OULU UNIVERSITY OF
APPLIED SCIENCES

Triet Le

iOS Development with Swift programming language

Creating iOS application for improving personal physical abilities

iOS Development with Swift programming language

Creating iOS application for improving personal physical abilities

Triet Le
Thesis
Autumn 2016
Business Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Business Information Technology

Author(s): Triet Le
Title of Bachelor´s thesis: iOS Development with Swift programming language
Supervisor(s): Jouni Juntunen
Term and year of completion: Spring 2017 Number of pages: 36

Since 2007, Apple introduced various models of handheld devices including iPhone, iPad, iPod and Apple Watch which provide a potential market for applications. As a result, iOS development becomes a significant topic for developers around the world to explore and study in detail. Consequently, this thesis also focuses on how to create an iOS application from scratch by introduce a simple health improving application called Handy Trainer.

The goal of the research is to learn how to develop an iOS application from scratch within supportive environment and Swift programming language from Apple. People are working extremely hard for getting a better life. However, because of a long-hour working day, physical health is reducing rapidly. Moreover, there are too many fault information on the internet that can lead to more serious health issues, including injuries, stress and muscle tightness. Handy trainer is a mobile application that can help users to calculate the right number of calories in take a day and recommend the right plan for long-term followers, including macro nutrition plan and work-out program. Users can access to the system anytime and anywhere with just few clicks due to its convenience as well as the use of latest software and tools. The main language for this application is Swift 3.0, which is a programming language designed by Apple. Besides, several background databases for all recommendations will be used. The golden goal of this application is to provide the most accurate calculation in BMR as well as suggest suitable nutrition and work-out plans for different individuals based on their own body measurements.

In the future, Handy trainer can be upgraded by adding more add-on functions as well as improving the databases for more accurate results. Moreover, sending notifications and reminders to users will be the main goal for further development.

Keywords: Trainer, iOS application, health, calories, meal plan, nutrition.

CONTENTS

1 INTRODUCTION 5
2 IOS APPLICATION DEVELOPMENT WITH SWIFT 3.0 7
2.1 iOS application architecture 7
2.2 XCode – Apple native integrated development environment 8
2.3 Creating user interface 9
2.4 Swift programming language 11
2.5 Advantages of Swift programming language 12
2.6 MVC pattern for iOS development 13
3 ENERGY CALORIES BASAL METABOLIC RATE (BMR) 15
3.1 BMR calculation 15
3.2 Cardio for fat loss 16
3.3 BMR calculation 16
4 APPLICATION DEVELOPMENT AND RESULT 20
4.1 User Interface Design 21
4.2 Execution 24
4.2.1 Exception handling 24
4.2.2 Finalizing application 26
5 DISCUSSION 31
REFERENCES 33
APPENDIX 1 – COLLECTING DATA SURVEY FORM 36

1 INTRODUCTION

Apple Computer Inc. introduced their first iPhone on 2007, which created a big revolution for Smartphone industry with a big touchscreen and one home button for navigation. iPhone is the first user-friendly smartphone designed to gives their users a comfortable feeling when surfing web content, exploring applications with simple touches as well as various standard mobile phone features. Besides, iPhone allows users to touch the contents while, at the time, there are not many companies can. According to Goadrich and Rogers (2011), iOS development is getting more and more popular over the time, which aims to create applications selling on App Store and can be applicable for iPhone or iPad's users. In the first three months of 2017, Apple is having 42.7 percent of users from all the gadgets and mobile market while Android shares more than 50 percent and other mobile companies' shares are just over 2 percent of the market (Wee, 2017). Moreover, in 2017, the number of downloads through Apple store reaches 25 billion which is unreal and a milestone for company (Apple 2015, cited 22.02.2017).

Therefore, the aim of this thesis is to describe how to develop an iOS application, named Handy trainer, from scratch. This application is going through three steps that include the design, the development and the testing phase. The application's interface is designed using Stack View method which is an automated tool to design user interface. In the other hand, MVC pattern will be used to design the skeleton of application while navigation controller is the main controller for navigating between screens. The development will apply XCode 8 which is an Integrated Development Environment (IDE), where developers can create, test and debug apps using Swift 3.0 as the main programing language. At last, the testing phase will be accomplished manually to make sure that the application can work properly and smoothly. Practicing the knowledge from the research, a health iOS application will be developed. Besides, simple calculating formula will be integrated into the application to practice the implementing skills as well as testing skills. The calculating formula will be chosen by study three different kinds which are Mifflin's equation, Basmore's equation and the rule of ten.

The application, Handy Trainer, is expected to have some simple yet practical features such as BMR calculation and recommendations for training programs, such as High-intensity interval training and Low-intensity steady state training, that are suitable for different users by entering their height, weight, gender and age. By developing these app's features, there are various

aspects of Swift can be explored. The process of creating the user interface for Handy Trainer also will improve experience and knowledge of multiple user interfaces' objects such as static and dynamic texts, labels and text field for inputting data. Besides, by applying the real calculating equation for BMR, knowledge of controlling and limiting errors is gained. However, because of the research scope, there are not so many features can provide from the application. Also, the testing part is done manually so there might be few unexpected cases that cannot be controlled properly (appendix 1).

Besides, there are still some limitations from the application. Firstly, the application will return wrong result if user input extremely large data like one million for the height, the weight or the age. Secondly, some advanced features such as notifications, reminder and multiple recommended training programs are not included due to the limited knowledge. Lastly, the application will only be tested by installing archived version while there is no certified developer account from Apple.

2 IOS APPLICATION DEVELOPMENT WITH SWIFT 3.0

2.1 iOS application architecture

iPhone operating system, best known for iOS, was introduced alongside with iPhone by Apple Inc. on 2007 (Gonzalez-Sanchez and Chavez-Echeagaray, 2010). iOS is exclusively designed and distributed by Apple and pplication is developed within supported environment. The environment, in this case, is not only considered as a place to develop application but also the whole ecosystem, that will start from the lowest level of knowledge called Cocoa touch layer. In order to utilize the best out of Cocoa touch layer, XCode native IDE from Apple and Swift programming language will be used accordingly.

Apple has continued to utilize iOS for iPod Touch, iPad, Apple TV and Apple Watch. Thus, iOS is continuously developed and enhanced. However, the most basic technology is kept, which includes three layers of system: Cocoa Touch layer, Media layer and Core layer. The Core layer has two sub layer called Core Services layer and Core OS layer (Apple 2014, cited 22.02.2017).

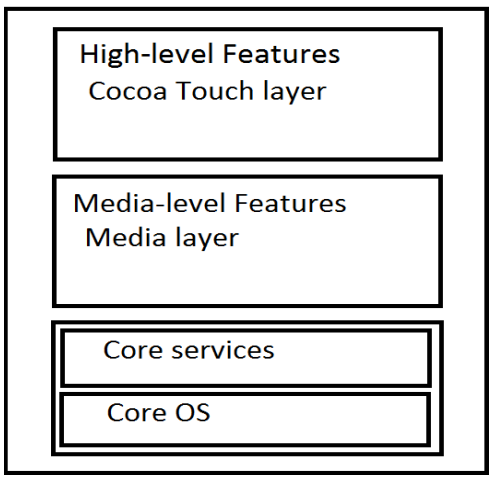

FIGURE 1. iOS Technology (Apple 2014, cited 22.02.2017)

The Cocoa touch layer (figure 1) can be considered as the user interface and user experience. It provides huge variety of components and design methodologies from Apple, including multitasking, touch-based input and output, notifications, and high-level system interactions. Therefore, Cocoa touch layer is the most fundamental layer of iOS application (Apple 2014, cited 22.02.2017).

Media layer provides various multimedia functionalities, including video, auto and graphics. Graphics technologies content numerous embedded libraries which enhances iOS applications' high-quality user interface and easy to customize due to its flexibility. Audio technologies, on the other hand, come with hearing experiences. Audio technologies support rich audio playback, audio recording, HDMI content and built-in sounds. Finally, video technologies utilize the power of iPhone to manage video contents such as playing and recording. Besides, with appropriate recording hardware, video technologies can also provide the ability for streaming video contents through internet.

As mentioned earlier, the Core layer has two sub layer called Core Services layer and Core OS layer. Core services layer is the layer for adding peer-to-peer services, such as iCloud, social media sharing and networking. iCloud, or iCloud storage, is the Apple's online data storage. iCloud service provides functionalities for writing user documentations and information to a central location. iCloud storage is protected by Apple at an extremely high-level of security so that users' data will not be stolen or leaked. On the other side, Core OS layer is the low-level layer which provides back-end processing functionalities. Core OS provides frameworks which support upper level layers with security and management. Managing functionalities are, mainly, local authentication, core Bluetooth, external assessor and network extension.

2.2 XCode – Apple native integrated development environment

XCode is an integrated development environment introduced by Apple in 2003. The latest and most stable version of XCode is version 8 (figure 2). XCode comes with modern yet powerful features which provide a full package for developers. XCode gives developers a supported UI designing, implementing and testing. The most powerful feature of XCode is runtime, which continuously tracks and alert developers about syntax bugs, designing recommendations and memory management. Additionally, with the latest version, XCode provides large amount of extensions which can be smoothly integrated into the application (Apple 2015, cited 22.02.2017).

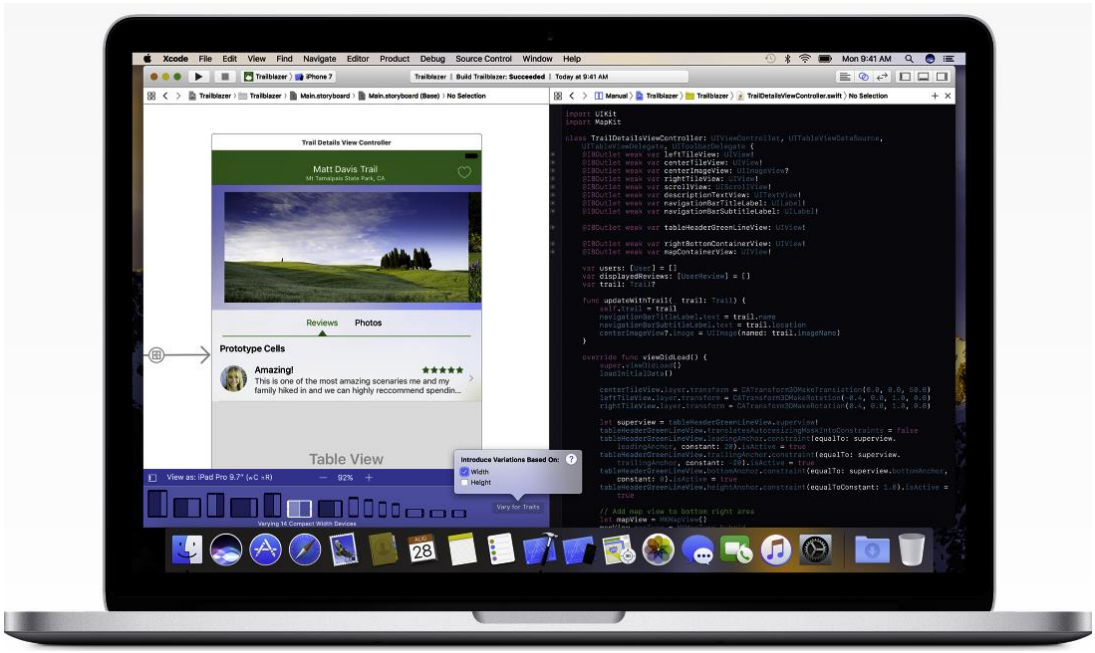

FIGURE 2. XCode GUI

XCode supports various types of programming languages and script writing including C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit and Swift. Despite of huge amount of programming languages supported, Apple decided to make Swift as the main programming langue for XCode (Apple 2017, cited 22.02.2017).

Interface builder is an editor within XCode. Interface builder supports drag and drop feature without coding which gives developers easier tasks for designing the user interface. Because of the Cocoa and Cocoa touch objects, developers can design user interface separately from implementation. Moreover, the connection between UI and code will be handled by integrated runtime. The connection, which can be considered as a label, allows developers to manipulate the UI components with controller. The runtime will notice developers instantly when the object is placed wrong or the conflict between object and logic behind.

2.3 Creating user interface

Apple Inc. (2017) stated that a complete iOS application is the combination between multiple views and connections. Therefore, Apple introduced StoryBoard, which can be considered as a designer to handle and connect all features as well as elements that developers implement,

including Table view controller, Collection view controller, Navigation controller, Tab bar controller, Page view controller and GLKit view controller. The table view controller provides developer ability to establish an excel-like table to display multiple contents, such as images, numbers or text. In the other hand, Collection view controller gives developer a powerful tool to manipulate the displaying style of user interface. The user interface can be arranged as a grid view, row view or album-like view. However, without the help of Navigation, Tab bar and Page view controllers, Collection view and Table view cannot provide the best experience for users. Navigation view controller is an embedded controller provided by Apple which gives developer less work to establish navigation between multiple application's screens. Besides, Tab bar controller adds more functionalities for Navigation controller to instantly navigate between specific screens. Even thought, Apple introduced a powerful controller which has both the power of Tab bar and Navigation controller, the Page view controller. Therefore, developers have more choices as well as freedom to decide between multiple supporting features (Apple 2014, cited 22.02.2017).

Beside all supported controllers, Auto Layout (figure 3) and Stack View is another built-in functionality that contributes to XCode powerful feature. Auto layout automatically arranges and defines constraints for each object in the user interface design. Thanks to this powerful feature, all components can be controlled automatically and precisely for different devices (Apple 2014, cited 22.02.2017).

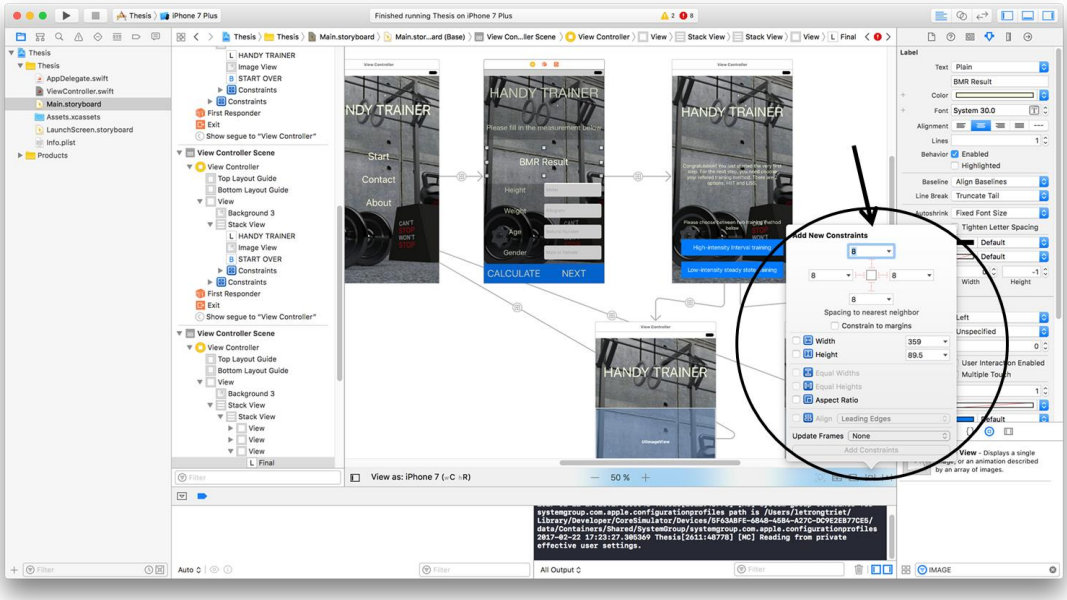

FIGURE 3. Auto layout sample

2.4 Swift programming language

Swift is a programming language created by Apple and was developed from objective-c. However, Swift is more modern and user-friendly language that is designed for users and by users. It has both the power of low-level programming languages like C or C++ and the smoothness of high-level languages like C# or JavaScript. Moreover, Swift is light and packs with pre-defined library. "The goal of the Swift project is to create the best available language for users ranging from systems programming, to mobile and desktop apps, scaling up to cloud services.", stated Apple Inc (2017).

Swift has several features and characteristics which provide the best environment for developers. The main three key elements that help Swift stands out from the crowd are safe, fast and expressive. Runtime, as mentioned above, is the built-in debugger, which can track and alert developers spontaneously. Thanks to Runtime, Swift can provide the safe and user-friendly environment for developers. Besides, Swift is developed from C-based languages, such as C, C++ and Objective-C so it inherences all the best and improvement from C-base family. Thus, Swift is predicable, consistent and fast as well as expressive since it is the modern programming language which has been developing for years (Apple 2015, cited 22.02.2017).

Some other features that can be seen as an advantage of Swift are worry-free syntax, inferred style variable and nested comment style. In term of worry-free syntax, developers do not need to remember the semicolon at the end of every single code line as in C++, Python or Objective-C, which can give developers more time and confidence when developing an application (figure 4).

```
var nums = [1,2,3,5,7,11]
for n in nums
{
  println(n)
}
```

FIGURE 4. Modern FOR loop

```
var msg = "A String in Swift"
var s = msg.utf16Count // s is 17

auto msg = "A zero-terminated const char[] in C++";
auto s = msg.length(); // error no such member

auto msg1 = std::string("A string in C++");
auto s = msg.length(); // s is 15
```

FIGURE 5. Auto STRING creation

Besides, regarding inferred style variable, Swift can provide inferred string type variable while in C++, the auto feature cannot provide auto string without declaring std::string explicitly (figure 5). Lastly, concerning the comment style, Swift has single-line and multiple-line comments which is similar to old and modern programming languages. However, it has one stand out feature that is nested comment (figure 6).

```
// C++ single line comment
// Swift single line comment

/* C++ Multiline
comment but cannot nest */

/* Swift Multiline
comment but /*can*/ nest */
```

FIGURE 6. Nested Comment

2.5 Advantages of Swift programming language

In 1980s, Objective-C was developed base on C programming language. Objective-C focuses mainly in object-oriented programming style which is the reason why Apple used it to develop iOS as well as MacOS applications. However, since 2014, Apple decided that Swift is the main programming language. There are various reasons why developers and Apple prefer Swift rather than Objective-C. Swift is developed by improving and revising the old and outdated Objective-C. Thus, Swift is more natural and user-friendly than Objective-C. Swift has the compiler named LLVM which reduces the files needed to generate a completed product without losing the flexibility. As mentioned in previous section, Swift has a runtime debugger which can work spontaneously with developers to track and alert all errors. As a result, Swift is easy to read, easy to maintain and user-friendly (Hillyer 2016, cited 22.02.2017)

Due to less amount of code and files, Swift can compile and build the program tremendously faster than Objective-C. Swift provides a side feature called Playground. Playground can be considered as a simulator which provides a test environment for developers to develop single module without building the entire program. Swift is developed within huge number of rebuilt libraries and frameworks which can provide supportive programming environment for developers (Solt 2015, cited 22.02.2017).

2.6 MVC pattern for iOS development

The Model-View-Controller, best known for MVC (figure 7), design pattern is the fundamental for object-oriented programming. The MVC model is the combination of three crucial components, including *model, view and controller* (Apple 2015, cited 22.02.2017).

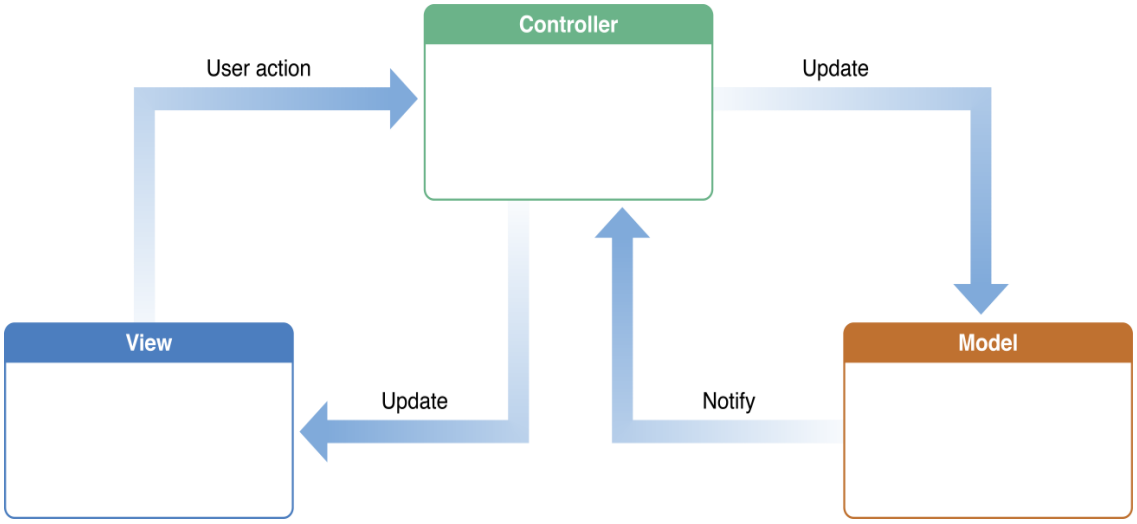

FIGURE 7. MVC design pattern (Apple 2014, cited 22.02.2017)

The *model* component can be considered as a database for application. *Model* is used to store the data as well as the logic provided by developer to control and process data storage. In Swift, *model* acts like a slave while the master acts like a controller. The *model* will process data and give response whenever there is a command from controller. *Model* can be a small built-in database from XCode or an outside database connected by controllers.

The *view* component refers to all the visual objects that contribute to user interface. Normally, *view* is used to display the data received from *model* and sends notifications regarding users' actions. *View* acts like a slave from controller and a reflection from *model*.

The *controller* is the master of application. *Controller* acts as an intermediary between *view* and *model*. *Controller* will receive the notifications and input from *view*, analyze these notifications and transfer the command to *model* for processing. After receiving the response from *model*, *controller* will process the feedback and transfer to *view* as an update for specific objects. *Controller* can be considered as a waiter or waitress in a restaurant where *model* is the chef and *view* is the customer.

At a glance, Swift's MVC (figure 8) pattern is the normal MVC pattern being used by many programming languages. However, the MVC pattern in iOS development has own strengths, such as dependency of *view* component or *controller* component. In iOS development, the view will, only, be controlled if developers decide to create a connection between view and controller. However, with the help of multiple controller classes provided by Apple, MVC design pattern in iOS development is well supported. The *view* component can be designed separately without writing the code or XML script. Besides, the MVC pattern from iOS development is adopting the flexibility of MVVM (Model-View-View-Model) to enhance its features by distribute equally tasks between *model* and *controller*.

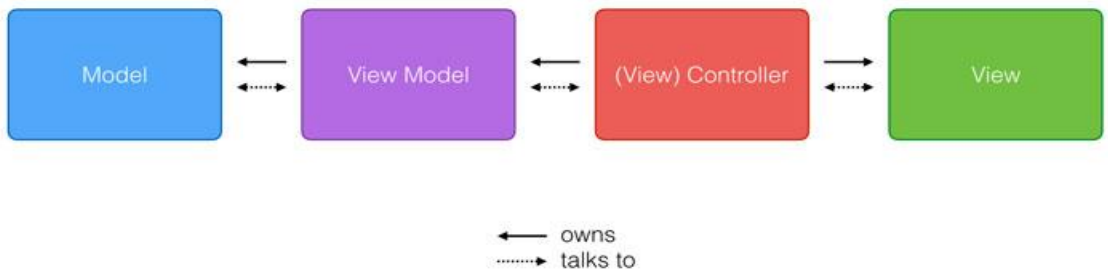

FIGURE 8 iOS specific MVC pattern

3 ENERGY CALORIES BASAL METABOLIC RATE (BMR)

3.1 BMR calculation

Handy trainer is a mobile application that plays role as a personal assistant to help users improve their own physical health. The most important attribution of Handy trainer is energy. Hall et al. (2012) stated that human body produce energy by convert food or drink in the digestion system. Energy can be measured as calories (cal) or kilojoules (kj).

Luke and Schoeller (1992) mentioned that Energy Calories Basal Metabolic Rate (BMR) is one of the most effective models to measure the energy level that the human body should maintain during the day. According to Black (2000), in everyday activities, human body needs a specific amount of energy for living, physical or mental activities, called basal metabolic rate (BMR). By calculating the BMR, users can determine a specific plan to achieve their goals, which mainly focus on losing weight, gaining weight or maintaining weight. Also, Luke and Schoeller (1992) believed that BMR can be affected by several factors such as training frequency, body type, body composition, etc. Following above theoretical points, this thesis will collect data from users including age, gender, weight and height to take into account as well as calculate the most accurate BMR.

Black (2000) also introduced the concept of energy-in and energy-out while calculating the BMR. Base on different work-out goals, the energy-in is defined by users' nutrition plan, which includes how much calories per day their body should absorb. Handy trainer will adapt this concept to recommend suitable meal plan for different individual base on their specific objectives. Sacks et al. (2009) divided energy consumption into three main elements, which are carbohydrates (carb), proteins (pro) and fats (f). Carbohydrate can be found in starchy food like rice, noodle, potato or similar products. Protein can be found in animal products including meat, eggs or milk. Regarding vegan users, protein can also be taken from protein-rich vegetables such as bean, pea or lentil. Fat can be found in almost all food. For instance, vegetables, nuts, milk or animal meat. On the other side, energy-out is estimated by users' physical activities plan. Sacks et al. (2009) stated that to achieve users' specific goal, the energy consumption will only take 70 percent, another 30 percent will be the work-out plan.

3.2 Cardio for fat loss

According to Schoenfeld (2011), Cardio or cardio workouts are physical activities which involve both fast or slow pace movements. Cardio can be walking, running, jogging, swimming or cycling. Cardio plan will be suitable for users whose goal is losing weight. Besides, cardio workouts will improve users' health, especially heart and lungs.

Laursen (2002) defined high-intensity interval training (HIIT) as a training method which requires quick and intense physical activities within short period of resting. Therefore, with HIIT, the users' heart rate as well as metabolism will be increased, which fastens the process of burning energy. According to Laforgia et al. (2006), the purpose of HIIT is to give users Excess post-exercise oxygen consumption (EPOC), which refers to the state when body requires more oxygen during recovery after HIIT workout. EPOC gives users the ability to burn more calories and fat than other normal physical activities. Thanks to the short period of resting combined with intense workout, HIIT also increase the endurance of users. Since HIIT focuses on high intensity workout in short period of time within brief resting time, it provides quick and convenient sets of exercises for users to follow. It is usually a no-equipment workout program and requires, almost, all body-weight exercises (Milanović et al., 2015).

According to Klein et al. (1994), low-intensity steady state, known as LISS, is a training approach, which requires slow to medium pace exercises within fixed period. LISS can be considered as light cardio due to its type of exercises. LISS provides exercises which do not require endurance or advanced knowledge to be accomplished. Comparing to HIIT, LISS requires people with time and calmness since it is time-consuming and reasonably slow. Also, it requires no equipment to be accomplished and similar to HIIT, LISS fastens the process of burning calories and fat.

3.3 BMR calculation

Following previous researches about BMR Model, there will be three main formulas for testing and experiencing in this phase. The formula is firstly developed by Mifflin et al. (1990). In this equation, gender will be the key elements for the outcomes:

- Male: BMR = 10 * weight(kg) + 6.25 * height(cm) - 5 * age(y) + 5
- Female: BMR = 10 * weight(kg) + 6.25 * height(cm) - 5 * age(y) – 161

Basically, these two kinds of equation require the same amount of information for calculating, which are weight, height and the age of users. As we can see from the above equation, the calories needed for women and men are different base on the bone structure as well as body fat distribution. Since female body will need more fat for essential parts, they tend to store less muscle. On the other hand, male body does not need a lot of essential fat; thus, they will need more muscle for heavy and active movements. Also, as can be seen from these equations, age is considered a critical element which can define the calories per day. When people get older, their body will store more fat rather than reserve muscle and muscle always burn more calories than fat. Therefore, the age must be included in order to archive the most accurate results.

There are two main systems for measurement which are metric and America. In Metric system, the height will be measured in meters, centimeters or millimeters. Weight will be measured in grams or kilograms. On the other side, in America system, weight will be in pounds and height will be in feet and inches. Following both kinds of systems, the main and only input and output unit will be integer, which is a whole number and does not content fraction or floating point. The age, weight and height, for calculating BMR, must be converted into integer for inputting. To ensure all input numbers are integer, exception handling will be done during the input session from users. As a result, the output BMR should also be introduced as a whole number, which is in fact quite easy and comfortable for users to remember by heart.

Duyff (2012) invented the rule of 10, which estimate the number of calories an adult need. This is the easiest formula among these three ones. However, since this equation only needs one element for calculation, which is bodyweight (in US system, pound), the outcome will be considered only as a reference and not so accurate. In this context, the simple equation for calculating BMR is:

- BMR= 10 calories/pound of bodyweight for women
- BMR= 11 calories/pound of bodyweight for men

Similarly, all the input and output for this equation will be integer for conveniences and time-saving. However, the equation only takes input as pound, in US system. Therefore, for users' convenience, we will also convert the input into Metric system as follow:

- BMR= 22 calories/kilogram of bodyweight for women
- BMR= 24 calories/kilogram of bodyweight for men

Similar to the Mifflin – St Jeor equation, bias checking and negative numbers will be checked during the input session from users.

Last but not least, the Sterling and Pasmore equation is adapted from Lowe (2006), which can be considered as the most complex yet accurate equation of all three. The equation requires the body lean mass to calculate BMR:

BMR= Body lean mass (lbs) x 13.8 calories

The equation can be used for both genders since there are no differences between male and female body lean mass. The body lean mass can be calculated based on the body fat using following calculation:

- Body fat % x weight= fat mass
- Weight - fat mass= body lean mass

However, only advanced or experienced users can use this equation since inexperienced users do not know how to calculate their own body fat. Therefore, the complexity of needed information will somehow narrow down the outcomes. This equation also uses US system to calculate the BMR; thus, automatic conversion will be needed to increase users' convenience.

After determining the final body lean mass, users will be able to calculate their basic BMR. However, this basic BMR is not the final result but one more factor will be used to estimate the final one, which is the frequency of physical activities of different users. There will be five main categories as follow:

- BMR x 1.2 for low intensity activities and leisure activities (primarily sedentary)
- BMR x 1.375 for light exercise (leisurely walking for 30-50 minutes 3-4 days/week, golfing, house chores)
- BMR x 1.55 for moderate exercise 3-5 days per week (60-70% MHR for 30-60 minutes/session)
- BMR x 1.725 for active individuals (exercising 6-7 days/week at moderate to high intensity (70-85% MHR) for 45-60 minutes/session)

- BMR x 1.9 for the extremely active individuals (engaged in heavy/intense exercise like heavy manual labor, heavy lifting, endurance athletes, and competitive team sports athletes 6-7 days/week for 90 + minutes/session)

Since the ration for calculating final BMR is diverse, all final BMR will be rounded up to the normal number or integer number.

4 APPLICATION DEVELOPMENT AND RESULT

Applying the MVC model, Handy Trainer has the user interface as the *view*, users' input data will be processes by *controller* and the *model* is the software logic. The *view* will content four pages, including welcome page, calculation page, result page and the recommended programs page. Regarding *model*, Handy Trainer will use local variables for storing users' input which will be processed by *controller*. Lastly, the *controller* will content the processes which handle exceptions and calculate users' inputs. In this section, the MVC model will be explained in more detail.

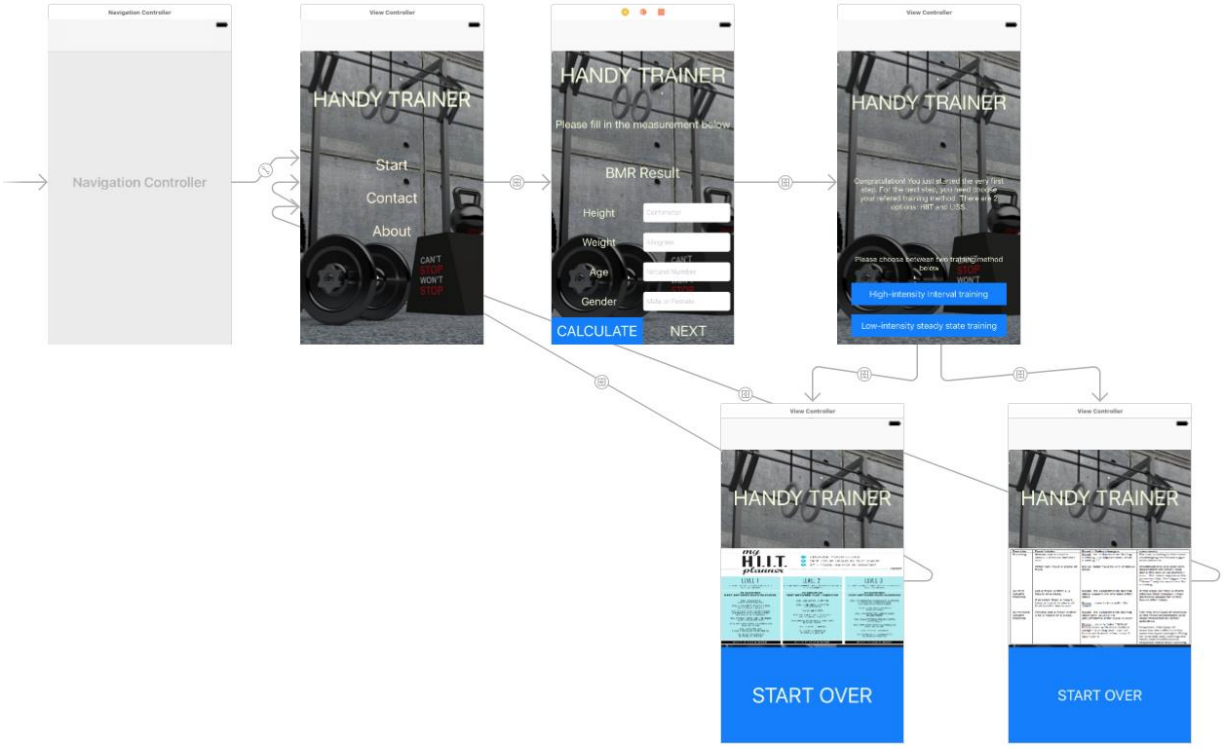

FIGURE 9. Five pages of Handy Trainer

In Handy Trainer, the design will be divided into five distinct user interfaces called pages (figure 9). The first page of design is the welcome page. In this page, users can look at some brief description about Handy Trainer application and the developer's contact information. There is a button which redirects users to the next page, which is the calculating page. In this page, users can input their personal measurements which are the key elements to calculate the BMR. After using the Calculate button, the label "BMR Result" will be replaced by the result. After calculating, users can go to recommended program page by pressing the button "Next". In this page, users

will be informed with some static text in the text view object. Users can choose from the page their favorite programs to follow, including High-intensity Interval Training (HIIT) or Low-intensity Steady State Training (LISS). The last two pages have the same number of objects and structure but different functionalities. One page is used to display the HIIT training methods while the other page is used to display the LISS training method. In these two pages, there is a button named "START OVER" which will erase all input data and redirect users to the very beginning.

The outcome of this research meets the defined objective. Users can use the application within least amount of errors thanks to exception handling. Besides, users can navigate between pages with the help of embedded navigation controller. After testing with different equations, the Mifflin – St Jeor equation is chosen to be the most suitable one for calculation BMR due to its user-friendly support which is easy to use and requires simple input information. Moreover, suitable condition controlling allows users to go back and forth between pages smoothly. Combined all of these outcomes together, Handy Trainer (figure 10) is successfully established as a complete iOS application.

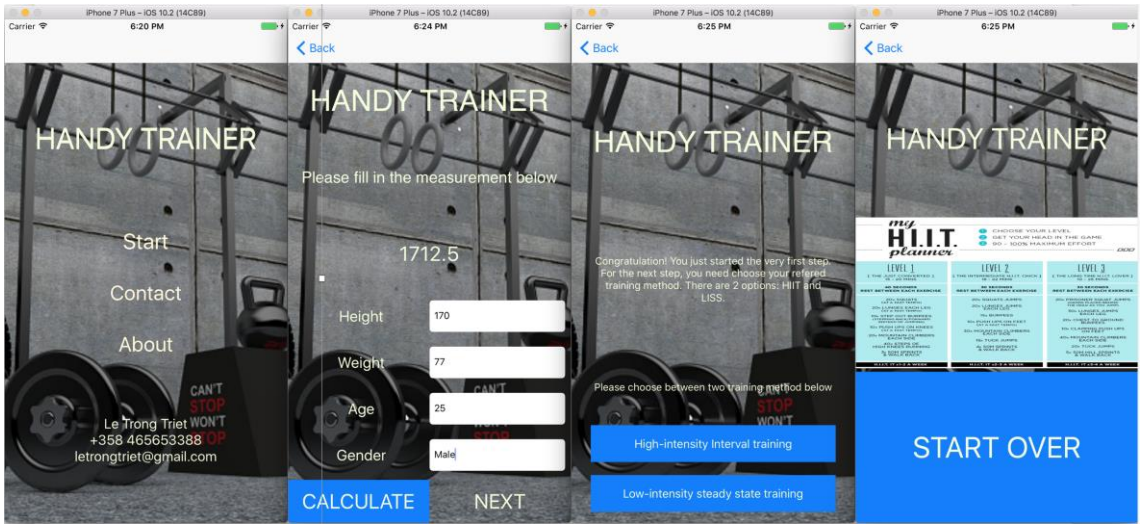

FIGURE 10. Completed version of Handy Trainer

4.1 User Interface Design

The first component of MVC design model is *view*, also known as user interface. User interface presents the needs of users. However, the design can be different from personal tastes and the

devices. Since 2007, Apple has introduced a huge range of products with different screen sizes and resolutions, such as iPhone with 4.7-inches screen or iPad with 2732 x 2048 resolutions. Therefore, the design of Handy Trainer will take these into account and adapt to all the variety of products from Apple. The method for designing Handy Trainer user interface is Auto Layout which is also one of several designing methods that is integrated into XCode (figure 11).

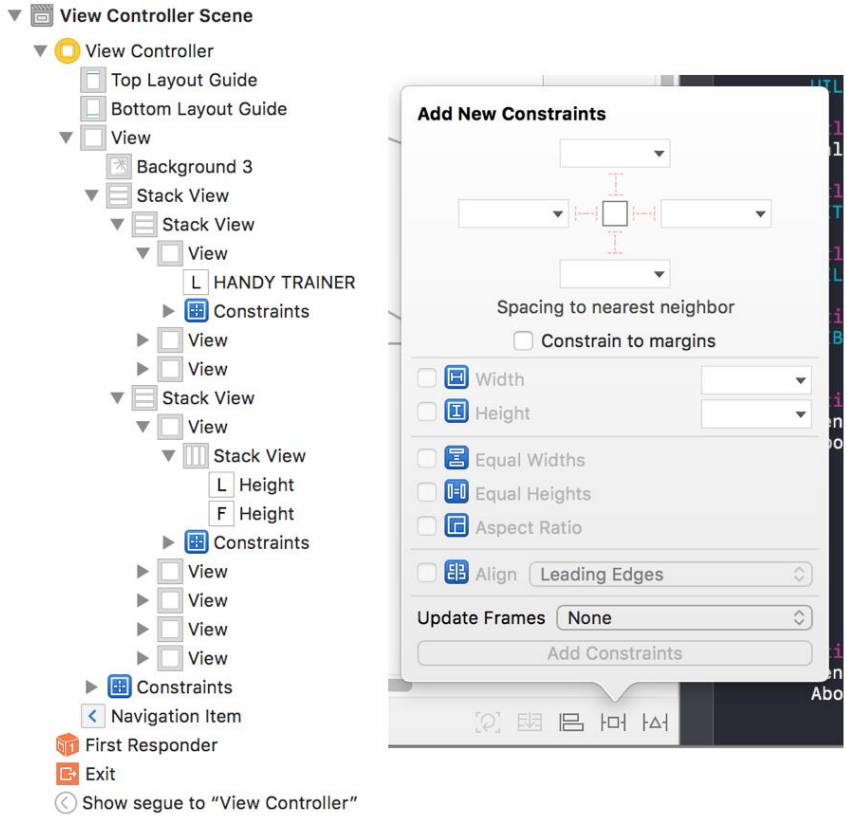

FIGURE 11. Auto Layout GUI

Auto layout is a combination of constrains and measurements which gives developers freedom to design the interface. On the surface, auto layout looks like other designing tools; however, the real power of auto layout is stack view. Stack view is a tool which combines several objects into one layer of design to manage. Stack view will expand or reduce the size of layers according to the models used. Stack view requires four constrains to perfect the design, including top, bottom, right and left. Following the above method, Handy Trainer will focus on simple and precise objects, such as label, text field, button or text view.

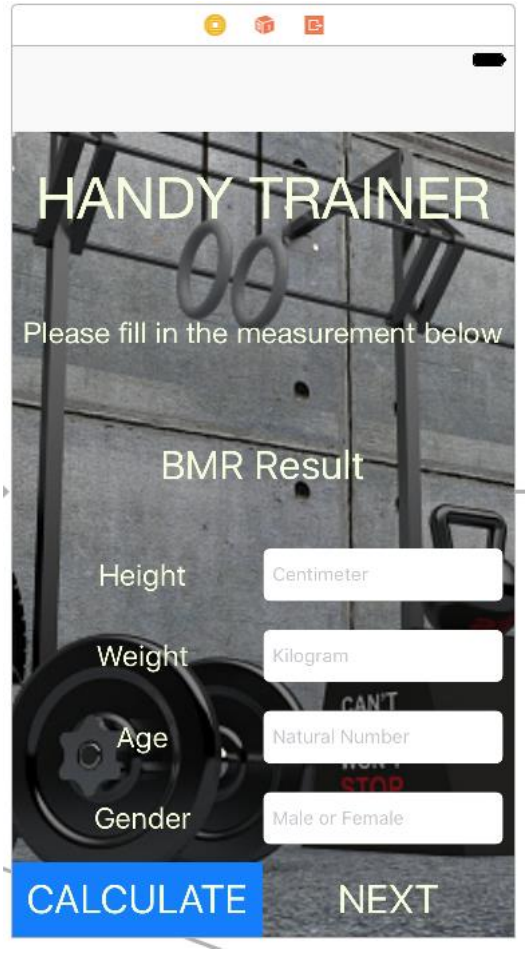

FIGURE 12. Handy Trainer User Interface

As we can see clearly in the figure above (figure 12), there are five objects in the view, including labels, Ulimage view, buttons, text view and text field. The label is used to display the name of application and some instruction for input data. The button is used for trigger functionalities and navigate between pages. Text view is used to display a short paragraph which cannot be displayed by label. Text field is reserved for users' input data. The Ulimage view is used to make up the background for the application. Beside these objects, there are some other adjustments and measurements which provide better editing for the interface.

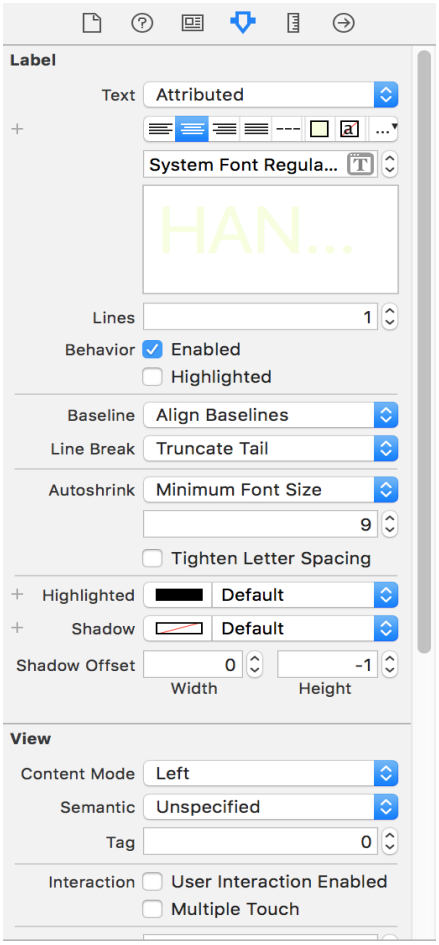

FIGURE 13. Attributes inspector

Attributes inspector (figure 13) is the most important part of any objects which provides various adjustments from fonts to alignments. Thanks to the help of attributes inspector, each user interface will have some standout criteria.

4.2 Execution

4.2.1 Exception handling

Exception handling refers to the process of errors checking. There are still few flaws that Swift may face up with. Therefore, exception handling is needed to ensure the proper functionalities of Handy Trainer. As discussed in the previous section, Handy Trainer will focus on input data from users to calculate the BMR result. Therefore, checking the users' input is the very first step for exception handling. In Handy Trainer, the data input from text field will be transfer into numbers; thus, empty input must be handled to ensure crash-free application. There are two possible cases

which can lead to an empty input, such as leaving empty or entering “space” character from keyboard.

```
if InputHeight != nil
{
    if InputWeight != nil
    {
        if InputAge != nil
        {
```

FIGURE 14. Checking for empty input

The figure above (figure 14) is the code to handle previous two cases which cause empty inputs. The code uses condition “IF” to compare the input with an empty string for checking empty input and a string with space character for checking the space entering. Ensuring the empty cases will not appear is the first important step before using Handy Trainer.

The next step is to handle the positive numbers input (figure 15). Generally, the weight, the height and the age of users must be positive numbers. Thus, the exception handling for this case is checking whether the input numbers are positive numbers or negative numbers. Negative number can be checked by providing an “IF” condition where numbers are small than zero.

```
if (InputWeight <= 0 || InputHeight <= 0 || InputAge <= 0)
{
    Final.text = "No negative input PLEASE"
}
```

FIGURE 15. Checking for negative input

Handy trainer will also need to check for the wrong inputs which are not numbers. There are many cases can appear while users inputting the data from keyboards, such as characters or special characters. For Handy Trainer, this case will be handled by casting string, input from users, to the type of number that is suitable for calculating.

```
var InputHeight: Float = 0.0
var InputWeight: Int = 0
var InputAge: Int = 0
var InputGender: String = ""

InputGender = Gender.text!
InputHeight = Float((Height.text)!)!
InputWeight = Int((Weight.text)!)!
InputAge = Int((UserAge.text)!)!
```

FIGURE 16. Checking for wrong input types

The general idea of above figure (figure 16) is to check the wrong input by using casting function from Swift. There are various cases and conditions for casting; however, Handy Trainer uses only two cases for casting which are cast string to integer and string to floating point. In this case, casting requires an absolute variable from casting to prevent the wrong input. As a result, wrong input will not be casted into the reserved variables and feedbacks will be sent to users by a dynamic text at the same screen.

4.2.2 Finalizing application

As mentioned in previous section, Handy Trainer uses built-in objects from XCode to create user interface. However, connections between objects and Controller need to be established. In Swift, there is a method called “Outlet” which helps developers to control all user interface objects with codes.

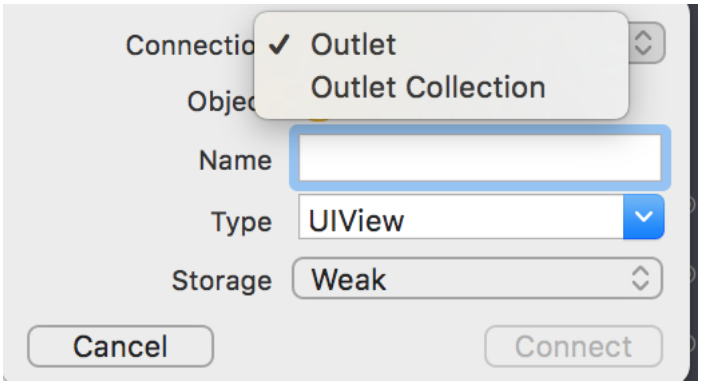

FIGURE 17. Outlet connection example

In the figure above (figure 17), there are five configurations which developers need to define when establishing an outlet for object. The first configuration is the connection in which developers can choose between “Outlet” or “Outlet Collection”. The only difference between

Outlet and Outlet Collection is the scale. The second configuration is the “Object”. Object is handled by Swift; thus, developers do not need to configure this one. The third configuration is “Type”. Developers can leave the configuration as defined by Swift or choose the suitable types for the connection. Type, normally, is the type of object needed to connect. The fourth configuration is “Name”, in which developers can assign a specific name for the object to control later. The last configuration is the “Storage” which defines how easy the object is changed.

```
@IBOutlet weak var Height: UITextField!
@IBOutlet weak var Weight: UITextField!
@IBOutlet weak var Age: UITextField!
@IBOutlet weak var Gender: UITextField!

@IBOutlet weak var UserAge: UITextField!

@IBOutlet weak var AboutContactButton: UITextView!

@IBOutlet weak var BMRResult: UILabel!

@IBOutlet weak var BMR: UILabel!

@IBOutlet weak var CalculateBMR: UITextView!

@IBOutlet weak var ShowBMR: UITextView!

@IBOutlet weak var Final: UILabel!

@IBAction func Next(_ sender: UIButton) {
}
```

FIGURE 18. Outlets for Handy Trainer

In the figure above (figure 18), Handy Trainer uses weak storage type for all objects instead of button functions. The weak storage type defines that the object or component can be modified more easily than the strong storage type. The main purpose is to make a change for objects during processing.

```
@IBAction func ContactButton(_ sender: UIButton) {
    AboutContactButton.text = "\n" + "\n" + "Le Trong Triet" +  "\n" + "+358 465653388" +  "\n" +
        "letrongtriet@gmail.com"

}

@IBAction func AboutButton(_ sender: UIButton) {
    AboutContactButton.text = "\n" + "\n" + "Improving Health with simple touch"
}
```

FIGURE 19. Button outlets

As can be seen from figure 19, the button outlets are considered as functions where processing codes are stored. A button outlet can act like a bridge between objects and commands or a trigger to navigate between multiple pages in application.

The next task to consider is creating navigation system (figure 20) which connects multiple pages. In Swift, there is an embedded navigation which allows developers to establish connections between pages without using code.

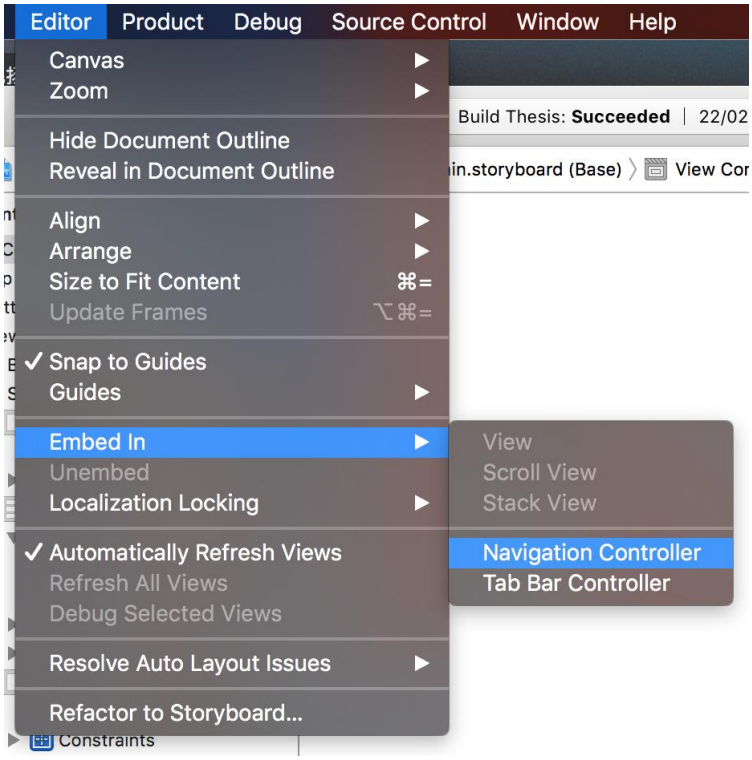

FIGURE 20. Create navigation system

Embedded navigation controller allows developers to go back and forth between pages by clicking the assigned button. In Handy Trainer, the first page is related to the second page by "Start" button (figure 21). The second page connects with the third page by the "Next" button. The third page connects with result pages by two buttons named "Low-intensity Interval Training" and "Low-intensity Steady Stage".

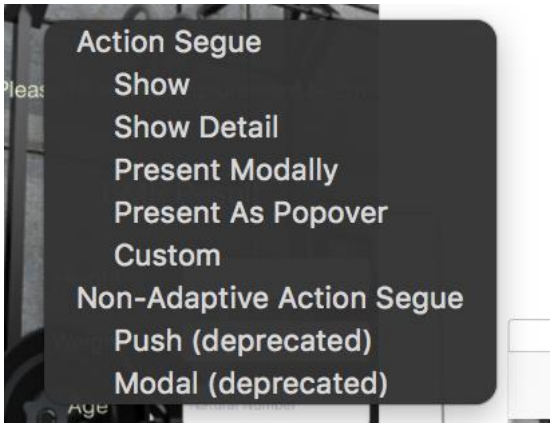

FIGURE 21. Establishing connection by button

Additionally, embedded navigation controller automatically creates the back button in the menu bar which allows users to navigate back to previous page. After creating all front-end components and connections, the process of calculating will be handled by code. The back-end process includes calculating the BMR and showing the result in user interface. This section will focus on calculating BMR. The BMR equation (figure 22), which is introduced in previous section, is calculated by taking input from users through text filed.

```
if InputHeight != nil
{
    if InputWeight != nil
    {
        if InputAge != nil
        {
            if (Gender.text != "" && (Gender.text == "Male" || Gender.text == "Female"))
            {
                if (InputWeight <= 0 || InputHeight <= 0 || InputAge <= 0)
                {
                    Final.text = "No negative input PLEASE"
                }
                else
                {
                    if InputGender == "Male"
                    {
                        var ResultMale: Float = Float(InputWeight)*10.0
                        ResultMale += InputHeight*6.25
                        ResultMale -= Float(InputAge)*5.0
                        ResultMale += 5.0

                        let someVar = String(ResultMale)

                        Final.text = "\(someVar)"
                    }
                    else if InputGender == "Female"
                    {
                        var ResultFemale: Float = Float(InputWeight)*10.0
                        ResultFemale += InputHeight*6.25
                        ResultFemale -= Float(InputAge)*5.0
                        ResultFemale -= 161.0

                        let someVar = String(ResultFemale)

                        Final.text = "\(someVar)"
                    }
                    else
                    {
                        Final.text = "Just Male or Female"
                    }
                }
            }
            else
            {
                Final.text = "Input Gender again"
            }
        }
        else
        {
            Final.text = "Wrong Input"
        }
    }
    else
    {
        Final.text = "Wrong Input"
    }
}
else
{
    Final.text = "Wrong Input"
}
```

FIGURE 22. BMR calculation

In the figure above, the BMR result will only be calculated when all needed conditions are checked, which has already been explained in Exception Handling section.

```
var InputHeight: Float = 0.0
var InputWeight: Int = 0
var InputAge: Int = 0
var InputGender: String = ""
```

FIGURE 23. Local variables

As mentioned in the previous section, Handy Trainer will use local variables for storing the BMR result and inputs from users. As we can see in figure 23, there are four local variables for storing all input called InputHeight, InputWeight, InputAge and InputGender where Height is floating point type, Weight and Age are integer and gender is string.

This section will focus on showing the result or error on user interface (figure 24). If there are errors or problems in term of data input, users will be informed by some simple notification text as below:

```
Final.text = "No negative input PLEASE"
```

FIGURE 24. Error text

In Handy Trainer, there is a text view used to display the result or error. Swift allows developers to cast only string type variables to the text view with simple transfer.

```
let someVar = String(ResultMale)

Final.text = "\(someVar)"
```

FIGURE 25. Casting result to the display

Since the BMR result is calculated (figure 25) as a floating-point type variable and the text view only accepts string type, the result must be casted from floating-point type to string by using intermediary variable.

5 DISCUSSION

The final goal of this thesis is to develop an iOS application from scratch which includes designing user interface, implementing source code and testing application manually with collected data. This thesis used Swift as the main programming language that is developed by improving existing excellent features and removing drawbacks. The learning process was strengthened by applying real idea which is improving health with simple application. During the learning process, designing user interface is created with various simple yet powerful built-in objects provided from XCode. The design process is fast and straight forward because of the drop-and-drop features given. In the other hand, controlling user interface objects is where all back-end processes occurred. There are several cases which can crash or stop the application from users' input data. Therefore, before implementing the calculation, some exceptions need to be handled. As a result, the application can run within small chance from crashing. The application provides options for users to calculate their BMR according to weight, height, age and gender. Besides, users can choose training programs from provided options which suitable for their favor.

Developing an iOS application from scratch is an experience which can improve programming skills rapidly. The process of developing includes designing user interface called *view*, mechanism for controlling user interface's components called *controller* and the logic for processing data called *model*. The three components combine to provide a design pattern named MVC. All the steps of developing this iOS application take place in XCode with Swift 3.0 programming language. XCode is a powerful IDE introduced by Apple; therefore, the working environment is practical and user-friendly. Also, Swift is perfectly developed to match with the well-performance of XCode. As a result, developers can have a full package to implement an application from scratch.

Due to the modern and user-friendly programming language Swift, the process of executing this thesis is fast and on-time. Swift does not require old style of programming like C++ or C while providing more useful features for developers to take advantage. Comparing to other programming languages, Swift is reasonably fast by removing extra executing time and processes. Consequently, developers can feel comfortable when working with Swift, especially in XCode environment. However, there are still some limitations which hold Swift down such as

string cannot be passed by value while dictionary and array are treated like string. Thus, developers still need to learn some Objective-C before starting with Swift. Despite only a few drawbacks, Swift is still one of the most powerful programming languages for developers.

Utilizing the supported environment from XCode and Swift, Handy Trainer is developed quickly and efficiently. Handy Trainer comes with the most needed features such as BMR calculation and recommendations for work out programs. Handy Trainer can, currently, calculate the BMR according to normal metric unit system which needs the height in centimeter, the weight in kilogram, the gender and the age of user. Besides, Handy Trainer can recommend two types of training programs called HIIT and LISS which can be chosen regarding the taste of user. As a result, Handy Trainer is an assistant which can help users with the most basic calculation and information.

REFERENCES

Apple Inc. (2014). Core OS Layer. Available at: https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreOSLayer/CoreOSLayer.html#//apple_ref/doc/uid/TP40007898-CH11-SW1. Cited 22.02.2017.

Apple Inc. (2015). Model-View-Controller. Available at: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html. Cite 25.02.2017.

Apple Inc. (2017). About Swift. Available at: https://swift.org/about/#swiftorg-and-open-source. Cite 22.02.2017.

Apple Inc. (2017). Xcode 8. Available at: https://developer.apple.com/xcode/. Cite 22.02.2017.

Black, A. E. (2000). Critical evaluation of energy intake using the Goldberg cut-off for energy intake: basal metabolic rate. *A practical guide to its calculation, use and limitations. International journal of obesity*, 24(9), 1119.

Burton-Freeman, B. (2000). Dietary fiber and energy regulation. *The Journal of nutrition*, 130(2), 272S-275S.

Crocco, A. G., Villasis-Keever, M., & Jadad, A. R. (2002). Analysis of cases of harm associated with use of health information on the internet. *Jama*, 287(21), 2869-2871.

Dembe, A. E. (2009). Ethical issues relating to the health effects of long working hours. *Journal of Business Ethics*, 84, 195-208.

Duyff, R. L. (2012). *American dietetic association complete food and nutrition guide*. Houghton Mifflin Harcourt.

Goadrich, M. H., & Rogers, M. P. (2011, March). Smart smartphone development: iOS versus Android. *In Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 607-612). ACM.

Gonzalez-Sanchez, J., & Chavez-Echeagaray, M. E. (2010, October). iPhone application development. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (pp. 321-322). ACM.

Hall, K. D., Heymsfield, S. B., Kemnitz, J. W., Klein, S., Schoeller, D. A., & Speakman, J. R. (2012). Energy balance and its components: implications for body weight regulation. *The American journal of clinical nutrition*, 95(4), 989-994.

Hillyer, N. (2016). The Ultimate Guide to Choosing Objective-C or Swift for Your Project. Available at: https://savvyapps.com/blog/ultimate-guide-choosing-objective-c-or-swift. Cite 22.02.2017.

Klein, S., Coyle, E. F., & Wolfe, R. R. (1994). Fat metabolism during low-intensity exercise in endurance-trained and untrained men. *American Journal of Physiology-Endocrinology And Metabolism*, 267(6), E934-E940.

Laforgia, J., Withers, R. T., & Gore, C. J. (2006). Effects of exercise intensity and duration on the excess post-exercise oxygen consumption. *Journal of sports sciences*, 24(12), 1247-1264.

Laursen, P. B., & Jenkins, D. G. (2002). The scientific basis for high-intensity interv training. *Sports Medicine, 32*(1), 53-73.

Lowe, J. C., Yellin, J., & Honeyman-Lowe, G. (2006). Female fibromyalgia patients: lower resting metabolic rates than matched healthy controls. *Medical science monitor*, 12(7), CR282-CR289.

Luke, A., & Schoeller, D. A. (1992). Basal metabolic rate, fat-free mass, and body cell mass during energy restriction. *Metabolism*, 41(4), 450-456.

Mifflin, M. D., St Jeor, S. T., Hill, L. A., Scott, B. J., Daugherty, S. A., & Koh, Y. O. (1990). A new predictive equation for resting energy expenditure in healthy individuals. *The American journal of clinical nutrition*, 51(2), 241-247.

Milanović, Z., Sporiš, G., & Weston, M. (2015). Effectiveness of high-intensity interval training (HIT) and continuous endurance training for VO2max improvements: a systematic review and meta-analysis of controlled trials. *Sports medicine, 45*(10), 1469-1481.

Sacks, F. M., Bray, G. A., Carey, V. J., Smith, S. R., Ryan, D. H., Anton, S. D., & Leboff, M. S. (2009). Comparison of weight-loss diets with different compositions of fat, protein, and carbohydrates. *N Engl J Med*, 2009(360), 859-873.

Schoenfeld, B. (2011). Does cardio after an overnight fast maximize fat loss?. *Strength & Conditioning Journal*, 33(1), 23-25.

Solt, P. (2015). Swift vs. Objective-C: 10 reasons the future favors Swift. Available at: http://www.infoworld.com/article/2920333/mobile-development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html. Cite 22.02.2017.

Wee, A. (2017). Microsoft Surface Phone: A new threat for Apple iPhone?. Nexus. Chicago

APPENDIX 1 – COLLECTING DATA SURVEY FORM

Measurement collecting survey

Measurements will be collected to test manually Handy Trainer

What is your current weight ? (Please input by kilograms)

Your answer

What is your current height ? (Please input by meters)

Your answer

How old are your ? (Please input natural number)

Your answer

What is your gender ?

- ○ Male
- ○ Female

SUBMIT

Never submit passwords through Google Forms.