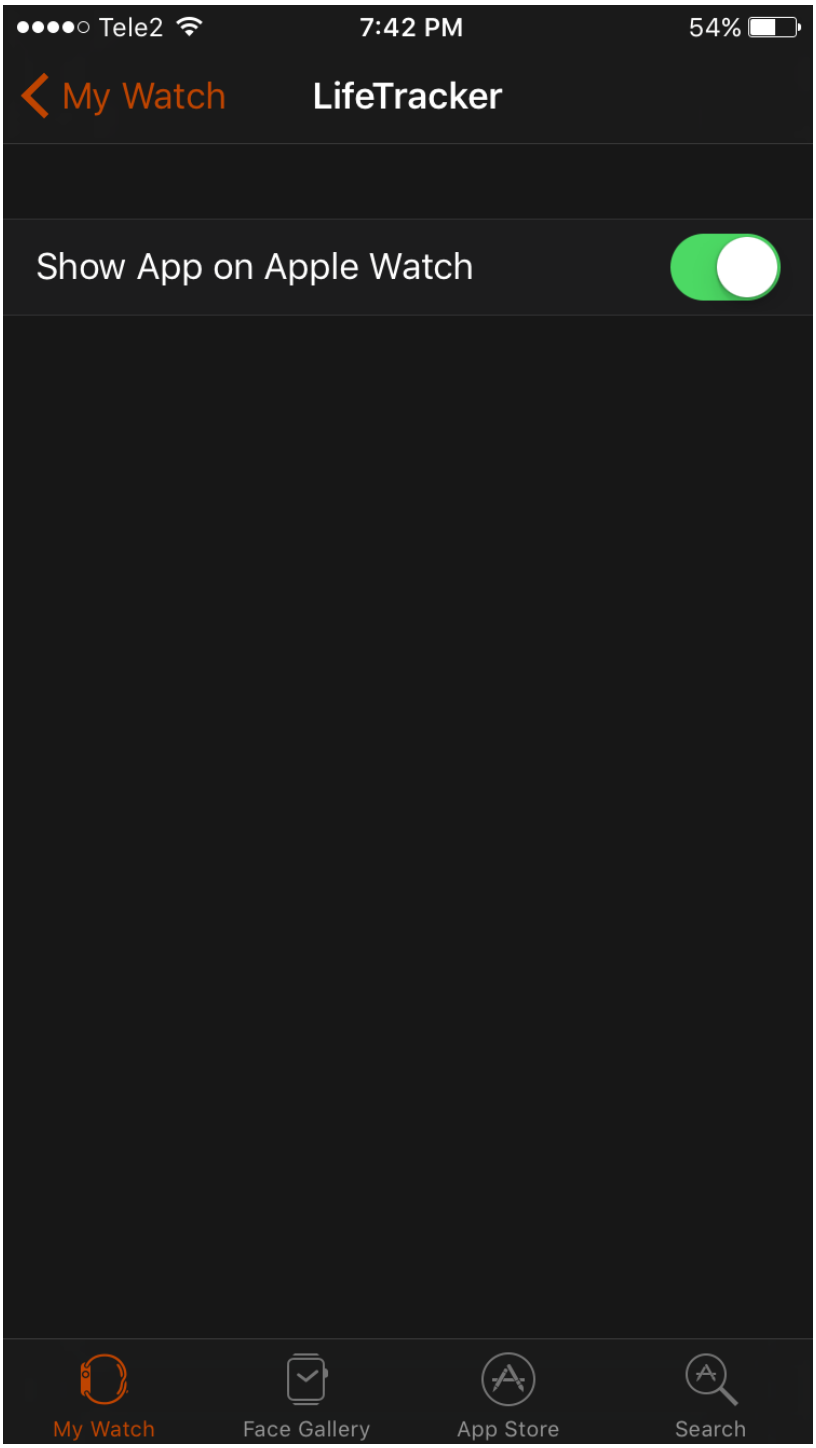

Figure 9. Apples Watch application

After that you need to wait for a while to complete installation. When you see that the application with the same icon appeared on the Apples Watch screen you can start using an amazing “LifeTracker” application. Figure 10 shows an application icon on the Apples Watch screen.

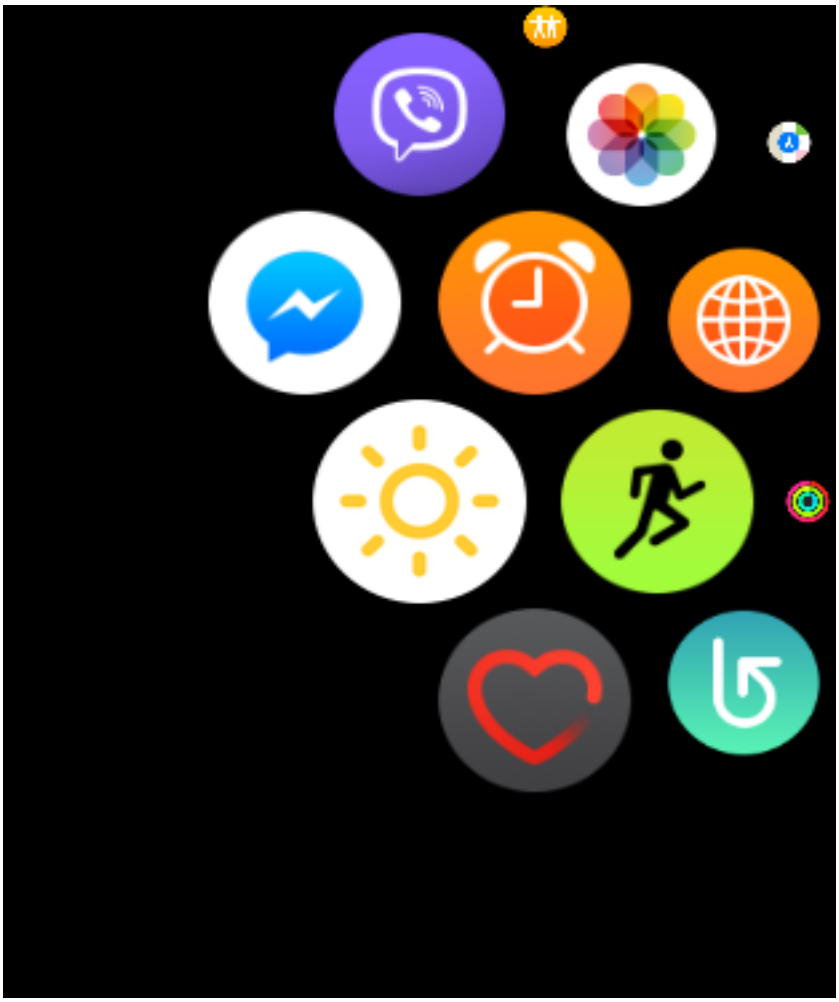

Figure 10. Apple Watch screen and LifeTracker icon

After that you need to tap on LifeTracker icon and welcome screen appears with the list of currently supported exercises.

Figure 11 represents LifeTracker WatchOS application main screen with the list of exercises.

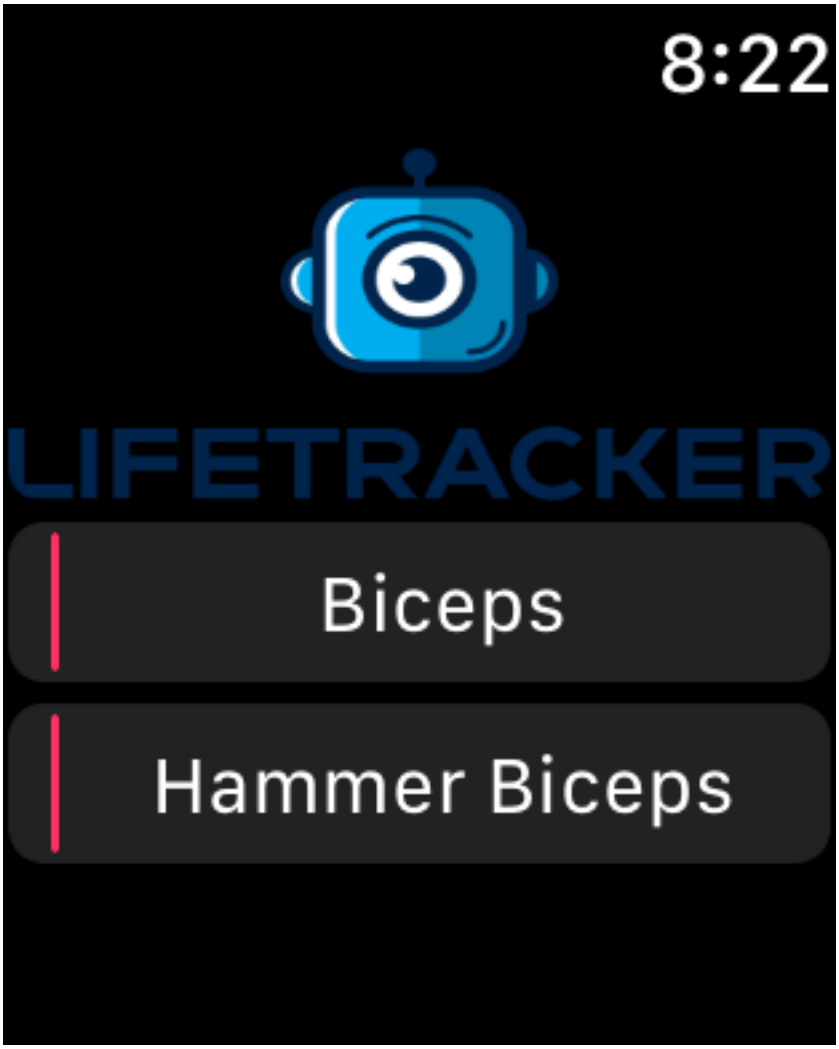

Figure 11. LifeTracker WatchOS main screen

On the main screen you can choose type of exercise you want to perform and after that you will be switched to the exercise screen which is shown on the Figure 12.

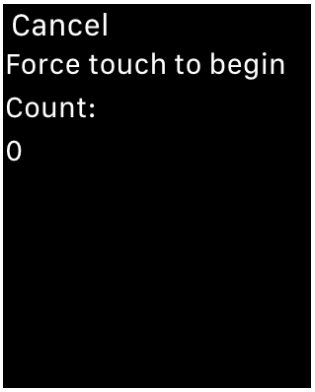

Figure 12. LifeTracker exercise screen

To start workout you need to do a force touch onto screen and the following dialog on the Figure 13 appears.

Figure 13. Force touch menu

Here you can start or stop your workout. Once you tap start you can begin your exercise. Let's assume that you have chosen "Biceps", then you need to perform it 8 times to complete. 8 is a hardcoded value, but it is easy to change it or make it user-friendly. Once you finished the you will be notified with a Haptic type signal with sound and a user interface that tells you about successful completion as on the Figure 14.

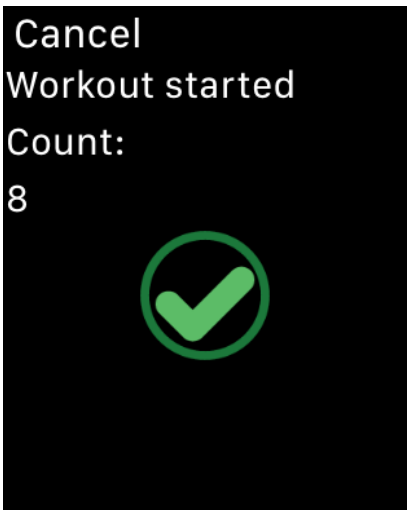

Figure 14. LifeTracker completion sign

And that is it, you finished your workout. If you want to have some statistics, then you can use iOS part of LifeTracker application. If you launch an iOS LifeTracker part, then you will see a list of completed exercises and dates respectively as on the Figure 15.

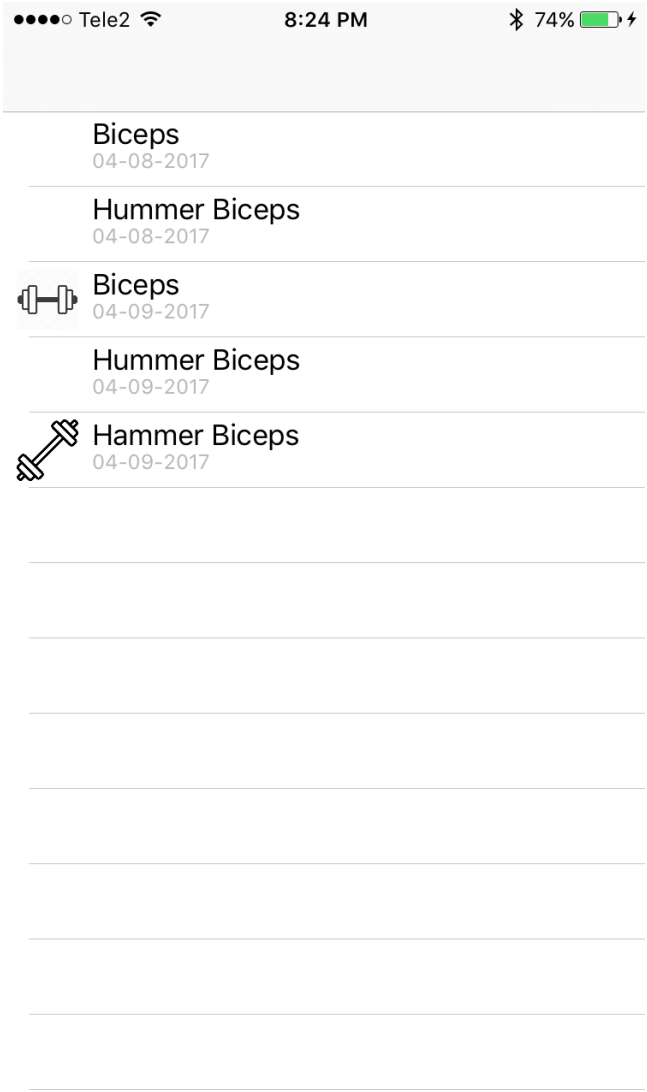

Figure 15. iOS part initial view controller

If you wish to see exercises detail view, then you can tap on the table row and observe some details. As for the prototype application it is enough to provide some simple data such as type, name, time consumed, amount of sets and date. Figure 16 highlights this data and shows exercises image to make it more presentable.

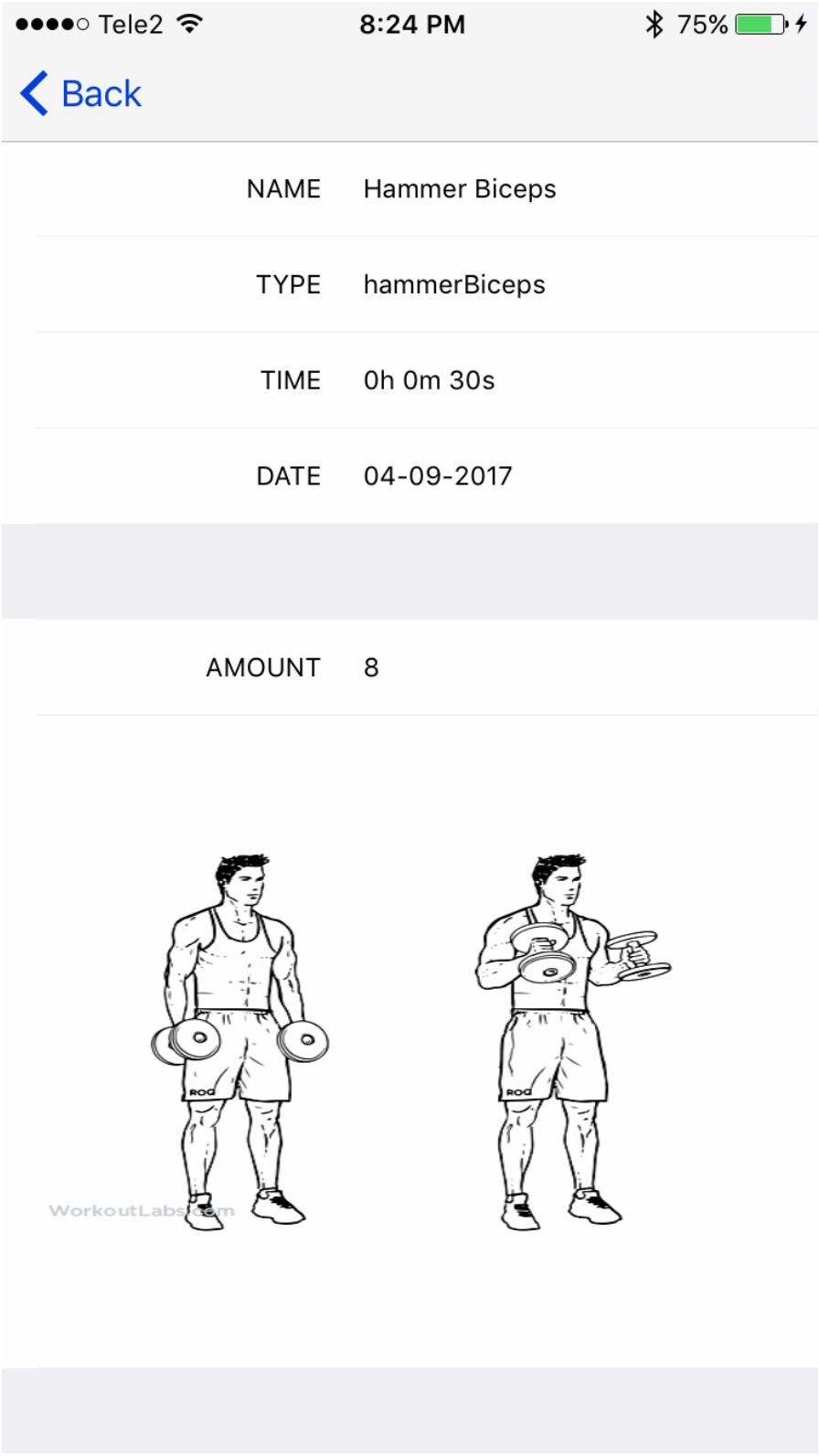

Figure 16. iOS part exercise detail view controller

That is it! It is fair to say that this application was developed thoroughly using Apples Human design guidelines and architecture patterns that enable code extendibility, testability and maintainability.

Summary

Current chapter shows us how to create life-tracking application from scratch. From the provided information it is easy to say that it is not hard to create your own application. You can use any available device/sensors you need (Apple Watch, iPhone, Samsung, Gear etc.) or you can create your own device like many companies. Runtastic [43] is a great example of such company. They have developed their own devices and software as well.

5. Conclusion

To sum up it can be said that life-tracker application is a method and approach to make our life easier and more colourful. We discovered that life-tracking is not something new, it has its own community and it is worth to say that this community(QS) is quite large.
Currently, there is an enormous market of life-tracking applications and you can download any application you need. It can be a sleep tracking application, steps calculator or some other activity tracker. As nowadays mobile applications on the top of the use this is a list of some useful mobile applications that can be helpful:

- Sleep Cycle: Sleep Cycle is an iPhone app that allows users to track their sleep. It creates a nightly record of users' sleep by analyzing the measurements from the device accelerometer.
- Trixie Tracker: application conceived to keep tracking of a baby's life; e.g., naps taken or bottles fed. Its goal is to discover patterns and detect sleep schedules.
- oneLog: application for iPhone that allows tracking of any sort of information
- or activity.
- My Tracks: application that uses GPS position to track several parameters (path, speed, elevation) while the user walks, runs, or does any other outdoors activity. Data are available live, announcements notify users about their progress, and annotation features are provided.
- Memolane: this application collects and connects content from other sites such as Facebook or Twitter. Its goal is to create an easy way of exploring past social network content.

It is just an example list, but you can find many other applications on App Store or Google Play Market.
Additionally, there are a lot of methods how to calculate and measure tracked data and we highlighted some of them:

- Tracking cues: capturing and keeping track of certain data
- Triggering: gathering data and analysis
- Recalling and revisiting experiences: recalling and revisiting past experiences through the enrichment and presentation of data in order to make sense of past experiences.

To make these methods clear, there were two exemplary applications report provided. For Philips DirectLife and Moodscope.

The main part of the thesis was a prototype application for fitness tracking. This application was developed to prove that it is feasible to make it by your own and you do not need any high-level math or physics skills. Everything you need is already provided in mobile devices, therefore you do not need to concentrate your attention on hardware, you can be concentrated on software and already provided libraries for gathering data. In case of iOS development it is really easy as most of the tools and libraries already exists and they are in the heart of the native apples development, so the only thing you need is a devices for testing(as in our case – iPhone and Apple Watch), but you can even do it with simulators provided by xCode and of course you need to have a comprehensive knowledge in programming.

References

[1] Margreet Riphagen. (2013) “Learning tomorrow: Visualising student and staff’s daily activities and reflect on it” (pdf). ICERI2013

[2] US Patent application 20020198685

[3] Swan, Melanie (June 2013). "The Quantified Self: Fundamental Disruption in Big Data Science and Biological Discovery". Big Data. 1 (2)

[4] Singer, Emily. "The Measured Life". MIT. Retrieved February 14 2017.

[5] Wolf, Gary. "Quantified Self". Gary Wolf. Archived from the original on 2012-03-26. Retrieved February 14 2017.

[6] Wolf, Gary. "The quantified self". TED (conference). Retrieved February 14 2017.

[7] "Invasion of the body hackers". Financial Times. 2011-06-10. Archived from the original on 2012-03-26.

[8] Hesse, Monica (September 9, 2008). "Bytes of Life". Washington Post. Retrieved February 18 2017.

[9] Stinson, Ben. "How wearables became the key tech trend of 2014". TechRadar. Retrieved February 20 2017

[10] "Edelman - Conversations - The Quantified Self and Corporate Wellness". 22 February 2013.

[11] No Author "When IoE Gets Personal: The Quantified Self Movement!"

[12] Lee, Victor R (1 January 2013). "The Quantified Self (QS) movement and some emerging opportunities for the educational technology field". 53 (6).

[13] Heussner, Ki Mae (11 July 2013). "The quantified baby: Do parents really need infant-ready sensor tech?". GigaOM. Retrieved February 21 2017.

[14] Higginbotham, Stacy (18 April 2013). "Podcast: How the internet of things may make parents less worried but more neurotic". GigaOM. Retrieved February 25 2017.

[15] Brooks, Ross (9 September 2013). "Baby Jumpsuit Reports Nighttime Activity Levels To Anxious Parents Baby Jumpsuit Reports Nighttime Activity Levels To Anxious Parents". PSFK. Retrieved February 28 2017.

[16] URL: http://daytum.com Accessed March 11, 2017.

[17] URL: http://www.trixietracker.com Accessed March 12, 2017.

[18] URL: http://www.sleepcycle.com/ Accessed March 14, 2017.

[19] URL: http://www.nikeplus.com Accessed March 14, 2017.

[20] URL: http://research.microsoft.com/en- us/um/cambridge/projects/sensecam Accessed March 14, 2017.

[21] A. J. Sellen and S. Whittaker. Beyond total capture: a constructive critique of lifelogging. *Communications of the ACM*, 53:70–77, May 2010. ISSN 0001-0782. doi: 10.1145/1735223.1735243.

[22] Li, A. K. Dey, and J. Forlizzi. Understanding my data, myself: supporting self- reflection with ubicomp technologies. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 405–414, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0630-0. doi: 10.1145/2030112.2030166.

[23] URL: http://www.rescuetime.com Accessed March 15, 2017.

[24] URL: http://connect.garmin.com Accessed March 20, 2017.

[25] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

[26] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *In HUC 1999: Proceedings of the 1st international sympo- sium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[27] URL: http://www.directlife.philips.com Accessed March 20, 2017.

[28] D. Watson, L. A. Clark, and A. Tellegen. Development and validation of brief measures of positive and negative affect: the panas scales. *Journal of Personality and Social Psychology*, 54(6):1063–1070, 1988. URL http://www.ncbi.nlm. nih.gov/pubmed/3397865.

[29] H. Gimpel, M. Nissen, and R. A. Go¨rlitz. Quantifying the Quantified Self: A Study on the Motivations of Patients to Track Their Own Health. In *ICIS 2013 Proceedings (International Conference on Information Systems)*, 2013. URL http://aisel.aisnet.org/icis2013/proceedings/HealthcareIS/3/.

[30] E. K. Choe, N. B. Lee, B. Lee, W. Pratt, and J. A. Kientz. Understanding quantified- selfers' practices in collecting and exploring personal data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1143– 1152, 2014. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557372.

[31] J. Rooksby, M. Rost, A. Morrison, and M. Chalmers. Personal Tracking As Lived Informatics. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1163–1172, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557039.

[32] E. Isaacs, A. Konrad, A. Walendowski, T. Lennig, V. Hollis, and S. Whittaker. Echoes from the past: How technology mediated reflection improves well- being. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1071–1080. ACM, 2013.

[33] M. E. Morris, Q. Kathawala, T. K. Leen, E. E. Gorenstein, F. Guilak, M. Labhard, and W. Deleeuw. Mobile Therapy: Case Study Evaluations of a Cell Phone Application for Emotional Self-Awareness. *Journal of Medical Internet Research*, 12:10, 2010.

[34] J. Froehlich, T. Dillahunt, P. Klasnja, J. Mankoff, S. Consolvo, B. Harrison, and J. A. Landay. Ubigreen: Investigating a mobile tool for tracking and supporting green transportation habits. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1043–1052, 2009. ISBN 978-1- 60558-246-7. doi: 10.1145/1518701.1518861.

[35] J. Maitland, S. Sherwood, L. Barkhuus, I. Anderson, M. Hall, B. Brown, M. Chalmers, and H. Muller. *Increasing the Awareness of Daily Activity Levels with Pervasive Computing*. Institute of Electrical and Electronics Engineers (IEEE), 2006.

[36] D. McDuff, A. Karlson, A. Kapoor, A. Roseway, and M. Czerwinski. AffectAura: an intelligent system for emotional memory. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 849– 858, 2012. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2208525.

[37] P. Sengers, K. Boehner, S. David, and J. J. Kaye. Reflective Design. In *Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense and Sensibility*, CC '05, pages 49–58, New York, NY, USA, 2005. ACM. ISBN 1-59593-203-8. doi: 10.1145/1094562.1094569.

[38] S. Consolvo, J. A. Landay, and D. W. McDonald. Designing for behavior change in everyday life. *Computer*, 42:86–89, 2009a. ISSN 0018-9162. doi: http://doi. ieeecomputersociety.org/10.1109/MC.2009.185.

[39] K. K. Rachuri, C. Mascolo, P. J. Rentfrow, and C. Longworth. EmotionSense: A Mobile Phones based Adaptive Platform for Experimental Social Psychology Research. *International Studies*, pages 281–290, 2010. doi: 10.1145/1864349. 1864393.

[40] K. Tollmar, F. Bentley, and C. Viedma. Mobile Health Mashups: Making sense of multiple streams of wellbeing and contextual data for presentation on a mobile device. In *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2012 6th International Conference on*, pages 65–72, May 2012.

[41] URL: https://www.bodybuilding.com/exercises/main/popup/name/hammer-curls Accessed April 20, 2017.

[42] URL: https://developer.apple.com/reference/coremotion Accessed April 21, 2017.

[43] URL: https://developer.apple.com/reference/watchconnectivity Accessed April 22, 2017.

[44] URL: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html?utm_source=iosstash.io Accessed April 22, 2017.

[45] URL: https://developer.apple.com/reference/uikit Accessed April 22, 2017.

[46] URL: https://www.runtastic.com Accessed April 28, 2017.

[47] G. Wolf. Our Three Prime Questions, September 2011. URL http: //quantifiedself.com/2011/09/our-three-prime-questions/. [Accessed January, 2015].

[48] G. Bell and J. Gemmell. *Your Life, Uploaded*. Plume, October 2010. ISBN 0452296560, 9780452296565. URL http://totalrecallbook.com/.

[49] J. Gemmell, G. Bell, and R. Lueder. MyLifeBits: a personal database for everything. *Communications of the ACM (CACM)*, 49(1):88–95, January 2006. URL http://research.microsoft.com/apps/pubs/default. aspx?id=64157. also as MSR-TR-2006-23.

[50] S. Hodges, L. Williams, E. Berry, S. Izadi, J. Srinivasan, A. Butler, G. Smyth, N. Kapur, and K. Wood. SenseCam: A Retrospective Memory Aid. In *Proceed- ings of the 8th International Conference of Ubiquitous Computing (UbiComp 2006)*, pages 177–193. Springer Verlag, September 2006. URL http://research. microsoft.com/apps/pubs/default.aspx?id=132537.

Appendix 1 – Source code for iOS application part

- LTAppDeleagate.swift -

```
//
//  AppDelegate.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 3/14/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import UIKit
import CoreData
import WatchConnectivity

@UIApplicationMain
class LTAppDelegate: UIResponder, UIApplicationDelegate  {

    var window: UIWindow?

    var session: WCSession? {
        didSet {
            if let session = session {
                session.delegate = self
                session.activate()
            }
        }
    }

    var storage: Storage = Storage.shared
    var fetchManagedObjectContext: NSManagedObjectContext?


    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        if WCSession.isSupported() {
            session = WCSession.default()
        }


        NotificationCenter.default.addObserver(forName:
NSNotification.Name.NSManagedObjectContextDidSave, object: nil,
queue: nil) { (note) in
            let lockQueue = DispatchQueue(label: "com.sync.q")
            lockQueue.sync() {
                let mainContext = self.storage.context
                let otherMoc: NSManagedObjectContext =
note.object as! NSManagedObjectContext

             if(otherMoc.persistentStoreCoordinator ==
mainContext.persistentStoreCoordinator){
                    if(otherMoc != mainContext){
```

```
                        mainContext.perform({

mainContext.mergeChanges(fromContextDidSave: note)
                        })
                    }
                }
            }
        }

        return true
    }

    func applicationWillResignActive(_ application:
UIApplication) {
        // Sent when the application is about to move from
active to inactive state. This can occur for certain types of
temporary interruptions (such as an incoming phone call or SMS
message) or when the user quits the application and it begins
the transition to the background state.
        // Use this method to pause ongoing tasks, disable
timers, and invalidate graphics rendering callbacks. Games
should use this method to pause the game.
    }

    func applicationDidEnterBackground(_ application:
UIApplication) {
        // Use this method to release shared resources, save
user data, invalidate timers, and store enough application state
information to restore your application to its current state in
case it is terminated later.
        // If your application supports background execution,
this method is called instead of applicationWillTerminate: when
the user quits.
    }

    func applicationWillEnterForeground(_ application:
UIApplication) {
        // Called as part of the transition from the background
to the active state; here you can undo many of the changes made
on entering the background.
    }

    func applicationDidBecomeActive(_ application:
UIApplication) {
        // Restart any tasks that were paused (or not yet
started) while the application was inactive. If the application
was previously in the background, optionally refresh the user
interface.


    }

    func applicationWillTerminate(_ application: UIApplication)
{
        // Called when the application is about to terminate.
Save data if appropriate. See also
applicationDidEnterBackground:.
```

```
    }

    /**
     Loader managed object context
     - Returns:
NSManagedObjectContext(privateQueueConcurrencyType)
     */
    func loaderManagedObjectContext() -> NSManagedObjectContext{
        if(self.fetchManagedObjectContext != nil){
            return self.fetchManagedObjectContext!
        }
        let parent = self.storage.context
        self.fetchManagedObjectContext =
NSManagedObjectContext(concurrencyType:
.privateQueueConcurrencyType)
        self.fetchManagedObjectContext?.parent = parent
        return self.fetchManagedObjectContext!
    }


}

extension LTAppDelegate: WCSessionDelegate {

    func session(_ session: WCSession, didReceiveMessage
    message: [String : Any], replyHandler: @escaping ([String :
Any]) ->
        Void){
        if let displayName = message["displayName"] as? String,
let type = message["type"] as? String, let countGoal =
message["countGoal"] as? NSNumber, let time = message["time"]
as? NSNumber {

            ExerciseDataManager.addExerciseAsync(name:
displayName, type: type, time: time, countGoal: countGoal,
completionBlock: {

            })
            replyHandler([:])
        }
    }

    func session(_ session: WCSession, activationDidCompleteWith
activationState: WCSessionActivationState, error: Error?) {

    }

    func sessionDidBecomeInactive(_ session: WCSession) {

    }

    func sessionDidDeactivate(_ session: WCSession) {

    }

}
```

- NSPersistentStoreCoordinator+Extension.swift -

```
//
//  NSPersistentStoreCoordinator+Extension.swift
//
//  Created by Sergey Spivakov on 1/30/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

/// NSPersistentStoreCoordinator extension

import Foundation
import CoreData

/// Extension to support iOS 9 & iOS10+
extension NSPersistentStoreCoordinator {

    /// NSPersistentStoreCoordinator error types
    public enum CoordinatorError: Error {
        /// .momd file not found
        case modelFileIsNotFound
        /// NSManagedObjectModel creation fail
        case modelCreation
        /// Gettings document directory fail
        case storePathIsNotFound
    }

    /// Return NSPersistentStoreCoordinator object
    static func coordinator(name: String) throws ->
NSPersistentStoreCoordinator? {

        guard let modelURL = Bundle.main.url(forResource: name,
withExtension: "momd") else {
            throw CoordinatorError.modelFileIsNotFound
        }

        guard let model = NSManagedObjectModel(contentsOf:
modelURL) else {
        throw CoordinatorError.modelCreation
        }

        let coordinator =
NSPersistentStoreCoordinator(managedObjectModel: model)

        guard let documents = FileManager.default.urls(for:
.documentDirectory, in: .userDomainMask).last else {
            throw CoordinatorError.storePathIsNotFound
        }

        do {
            let url =
documents.appendingPathComponent("\(name).sqlite")
            let options = [
NSMigratePersistentStoresAutomaticallyOption : true,
```

```
NSInferMappingModelAutomaticallyOption : true ]
            try coordinator.addPersistentStore(ofType:
NSSQLiteStoreType, configurationName: nil, at: url, options:
options)
        } catch {
            throw error
        }

        return coordinator
    }
}

struct Storage {

    static var shared = Storage()

    @available(iOS 10.0, *)
    private lazy var persistentContainer: NSPersistentContainer
= {
        let container = NSPersistentContainer(name: "Model")
        container.loadPersistentStores { (storeDescription,
error) in
            print("CoreData: Inited \(storeDescription)")
            guard error == nil else {
                print("CoreData: Unresolved error
\(String(describing: error))")
                return
            }
        }
        return container
    }()

    private lazy var persistentStoreCoordinator:
NSPersistentStoreCoordinator? = {
        do {

     return try NSPersistentStoreCoordinator.coordinator(name:
"Model")
        } catch {
            print("CoreData: Unresolved error \(error)")
        }
        return nil
    }()

    private lazy var managedObjectContext:
NSManagedObjectContext = {
        let coordinator = self.persistentStoreCoordinator
        var managedObjectContext =
NSManagedObjectContext(concurrencyType:
.mainQueueConcurrencyType)
        managedObjectContext.persistentStoreCoordinator =
coordinator
        return managedObjectContext
    }()

    // MARK: Public methods
```

```
    enum SaveStatus {
        case saved, rolledBack, hasNoChanges
    }

    var context: NSManagedObjectContext {
        mutating get {
            if #available(iOS 10.0, *) {
                return persistentContainer.viewContext
            } else {
                return managedObjectContext
            }
        }
    }

    mutating func save() -> SaveStatus {
        if context.hasChanges {
            do {
                try context.save()
                return .saved
            } catch {
                context.rollback()
                return .rolledBack
            }
        }
        return .hasNoChanges
    }

}
```

- ExerciseDataManager.swift -

```
//
//  ExerciseDataManager.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 4/7/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import Foundation
import CoreData
import UIKit


class ExerciseDataManager{
    class func addExerciseAsync(name: String, type: String,
time: NSNumber, countGoal: NSNumber, completionBlock: @escaping
() -> Void){
        guard let appDelegate =
            UIApplication.shared.delegate as? LTAppDelegate else
{
                return
        }
        DispatchQueue.global().async {
            let coordinator =
appDelegate.storage.context.persistentStoreCoordinator
```

```
            let managedContext =
NSManagedObjectContext(concurrencyType:
.privateQueueConcurrencyType)
            managedContext.persistentStoreCoordinator =
coordinator
            managedContext.mergePolicy =
NSMergeByPropertyObjectTrumpMergePolicy
            var exercise: Exercise?
            exercise =
NSEntityDescription.insertNewObject(forEntityName: "Exercise",
into: managedContext) as? Exercise
            exercise?.count = countGoal.int16Value
            exercise?.name = name
            exercise?.time = time.int64Value
            exercise?.date = NSDate()
            exercise?.type = type

            do {
                try managedContext.save()
            } catch let error as NSError {
                print("Could not save. \(error),
\(error.userInfo)")
            }
            completionBlock()
        }
    }
}
```

- LTExercise.swift -

```
//
//  Exercise.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 3/31/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import UIKit

class LTExercise {
    var displayName: String
    var type: LTExerciseType
    var countGoal = 8
    var time: Int64?

    var startDate: Date?
    var endDate: Date?
    init(displayName: String, type: LTExerciseType, countGoal:
Int!){
        self.displayName = displayName
        self.type = type
        if let cGoal = countGoal {
            self.countGoal = cGoal
        }
    }
```

```
}
```

- LTExerciseType.swift -

```
//
//  CalculatorType.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 3/31/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import Foundation


enum LTExerciseType : String{
    case biceps = "biceps"
    case hammerBiceps = "hammerBiceps"
}
```

- LTViewController.swift -

```
//
//  ViewController.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 3/14/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import UIKit
import CoreData

class LTViewController: LTBaseViewController {
    // MARK: - Properties
    @IBOutlet var tableView: UITableView!
    var currentlySelectedRow: Exercise?

    static let segueToDetailExercise = "segueToDetailExercise"

    lazy var refreshControl: UIRefreshControl = {
        let refreshControl = UIRefreshControl()
        refreshControl.backgroundColor = UIColor.purple
        refreshControl.tintColor = UIColor.white
        refreshControl.addTarget(self, action:
#selector(LTViewController.handleRefresh(refreshControl:)), for:
UIControlEvents.valueChanged)

        return refreshControl
    }()
```

```
    fileprivate lazy var fetchedResultsController:
NSFetchedResultsController<Exercise> = {

        let appDelegate = UIApplication.shared.delegate as?
LTAppDelegate

        // Create Fetch Request
        let fetchRequest: NSFetchRequest<Exercise> =
Exercise.fetchRequest()

        // Configure Fetch Request
        fetchRequest.sortDescriptors = [NSSortDescriptor(key:
"date", ascending: true)]

        // Create Fetched Results Controller
        let fetchedResultsController =
NSFetchedResultsController(fetchRequest: fetchRequest,
managedObjectContext: (appDelegate?.storage.context)!,
sectionNameKeyPath: nil, cacheName: nil)

        // Configure Fetched Results Controller
        fetchedResultsController.delegate = self

       return fetchedResultsController
    }()

    ///Returns last update string
    func getRefreshControlString() -> NSAttributedString{
        let formatter =  DateFormatter()
        formatter.dateFormat = "MMM d, h:mm a"
        let title = "Last update: \(formatter.string(from:
Date()))"
        let attrDict = [NSForegroundColorAttributeName:
UIColor.white]
        let attrStr: NSAttributedString =
NSAttributedString(string: title, attributes: attrDict)

        return attrStr
    }


    /// Updates tableView data
    func handleRefresh(refreshControl: UIRefreshControl) {
        updateView()

        let attrStr: NSAttributedString =
self.getRefreshControlString()
        self.refreshControl.attributedTitle = attrStr
        self.refreshControl.endRefreshing()
    }

    fileprivate func updateView() {
        var hasExercises = false
        self.loadingView?.show(whereToShow: self)

        do {
            try self.fetchedResultsController.performFetch()
```

```
        } catch {
            let fetchError = error as NSError
            print("Unable to Perform Fetch Request")
            print("\(fetchError),
\(fetchError.localizedDescription)")
        }


        if let exercises =
fetchedResultsController.fetchedObjects {
            hasExercises = exercises.count > 0
        }

        tableView.isHidden = !hasExercises

        self.loadingView?.hide(whereToHide: self)
    }
```

- LTViewController.swift -

```
     // MARK: - View Life Cycle

    override func viewDidLoad() {
        super.viewDidLoad()
        self.tableView.addSubview(self.refreshControl)

self.tableView.register(UITableViewCell.self,forCellReuseIdentif
ier: "Cell")
        self.tableView.autoresizingMask = [.flexibleWidth,
.flexibleHeight]
        self.updateView()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}

extension LTViewController: UITableViewDataSource {

    ///Returns UILabel with specified text
    func getBackgroundMessageLabel(text:String, bounds:CGRect) -
> UILabel{

        let messageLabel = UILabel(frame: CGRect(x: 0, y: 0,
width: bounds.size.width, height: bounds.size.height))
        messageLabel.text = text;
        messageLabel.textColor = UIColor.black
        messageLabel.numberOfLines = 0;
        messageLabel.textAlignment = NSTextAlignment.center;
        messageLabel.font = UIFont(name: "Palatino-Italic",
size: 20)
        messageLabel.sizeToFit()
```

```
        return messageLabel
    }

    func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        let exercises = fetchedResultsController.fetchedObjects

        if exercises == nil || exercises?.count == 0{
            // Display a message when the table is empty
            let messageLabel =
self.getBackgroundMessageLabel(text: "No data is currently
available", bounds: self.view.bounds)
            self.tableView.backgroundView = messageLabel
            self.tableView.separatorStyle =
UITableViewCellSeparatorStyle.none
            return 0
         }else{
            self.tableView.backgroundView = nil
            return exercises!.count
        }
    }

    func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        guard let cell =
tableView.dequeueReusableCell(withIdentifier:
ExerciseTableViewCell.reuseIdentifier, for: indexPath) as?
ExerciseTableViewCell else {
            fatalError("Unexpected Index Path")
        }

        // Fetch
        let exercise = fetchedResultsController.object(at:
indexPath)

        // Configure Cell
        cell.nameLbl.text = exercise.name

        //date conversion
        let dateFormatter = DateFormatter()
        dateFormatter.locale = Locale.init(identifier: "en_US")
        dateFormatter.dateFormat = "MM-dd-yyyy"

        cell.dateLbl.text = dateFormatter.string(from:
exercise.date! as Date)
        if let type = exercise.type {
            switch type {
            case LTExerciseType.biceps.rawValue:
                cell.typeIV.image = UIImage(named: "bicepsGym")
                break
            case LTExerciseType.hammerBiceps.rawValue:
                cell.typeIV.image = UIImage(named:
"hummerBiceps")
                break

            default:
                break
```

```
            }
        }
        return cell
    }
}

extension LTViewController: NSFetchedResultsControllerDelegate {
    func controllerWillChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>) {
        tableView.beginUpdates()
    }
    func controllerDidChangeContent(_ controller:
NSFetchedResultsController<NSFetchRequestResult>) {
        tableView.endUpdates()

        updateView()
    }

    func controller(_ controller:
NSFetchedResultsController<NSFetchRequestResult>, didChange
anObject: Any, at indexPath: IndexPath?, for type:
NSFetchedResultsChangeType, newIndexPath: IndexPath?) {
        switch (type) {
        case .insert:
            if let indexPath = newIndexPath {
                tableView.insertRows(at: [indexPath], with:
.fade)
            }
            break;
        default:
            print("...")
        }
    }
}

//MARK: - UITableViewDelegate implementation
extension LTViewController:UITableViewDelegate{
    func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
        self.currentlySelectedRow =
fetchedResultsController.object(at: indexPath)
        self.performSegue(withIdentifier:
LTViewController.segueToDetailExercise, sender: self)
    }
}
//MARK: - Transition between VC implementation
extension LTViewController{
    override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
        if(segue.identifier ==
LTViewController.segueToDetailExercise){
            guard let destVC = segue.destination as?
LTExerciseDetailTableViewController else{
                return
            }
            guard let exercise = self.currentlySelectedRow else{
                return
```

```
            }
            destVC.exercise = exercise
        }
    }
}
```

- LTBaseViewController.swift -

```
//
//  LTBaseViewController.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 4/8/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import UIKit

class LTBaseViewController: UIViewController {

    /// Custom loading view
    var loadingView: LTProgressIndicator?

    override func viewDidLoad() {
        super.viewDidLoad()

        self.loadingView =
LTProgressIndicator.instantiateFromNib(nibName:
"LTProgressIndicator", bundle: Bundle(for: type(of: self)),
frame: self.view.frame, bounds:self.view.bounds, caption: "")
        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to
do a little preparation before navigation
    override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
        // Get the new view controller using
segue.destinationViewController.
        // Pass the selected object to the new view controller.
    }
    */

}
```

- LTExerciseDetailTableViewController.swift -

```
//
//  LTExerciseDetailTableViewController.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 4/8/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import UIKit

class LTExerciseDetailTableViewController: UITableViewController
{

    //MARK: - Properties
    @IBOutlet weak var imgIV: UIImageView!
    @IBOutlet weak var amountLbl: UILabel!
    @IBOutlet weak var timeLbl: UILabel!
    @IBOutlet weak var dateLbl: UILabel!
    @IBOutlet weak var typeLbl: UILabel!
    @IBOutlet weak var nameLbl: UILabel!

    var exercise: Exercise?

    override func viewDidLoad() {
        super.viewDidLoad()

        guard let exercise = exercise else {
            return
        }

        nameLbl.text = exercise.name
        if let type = exercise.type{
            typeLbl.text = type

            switch type {
            case LTExerciseType.biceps.rawValue:
                imgIV.image = UIImage(named: "bicepsBigImage")
            case LTExerciseType.hammerBiceps.rawValue:
                imgIV.image = UIImage(named:
"hummerBicepsBigImage")
            default:
                break
            }
        }

        //date conversion
        let dateFormatter = DateFormatter()
        dateFormatter.locale = Locale.init(identifier: "en_US")
        dateFormatter.dateFormat = "MM-dd-yyyy"

        dateLbl.text = dateFormatter.string(from: exercise.date!
as Date)
        let timeComponents =
secondsToHoursMinutesSeconds(seconds: exercise.time)
```

```
        timeLbl.text = "\(timeComponents.hours)h
\(timeComponents.minutes)m \(timeComponents.seconds)s"
        amountLbl.text = "\(exercise.count)"


    }

    func secondsToHoursMinutesSeconds (seconds : Int64) ->
(hours: Int64, minutes: Int64, seconds: Int64) {
        return (seconds / 3600, (seconds % 3600) / 60, (seconds
% 3600) % 60)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

Appendix 2 – Source code for WatchOS application part

- LTCalculator.swift -

```
//
//  Calculator.swift
//  SwingWatch
//
//  Created by Sergey Spivakov on 3/30/17.
//  Copyright © 2017 Apple Inc. All rights reserved.
//

import Foundation
import CoreMotion

protocol LTCalculator {
    init(delegate: LTMotionManagerDelegate)
    func receivedMotionUpdate(deviceMotion: CMDeviceMotion)
}
```

- LTBicepsCalculator.swift -

```
//
//  BicepsCalculator.swift
//  SwingWatch
//
//  Created by Sergey Spivakov on 3/30/17.
//  Copyright © 2017 Apple Inc. All rights reserved.
//

import Foundation
import CoreMotion

class LTBicepsCalculator: LTCalculator{

    var counter: Int = 0
    var delegate: LTMotionManagerDelegate
    let changeMagnituteBuffer = LTRunningBuffer(size: 10)


    var atInitialPos = true
    var atEndPos = false
    var changeSumThreshold = 350.0
    var yawMaxThreshold = 50.0
    var yawMinThreshold = 10.0

    required init(delegate: LTMotionManagerDelegate){
        self.delegate = delegate
    }

    func receivedMotionUpdate(deviceMotion: CMDeviceMotion){
        let gravity = deviceMotion.gravity

        let myPitch = degrees(radians: gravity.x)
        changeMagnituteBuffer.addSample(myPitch)
        let myYaw = degrees(radians: gravity.z)
```

```
        let bufferSum = changeMagnituteBuffer.sum()
        if(myYaw > yawMinThreshold && myYaw < yawMaxThreshold){
            if (atInitialPos && (bufferSum < 0) && (bufferSum <=
-changeSumThreshold)){
                incrementCount()
                changePos()
                print("INCREMENTED")
            }else if (atEndPos && (bufferSum > 0) && (bufferSum
>= changeSumThreshold)){
                changePos()
                print("NEXT SET")
            }
        }
    }

    fileprivate func changePos(){
        if atInitialPos{
            atInitialPos = false
            atEndPos = true
        }else{
            atInitialPos = true
            atEndPos = false
        }
    }

    fileprivate func incrementCount(){
        counter = counter + 1
        delegate.didUpdateCount(count: counter)
        if counter == 8{
            if let alarmDel = delegate as?
LTWorkoutAlarmDelegate{
                alarmDel.workoutCompleted()
            }
        }
    }

    func degrees(radians:Double) -> Double {
        return 180 / .pi * radians
    }


}
```

- LTHammerBicepsCalculator.swift -

```
//
//  BicepsCalculator.swift
//  SwingWatch
//
//  Created by Sergey Spivakov on 3/30/17.
//  Copyright © 2017 Apple Inc. All rights reserved.
//

import Foundation
import CoreMotion
```

```
class LTHammerBicepsCalculator: LTCalculator{

    var counter: Int = 0
    var delegate: LTMotionManagerDelegate
    let changeMagnituteBuffer = LTRunningBuffer(size: 10)


    var atInitialPos = true
    var atEndPos = false
    var changeSumThreshold = 350.0
    var yawMaxThreshold = 15.0
    var yawMinThreshold = -10.0

    required init(delegate: LTMotionManagerDelegate){
        self.delegate = delegate
    }

    func receivedMotionUpdate(deviceMotion: CMDeviceMotion){
        let gravity = deviceMotion.gravity

        let myPitch = degrees(radians: gravity.x)
        changeMagnituteBuffer.addSample(myPitch)
        let myYaw = degrees(radians: gravity.z)

        let bufferSum = changeMagnituteBuffer.sum()

        if(myYaw > yawMinThreshold && myYaw < yawMaxThreshold){
            if (atInitialPos && (bufferSum < 0) && (bufferSum <=
-changeSumThreshold)){
                incrementCount()
                changePos()
                print("INCREMENTED")
            }else if (atEndPos && (bufferSum > 0) && (bufferSum
>= changeSumThreshold)){
                changePos()
                print("NEXT SET")
            }
        }
    }

    fileprivate func changePos(){
        if atInitialPos{
            atInitialPos = false
            atEndPos = true
        }else{
            atInitialPos = true
            atEndPos = false
        }
    }

    fileprivate func incrementCount(){
        counter = counter + 1
        delegate.didUpdateCount(count: counter)
        if counter == 8{
            if let alarmDel = delegate as?
LTWorkoutAlarmDelegate{
                alarmDel.workoutCompleted()
```

```
            }
        }
    }

    func degrees(radians:Double) -> Double {
        return 180 / .pi * radians
    }


}
```

- LTRunningBuffer.swift -

```
/*
 Abstract:
 This class manages a running buffer of Double values.
 */

import Foundation

class LTRunningBuffer {
    // MARK: Properties

    var buffer = [Double]()
    var size = 0

    // MARK: Initialization

    init(size: Int) {
        self.size = size
        self.buffer = [Double](repeating: 0.0, count: self.size)
    }

    // MARK: Running Buffer

    func addSample(_ sample: Double) {
        buffer.insert(sample, at:0)
        if buffer.count > size  {
            buffer.removeLast()
        }
    }

    func reset() {
        buffer.removeAll(keepingCapacity: true)
    }

    func isFull() -> Bool {
        return size == buffer.count
    }

    func sum() -> Double {
        return buffer.reduce(0.0, +)
    }

    func min() -> Double {
        var min = 0.0
        if let bufMin = buffer.min() {
```

```
            min = bufMin
        }
        return min
    }

    func max() -> Double {
        var max = 0.0
        if let bufMax = buffer.max() {
            max = bufMax
        }
        return max
    }

    func recentMean() -> Double {
        // Calculate the mean over the beginning half of the
buffer.
        let recentCount = self.size / 2
        var mean = 0.0

        if (buffer.count >= recentCount) {
            let recentBuffer = buffer[0..<recentCount]
            mean = recentBuffer.reduce(0.0, +) /
Double(recentBuffer.count)
        }

        return mean
    }
}
```

- LTCalculatorManager.swift -

```
//
//  CalculatorManager.swift
//  SwingWatch
//
//  Created by Sergey Spivakov on 3/30/17.
//  Copyright © 2017 Apple Inc. All rights reserved.
//

import Foundation

class LTCalculatorManager{
    class func getCalculator(motionManager: LTMotionManager,
exerciseType: LTExerciseType) -> LTCalculator{
        var calculator: LTCalculator!
        switch exerciseType {
        case .biceps:
            calculator = LTBicepsCalculator(delegate:
motionManager)
        case .hammerBiceps:
            calculator = LTHammerBicepsCalculator(delegate:
motionManager)
        }
        return calculator
    }
}
```

- LTMotionManager.swift -

```
/*
 Copyright (C) 2016 Apple Inc. All Rights Reserved.
 See LICENSE.txt for this sample's licensing information

 Abstract:
 This class manages the CoreMotion interactions and
         provides a delegate to indicate changes in data.
 */

import Foundation
import CoreMotion
import WatchKit

/**
 `MotionManagerDelegate` exists to inform delegates of motion
changes.
 These contexts can be used to enable application specific
behavior.
 */
protocol LTMotionManagerDelegate: class {
    func didUpdateCount(count: Int)
}

class LTMotionManager: LTMotionManagerDelegate,
LTWorkoutAlarmDelegate {
    // MARK: Properties

    let motionManager = CMMotionManager()
    let queue = OperationQueue()
    let wristLocationIsLeft =
WKInterfaceDevice.current().wristLocation == .left

    // The app is using 50hz data and the buffer is going to
hold 1s worth of data.
    let sampleInterval = 1.0 / 50.0

    weak var delegate: LTMotionManagerDelegate?

    var calculator: LTCalculator?

    // MARK: Initialization

    init() {
        // Serial queue for sample handling and calculations.
        queue.maxConcurrentOperationCount = 1
        queue.name = "MotionManagerQueue"
    }

    // MARK: Motion Manager

    func startUpdates(exercise: LTExercise) {
        if !motionManager.isDeviceMotionAvailable {
            print("Device Motion is not available.")
            return
        }
        calculator =
```

```
LTCalculatorManager.getCalculator(motionManager: self,
exerciseType: exercise.type)
        // Reset everything when we start.
       // resetAllState()

        motionManager.deviceMotionUpdateInterval =
sampleInterval
        motionManager.startDeviceMotionUpdates(using:
.xArbitraryZVertical, to: queue) { (deviceMotion:
CMDeviceMotion?, error: Error?) in
            if error != nil {
                print("Encountered error: \(error!)")
            }

            if deviceMotion != nil {
                self.processDeviceMotion(deviceMotion!)
            }
        }
    }

    func stopUpdates() {
        if motionManager.isDeviceMotionAvailable {
            motionManager.stopDeviceMotionUpdates()
        }
    }

    func workoutCompleted() {
        if let alarmDel = delegate as? LTWorkoutAlarmDelegate {
            alarmDel.workoutCompleted()
        }
    }

    // MARK: Motion Processing


    func processDeviceMotion(_ deviceMotion: CMDeviceMotion) {
        calculator?.receivedMotionUpdate(deviceMotion:
deviceMotion)
    }

    func didUpdateCount(count: Int) {
        delegate?.didUpdateCount(count:count)
    }

}
```

- LTWorkoutManager.swift -

```
/*
 Copyright (C) 2016 Apple Inc. All Rights Reserved.
 See LICENSE.txt for this sample's licensing information

 Abstract:
 This class manages the HealthKit interactions and provides a
delegate
         to indicate changes in data.
 */

import Foundation
```

```
import HealthKit
import WatchKit
import WatchConnectivity


/**
 `WorkoutManagerDelegate` exists to inform delegates of swing
data changes.
 These updates can be used to populate the user interface.
 */
protocol LTWorkoutManagerDelegate: class {
    func didUpdateCount(manager: LTWorkoutManager, count: Int)
}

protocol LTWorkoutAlarmDelegate: class {
    func workoutCompleted()
}

class LTWorkoutManager: NSObject,
LTMotionManagerDelegate,LTWorkoutAlarmDelegate{
    // MARK: Properties
    let motionManager = LTMotionManager()
    let healthStore = HKHealthStore()

    weak var delegate: LTWorkoutManagerDelegate?
    var session: HKWorkoutSession?
    var exercise: LTExercise!

    // MARK: Initialization

    init(exercise: LTExercise) {
        super.init()
        motionManager.delegate = self
        self.exercise = exercise

    }

    // MARK: WorkoutManager

    func startWorkout() {
        // If we have already started the workout, then do
nothing.
        if (session != nil) {
            return
        }

        // Configure the workout session.
        let workoutConfiguration = HKWorkoutConfiguration()
        workoutConfiguration.activityType =
.functionalStrengthTraining
        workoutConfiguration.locationType = .indoor

        do {
            session = try HKWorkoutSession(configuration:
workoutConfiguration)
        } catch {
            fatalError("Unable to create the workout session!")
```

```
        }

        // Start the workout session and device motion updates.
        healthStore.start(session!)
        self.exercise.startDate = Date()
        motionManager.startUpdates(exercise:self.exercise)
    }

    func workoutCompleted(){
        print("SOUND PLAYED")

WKInterfaceDevice.current().play(WKHapticType.notification)
        stopWorkout()
        if let controller = delegate as?
LTExerciseControllerProtocol{
            controller.updateTitleLabel(text: "Workout stopped")
        }

        sendExerciseData()
    }

    func stopWorkout() {
        // If we have already stopped the workout, then do
nothing.
        if (session == nil) {
            return
        }
        self.exercise.endDate = Date()
        // Stop the device motion updates and workout session.
        motionManager.stopUpdates()
        healthStore.end(session!)

        // Clear the workout session.
        session = nil
    }

    func sendExerciseData(){

        if WCSession.isSupported() {
            let watchDelegate = WKExtension.shared().delegate
as? ExtensionDelegate

            guard let wcSession = watchDelegate?.wcSession else{
                return
            }
            let name = exercise.displayName
            let type = exercise.type.rawValue
            let countGoal = NSNumber(integerLiteral:
exercise.countGoal)

            let dayHourMinuteSecond =
Set<Calendar.Component>([.day, .hour, .minute, .second])
            let difference =
NSCalendar.current.dateComponents(dayHourMinuteSecond, from:
exercise.startDate!, to: exercise.endDate!)
            //in seconds
            var time: Int = 0
```

```
            if let second = difference.second {
                time = time + second
            }
            if let minute = difference.minute {
                time = time + minute
            }
            if let hour = difference.hour {
                time = time + hour
            }

            let timeObj = NSNumber(integerLiteral: time)

            wcSession.sendMessage(["displayName": name,"type":
type, "countGoal": countGoal, "time" : timeObj], replyHandler: {
(response) -> Void in
                print("MESSAGE SENT")
            }, errorHandler: { (error) -> Void in
                // 6
                print(error)
            })
        }
    }


    // MARK: MotionManagerDelegate

    func didUpdateCount(count: Int) {
        delegate?.didUpdateCount(manager: self, count: count)
    }
}
```

- LTExerciseInterfaceController.swift -

```
//
//  LTExerciseInterfaceController.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 4/2/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import UIKit
import WatchKit

protocol LTExerciseControllerProtocol: class {
    func updateTitleLabel(text: String)
}

class LTExerciseInterfaceController: WKInterfaceController,
LTWorkoutManagerDelegate {

    var workoutManager: LTWorkoutManager!
    var active = false
    var exerciseCount = 0
    var exercise: LTExercise!

    @IBOutlet var countLbl: WKInterfaceLabel!
    @IBOutlet var titleLbl: WKInterfaceLabel!
    @IBOutlet var doneImg: WKInterfaceImage!
```

```
    override func awake(withContext context: Any?) {
        if let exercise = context as? LTExercise { self.exercise
= exercise }
        workoutManager = LTWorkoutManager(exercise:
self.exercise)
        workoutManager.delegate = self
    }

    // MARK: WKInterfaceController

    override func willActivate() {
        super.willActivate()
        active = true

        // On re-activation, update with the cached values.
        updateLabels()
    }

    override func didDeactivate() {
        super.didDeactivate()
        active = false
    }

    // MARK: WorkoutManagerDelegate

    func didUpdateCount(manager: LTWorkoutManager, count: Int) {
        /// Serialize the property access and UI updates on the
main queue.
        DispatchQueue.main.async {
            self.exerciseCount = count
            self.updateLabels()
        }
    }

    // MARK: Convenience

    func updateLabels() {
        if active {
            countLbl.setText("\(exerciseCount)")
            if exerciseCount >= exercise.countGoal{
                doneImg.setHidden(false)
            }
        }
    }

    func updateTitleLabel(text: String){
        titleLbl.setText(text)
    }

    @IBAction func startExercise() {
        updateTitleLabel(text: "Workout started")
        workoutManager.startWorkout()
    }
    @IBAction func stopExercise() {
        updateTitleLabel(text: "Workout stopped")
        workoutManager.stopWorkout()
```

```
    }
}
```

- LTInitialInterfaceController.swift -

```
//
//  LTInitialInterfaceController.swift
//  LifeTracker
//
//  Created by Sergey Spivakov on 3/23/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//

import UIKit
import WatchKit

class LTInitialInterfaceController: WKInterfaceController{

    var exercises: [LTExercise]!

    @IBOutlet var tableView: WKInterfaceTable!

    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        exercises =  [LTExercise]()
        exercises.append(LTExercise(displayName: "Biceps", type:
.biceps, countGoal: nil))
        exercises.append(LTExercise(displayName: "Hammer
Biceps", type: .hammerBiceps, countGoal: nil))

        tableView.setNumberOfRows(exercises.count, withRowType:
"WorkoutRow")

        for index in 0..<tableView.numberOfRows {
            if let controller = tableView.rowController(at:
index) as? LTWorkoutRowController {
                controller.exercise = exercises[index]
            }
        }
    }

    override func table(_ table: WKInterfaceTable,
didSelectRowAt rowIndex: Int) {
        let exercise = exercises[rowIndex]
        presentController(withName: "LTExercise", context:
exercise)
    }

}
```

- ExtensionDelegate.swift -

```
//
//  ExtensionDelegate.swift
//  LifeTrackerWatches Extension
//
//  Created by Sergey Spivakov on 3/23/17.
//  Copyright © 2017 Sergey Spivakov. All rights reserved.
//
```

```
import WatchKit
import UserNotifications
import WatchConnectivity

class ExtensionDelegate: NSObject, WKExtensionDelegate,
UNUserNotificationCenterDelegate{

    var wcSession: WCSession? {
        didSet {
            if let wcSession = wcSession {
                wcSession.delegate = self
                wcSession.activate()
            }
        }
    }

    func applicationDidFinishLaunching() {
        if WCSession.isSupported() {
            wcSession = WCSession.default()
        }
        // Perform any final initialization of your application.
    }

    func applicationDidBecomeActive() {
        // Restart any tasks that were paused (or not yet
started) while the application was inactive. If the application
was previously in the background, optionally refresh the user
interface.
    }

    func applicationWillResignActive() {
        // Sent when the application is about to move from
active to inactive state. This can occur for certain types of
temporary interruptions (such as an incoming phone call or SMS
message) or when the user quits the application and it begins
the transition to the background state.
        // Use this method to pause ongoing tasks, disable
timers, etc.
    }

    func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>)
{
        // Sent when the system needs to launch the application
in the background to process tasks. Tasks arrive in a set, so
loop through and process each one.
        for task in backgroundTasks {
            // Use a switch statement to check the task type
            switch task {
            case let backgroundTask as
WKApplicationRefreshBackgroundTask:
                // Be sure to complete the background task once
you're done.
                backgroundTask.setTaskCompleted()
            case let snapshotTask as
WKSnapshotRefreshBackgroundTask:
                // Snapshot tasks have a unique completion call,
```

```
make sure to set your expiration date

snapshotTask.setTaskCompleted(restoredDefaultState: true,
estimatedSnapshotExpiration: Date.distantFuture, userInfo: nil)
            case let connectivityTask as
WKWatchConnectivityRefreshBackgroundTask:
                // Be sure to complete the connectivity task
once you're done.
                connectivityTask.setTaskCompleted()
            case let urlSessionTask as
WKURLSessionRefreshBackgroundTask:
                // Be sure to complete the URL session task once
you're done.
                urlSessionTask.setTaskCompleted()
            default:
                // make sure to complete unhandled task types
                task.setTaskCompleted()
            }
        }
    }

    func userNotificationCenter(_ center:
UNUserNotificationCenter, didReceive response:
UNNotificationResponse, withCompletionHandler completionHandler:
@escaping () -> Void) {

    }

    func userNotificationCenter(_ center:
UNUserNotificationCenter, willPresent notification:
UNNotification, withCompletionHandler completionHandler:
@escaping (UNNotificationPresentationOptions) -> Void) {

    }

}

extension ExtensionDelegate: WCSessionDelegate{
    func session(_ session: WCSession, didReceiveMessage
message: [String : Any]) {

    }

    func session(_ session: WCSession, activationDidCompleteWith
activationState: WCSessionActivationState, error: Error?) {
        print("WCSession is activated")
    }
}
```