



**SAN JOSÉ STATE  
UNIVERSITY**

**CMPE 277- Smart Phone Application Development**

**iHeart Monitor - Heart Rate Anomaly Detection**

**Submission Date**

24 May, 2018

**Submitted To**

Prof. Chandrasekar Vuppalapati

**Submitted By**

**Team: Code Monkeys**

<b>Harini Balakrishnan</b>	<b>010830755</b>
<b>Vidhi Sharma</b>	<b>012466155</b>
<b>Vijay Samuel</b>	<b>012506546</b>
<b>Yamini Muralidharen</b>	<b>012449632</b>

## ABSTRACT

The evolution of the technology has contributed to the adoption of smartphones and wearable in health monitoring applications. The smartphone's capabilities are not only restricted to make calls or send messages but also has a wide variety of use cases in healthcare applications due to its extensive processing power and cloud connectivity. On the other hand, wearable collect data from users' activities with the help of various inbuilt sensors. Currently, smartphones have gained features by integrating with features like public cloud services, Machine Learning, IoT and many other trending technologies. One of the most important features is that smartphones have the ability to monitor its surroundings, movements and various other things with the help of the sensors. Sensors are available both as paid accessories in the market or few smartphones have inbuilt sensors. These sensors provide facilities to monitor the health of the human beings like tracking their heart rate, food habits, activities, and sleep. One of the important factor to analyze the day-to-day health is food and heart rate. We wrote a paper with detail understanding of various health monitoring sensors and came up with a probabilistic determination of the consumer's heart health by collecting a range of human heart rate details using the smartphone sensors. In this project, we have tried to present an approach to data to detect anomalies in the heart rate by building a machine learning regression model from the time series data of user daily physical activities and heart rate values. The user activities are monitored by the Apple smartwatch and synced with iPhone application that is integrated with the Heathkit. This smart application gives a privilege of sharing their fitness activities with their friends and also alerts the users by giving custom notifications based on the activity track record.

### ***Keywords:***

*Apple, Smartphone, Apple Watch, HealthKit, Anomaly detection , Health Monitoring, Machine learning, Regression model, Time series data, Cloud, Firebase, iOS, Charts.*

## TABLE OF CONTENTS

<b>ABSTRACT</b>	<b>2</b>
<b>1. PROJECT DESCRIPTION</b>	<b>8</b>
<b>2. REQUIREMENTS</b>	<b>8</b>
2.1 Functional Requirements:	9
2.2 Hardware Requirements:	9
2.3 Software Requirements:	9
<b>3. MOBILE UI DESIGN PRINCIPLES</b>	<b>9</b>
<b>4. HIGH LEVEL ARCHITECTURE DESIGN □</b>	<b>10</b>
<b>5. COMPONENT LEVEL DESIGN</b>	<b>11</b>
<b>6. SEQUENCE/WORKFLOW</b>	<b>12</b>
<b>7. MOBILE AND CLOUD TECHNOLOGIES</b>	<b>13</b>
7.1 Mobile Technologies	13
7.2 Cloud Technologies	13
<b>8. INTERFACES RESTFUL AND SERVER SIDE DESIGN</b>	<b>13</b>
8.1 □Healthkit	13
8.2 Firebase	13
8.3 Firebase Setup:	14
8.4 Adding Firebase SDK to project:	14
8.2 Image scanning using Firebase MLKit	21
8.3 Anomaly detection using Firebase MLKit custom models	22
<b>9. CLIENT SIDE DESIGN</b>	<b>25</b>
9.1 Firebase Authentication	25
9.1.1 Login and Registration	25
9.1.2 Logout	26
9.2 Healthkit - Heart rate	26
9.3 Healthkit - Step Count	30

iHeart Monitor - Heart Rate Anomaly Detection	4
<b>10. TESTING</b>	31
<b>11. DESIGN PATTERNS USED</b>	33
11.1 Storyboard	33
11.2 Navigation controller	34
11.3 Tab Bar Controller	35
11.4 Segue	36
11.5 Segmented control	36
11.6 View Controllers	37
11.7 Charts	37
<b>12. PROFILING</b>	37
12.1 Xcode Profiler	37
12.2 Firebase Profiling	40
<b>13. Future work</b>	41
<b>14. References</b>	42
<b>TEAM MEMBER CONTRIBUTION</b>	43

## TABLE OF FIGURES

<b>Figure 1: iHeartMonitor App Architecture design</b>	<b>9</b>
<b>Figure 2: iHeartMonitor App MVC Pattern</b>	<b>9</b>
<b>Figure 3: Component Level design</b>	<b>10</b>
<b>Figure 4: iHeartMonitor Sequence/Workflow diagram</b>	<b>11</b>
<b>Figure 5: Launch Screen</b>	<b>14</b>
<b>Figure 6: Main Screen</b>	<b>14</b>
<b>Figure 7: Login Screen</b>	<b>15</b>
<b>Figure 8: Wrong email format alert generated in Login Screen</b>	<b>16</b>
<b>Figure 9: Wrong password credentials alert generated in Login Screen</b>	<b>16</b>
<b>Figure 10: Healthkit data access request</b>	<b>17</b>
<b>Figure 11: Healthkit data authorized</b>	<b>17</b>
<b>Figure 12: User information from Healthkit</b>	<b>18</b>
<b>Figure 13: Apple Watch</b>	<b>18</b>
<b>Figure 14: Requesting access for alerts</b>	<b>19</b>
<b>Figure 15: Alert generated on low step count</b>	<b>19</b>
<b>Figure 16: Requesting access for camera</b>	<b>20</b>
<b>Figure 17: Takes picture of the food facts</b>	<b>20</b>
<b>Figure 18: Code snippet for Image Picker</b>	<b>21</b>
<b>Figure 19 : Heart rate Prediction Model</b>	<b>22</b>
<b>Figure 20: Code snippet for adding email and password authentication using Firebase</b>	<b>23</b>

<b>Figure 21: Code snippet for adding email and password registration using Firebase</b>	<b>24</b>
<b>Figure 22: Code snippet for logging out using Firebase</b>	<b>25</b>
<b>Figure 23: Code snippet for fetching weekly heart rate</b>	<b>26</b>
<b>Figure 24: Code snippet for fetching daily heart rate</b>	<b>27</b>
<b>Figure 25: Code snippet for fetching monthly heart rate</b>	<b>28</b>
<b>Figure 26: Heart rate line chart for the month</b>	<b>29</b>
<b>Figure 27: Code snippet for fetching daily step count</b>	<b>30</b>
<b>Figure 28: Code snippet for unit testing using Xcode Test Navigator</b>	<b>31</b>
<b>Figure 29: Code snippet for unit testing using UI testing</b>	<b>31</b>
<b>Figure 30: Application Storyboard</b>	<b>32</b>
<b>Figure 31: Navigation Controller flow</b>	<b>33</b>
<b>Figure 32: Tab View Controller flow</b>	<b>35</b>
<b>Figure 33: CPU usage in Xcode Instrument profiler</b>	<b>37</b>
<b>Figure 34: Memory usage in Xcode Instrument profiler</b>	<b>38</b>
<b>Figure 35: Disk usage in Xcode Instrument profiler</b>	<b>38</b>
<b>Figure 36: Network usage in Xcode Instrument profiler</b>	<b>39</b>
<b>Figure 37: App users sign-in details on firebase console</b>	<b>39</b>
<b>Figure 38: Active users monitoring on firebase</b>	<b>40</b>
<b>Figure 39: Users activity and App cash monitoring on firebase console</b>	<b>40</b>

# 1. PROJECT DESCRIPTION

The statistics by World Health Organisation say that more than 20 million people died due to cardiovascular disease. The reasons behind cardiovascular disease are mainly attributed to Heredity ,Obesity, Hypertension, Blood pressure etc. The current trends say that every 4 in 10 adults are obese. The growing trend of obesity now is due to the fact of unhealthy lifestyle and inactivity. The aim of this application is to continuously collect user heart rates and monitor the user activities by using Apple Watch.

This application provides statistics of the time series data of heart beats. All the user health data are secured on the device. Users initially signup/login using credentials that are stored in the Firebase for Authentication. This application is integrated with the Apple health kit to collect the user heart rate and the user step count data from the Apple Watch and manipulate these data to provide users a better statistical view of their heart rate. In addition, we have tried to implement a calorie counter based on the food that the user consume.

# 2. REQUIREMENTS

The main functionality of the applications is to detect deviations in the heart rate ,track user activity and send notification to user in case of inactivity. We need to have some specific set of requirements to implement the application in iOS. From the user point of view, these are the following requirements that are necessary to run our application.

## 2.1 Functional Requirements:

- User logs in with Email username and password.
- User authorizes the application to use the Healthkit data
- User authorizes the application to use the mobile device's camera
- User authorizes the application to use the mobile device's photo gallery
- User will get notifications about their inactivity

## **2.2 Hardware Requirements:**

- Apple iPhone (iPhone 5s or later)
- Apple Watch

## **2.3 Software Requirements:**

- Xcode 8 or later
- Firebase Auth
- Firebase
- Google account
- Apple HealthKit library
- Cocoa Pods
- Charts library
- Firebase MLKit API

# **3. MOBILE UI DESIGN PRINCIPLES**

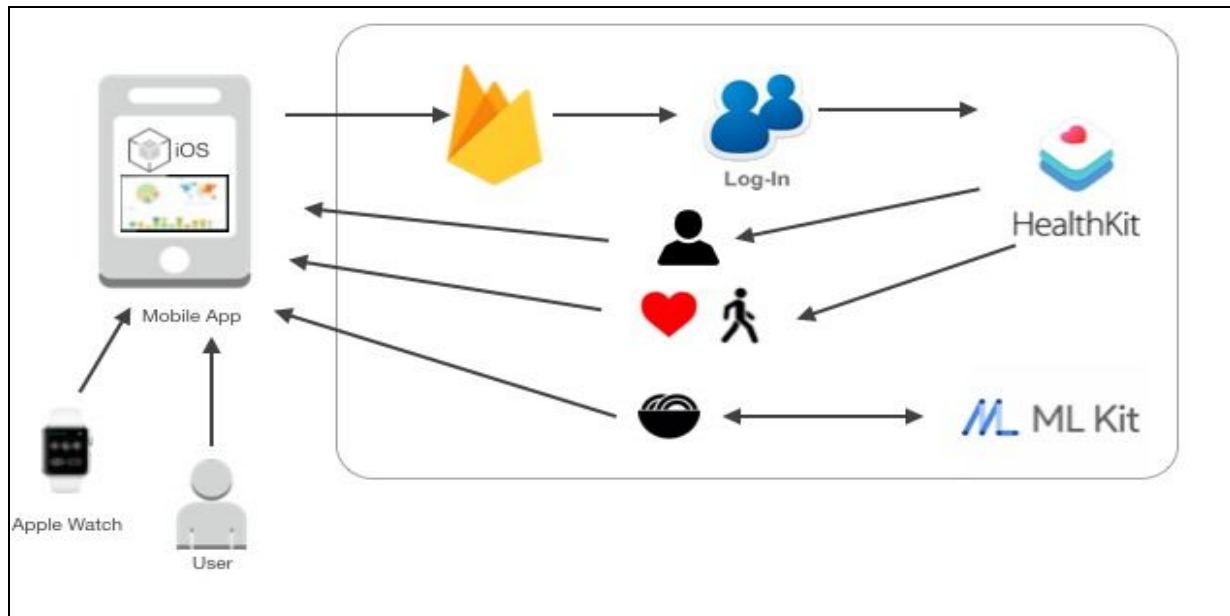
We have adopted the best practices when designing and developing our mobile application

- Short Message on Notifications and Alerts
- Keeping the UI content crisp and clear
- Providing great readability to the users by having correct font size
- The colors we have used in the application provide clear visibility to the target users
- Adding optimal UI components on the so that it does not look cluttered.
- The navigation should be self explanatory and easy to use
- The visual theme offered is consistent

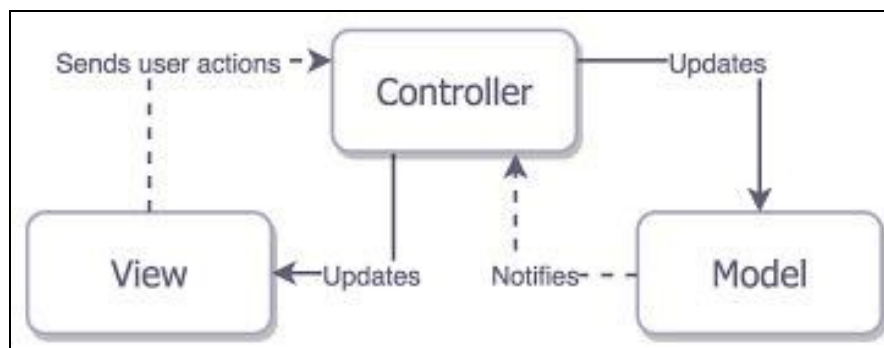


## 4. HIGH LEVEL ARCHITECTURE DESIGN □

In this section, we have discussed about the high level architectural overview of our application. The following diagram represents overview of various blocks connected together to form the application.



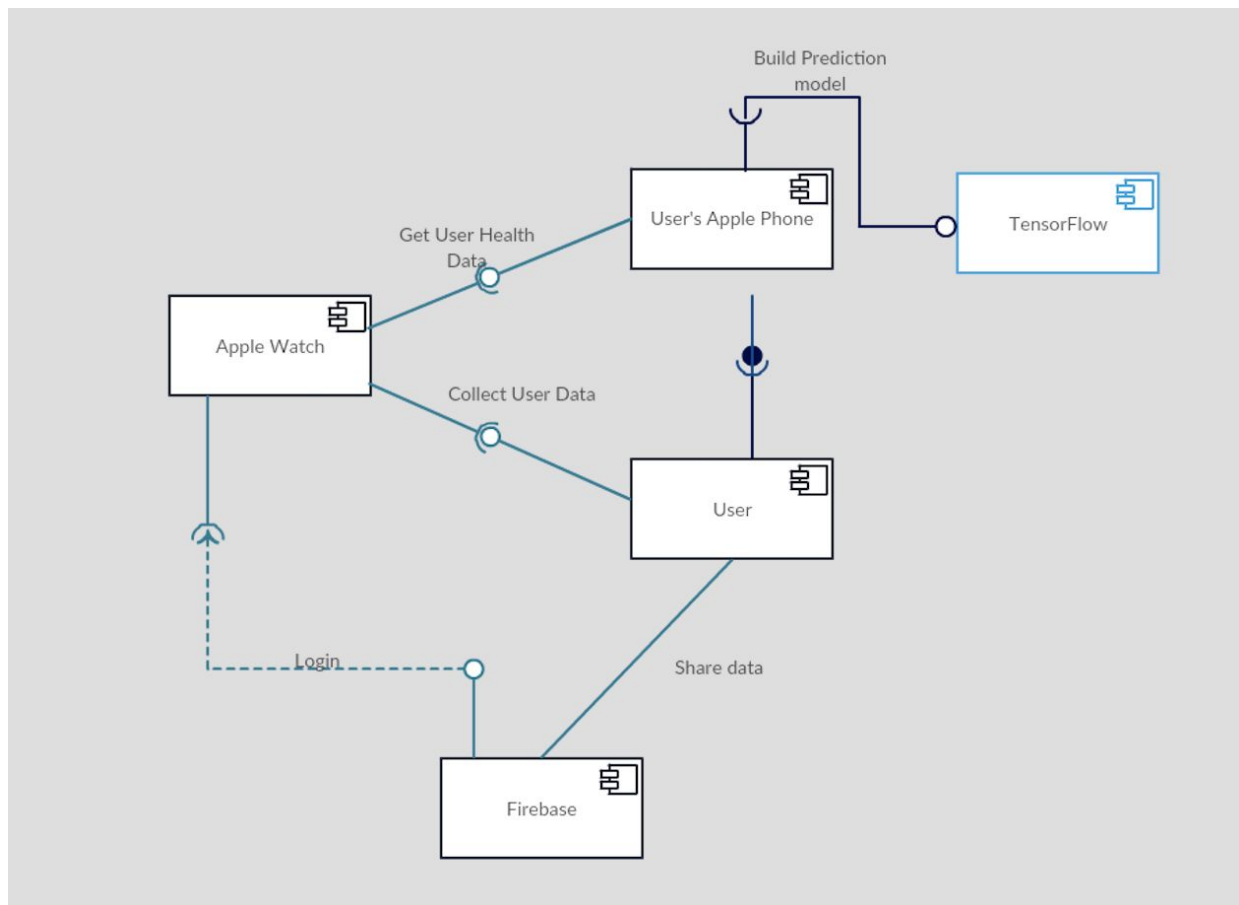
**Figure 1: iHeartMonitor App Architecture design**



**Figure 2: iHeartMonitor App MVC Pattern**

## 5. COMPONENT LEVEL DESIGN

The component level design presents the different components that are integrated to form the application. We have several several components that represent a distinct functionality dependant on each other. The following figure represents the component diagram of our application



**Figure 3: Component Level design**

## 6. SEQUENCE/WORKFLOW

The following figure represents the flow of events when the end user makes use of the application. The users logs in to the application and views the dashboard. The user's health information is fetched from apple watch via health kit. The user can navigate to different tabs to view heart rate graph, step count information and food calorie tracker.

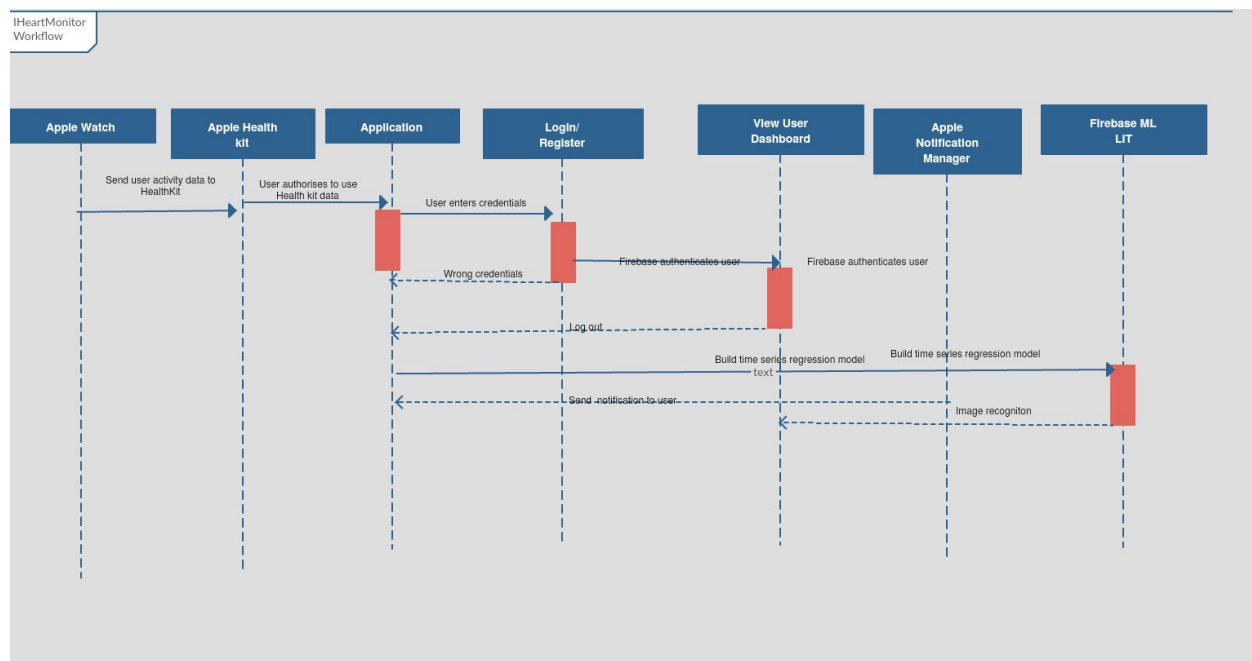


Figure 4: iHeartMonitor Sequence/Workflow diagram

## **7. MOBILE AND CLOUD TECHNOLOGIES**

### **7.1 Mobile Technologies**

The mobile technologies that form a integral part of this project are

- Apple HealthKit
- Apple Watch
- iOS

### **7.2 Cloud Technologies**

- FirebaseAuth
- Firebase ML kit

## **8. INTERFACES RESTFUL AND SERVER SIDE DESIGN**

### **8.1 Healthkit**

Apple created a software development kit for iOS and WatchOS, this kit is “a single place where apps can store, share, and read health-related data”. The user’s information is stored in a centralized, secure location and the user has the right to decide which data to share. In our application we have used Healthkit as health data source, we have analyzed the data and perform different operations and alerts on the data.

### **8.2 Firebase**

Firebase is tool developed by Google which helps in developing high quality mobile and web applications quickly. It is a Backend-as-a-Service (BAAS). Server side programming is not needed. Firebase acts as server, API and data store and all this can be modified according to the needs. The features provided by Firebase:

- Authentication: Provides a database like feature for managing users. Firebase has built in email/password authentication system, it also supports OAuth2 for some applications like Twitter, Google, Facebook and GitHub.
- Storage: Application data can be stored in Firebase. It has its own set of security rules
- Testing: Provides platform for A/B Testing
- Crash: Notifies about the app crashes

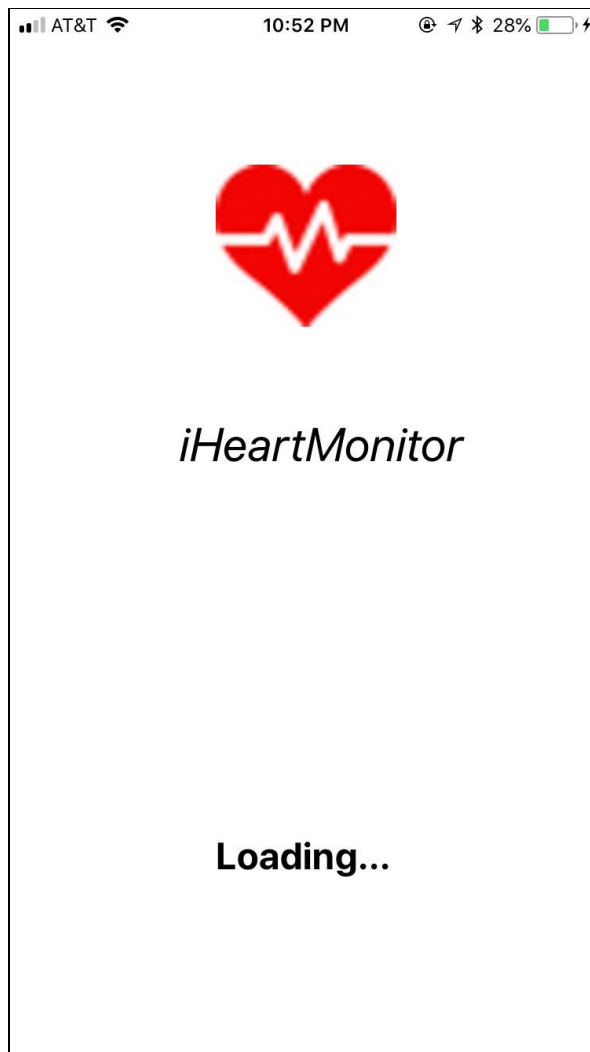
### **8.3 Firebase Setup:**

Steps for setting up Firebase account and adding Firebase to iOS app:

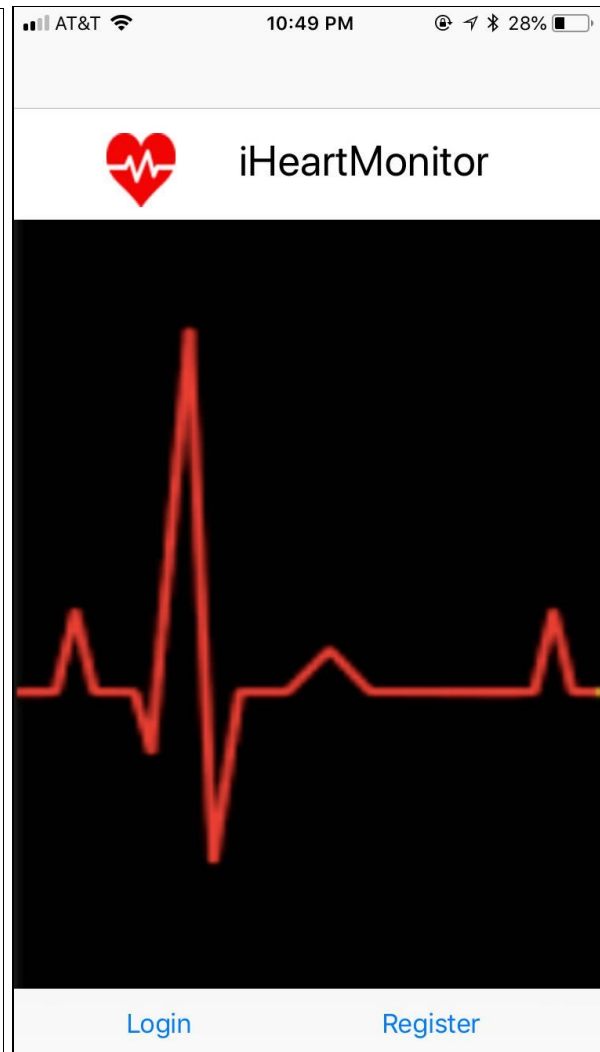
1. In Firebase console create a new iOS project
2. Add the bundle id during the setup process. It's necessary to keep same bundle id for the iOS app and Firebase project.
3. Download 'GoogleService-Info.plist' file
4. Add 'GoogleService-Info.plist' file in Xcode project.

### **8.4 Adding Firebase SDK to project:**

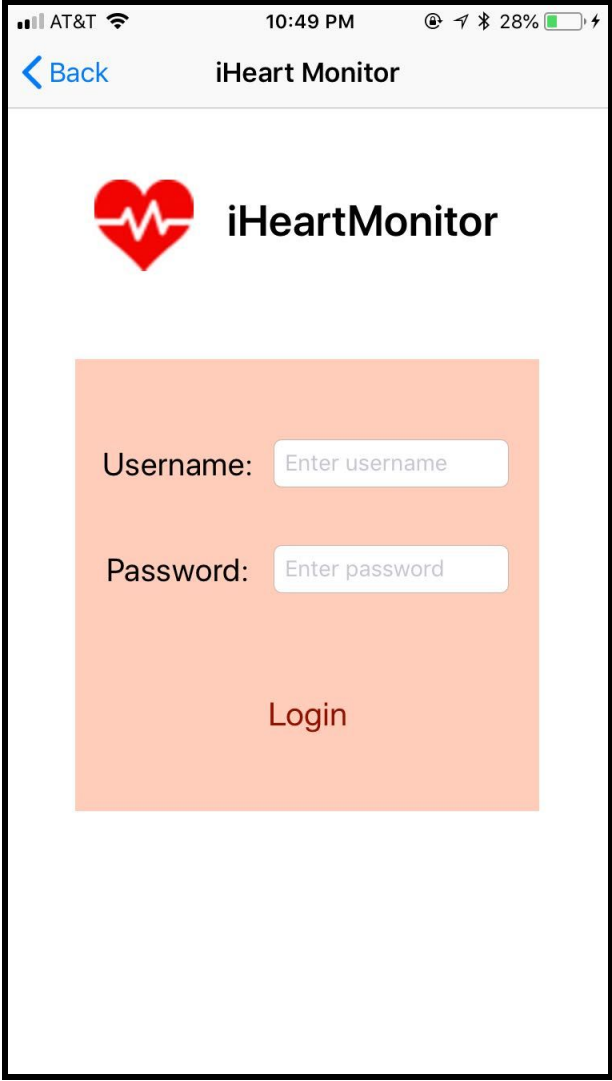
1. Install Cocoapods
2. Inside the app workspace, create a pod file using 'pod init' command
3. Add Firebase pods in the file
4. Use 'pod install' command to install all the
5. Open app workspace using 'open project Name.workspace'
6. Initialize Firebase in the app by adding 'FirebaseApp.configure()' in app Delegate file.  
(need to use import Firebase first by using 'import Firebase' command)



**Figure 5: Launch Screen**

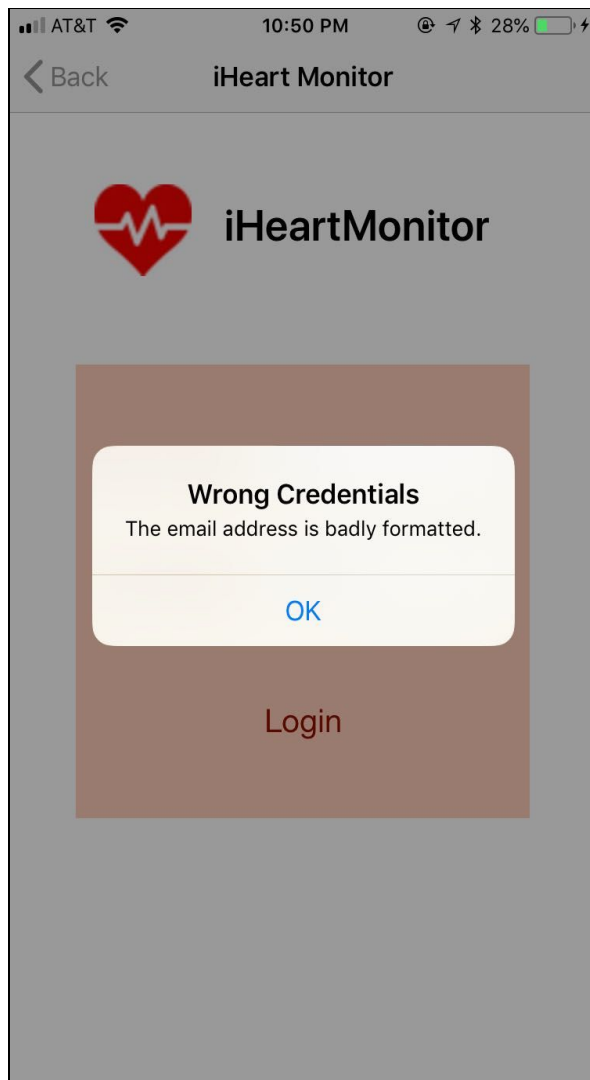


**Figure 6: Main Screen**

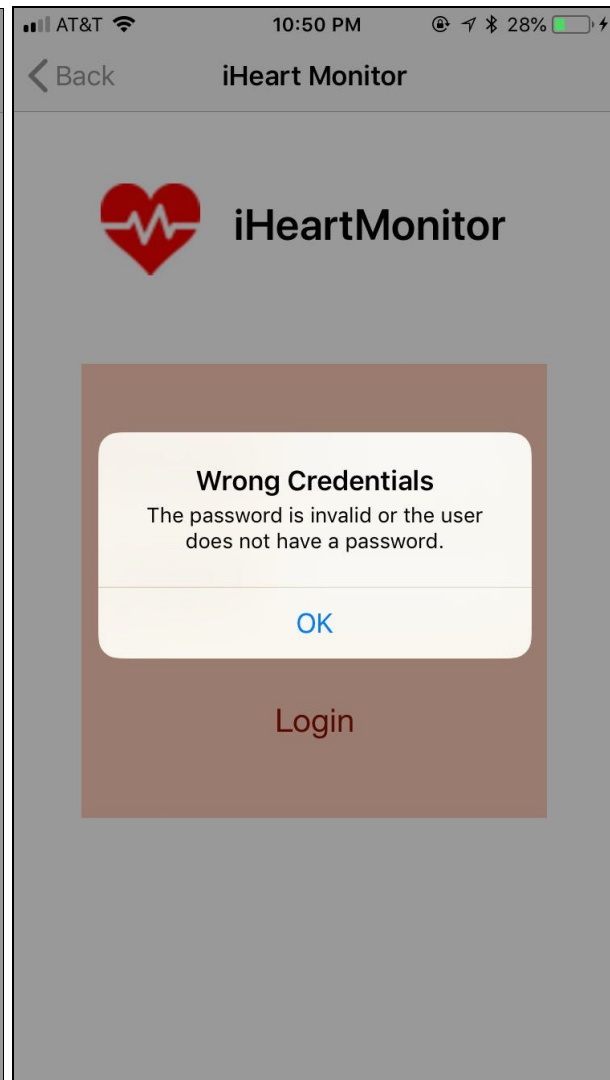


The image shows a mobile application login screen for "iHeart Monitor". At the top, there is a status bar with "AT&T", signal strength, time "10:49 PM", and battery level "28%". Below the status bar is a navigation bar with a blue "< Back" button and the title "iHeart Monitor". The main content area features a red heart icon with a white ECG line, followed by the text "iHeartMonitor". Below this is a light orange rectangular box containing the login form. The form has two input fields: "Username:" with a placeholder "Enter username" and "Password:" with a placeholder "Enter password". At the bottom of the orange box is a red "Login" button.

**Figure 7: Login Screen**



**Figure 8: Wrong email format alert  
generated in Login Screen**



**Figure 9: Wrong password credentials alert  
generated in Login Screen**



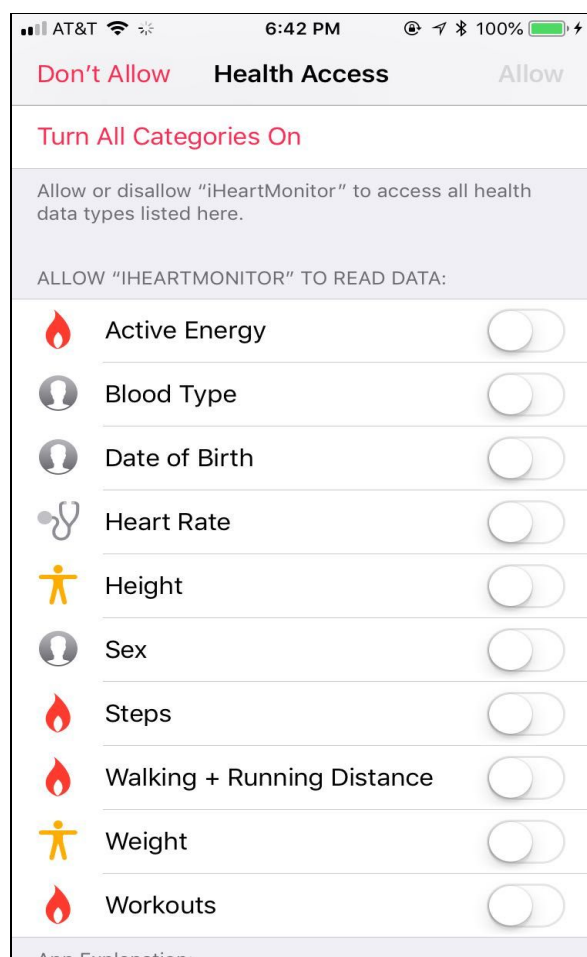


Figure 10: Healthkit data access request

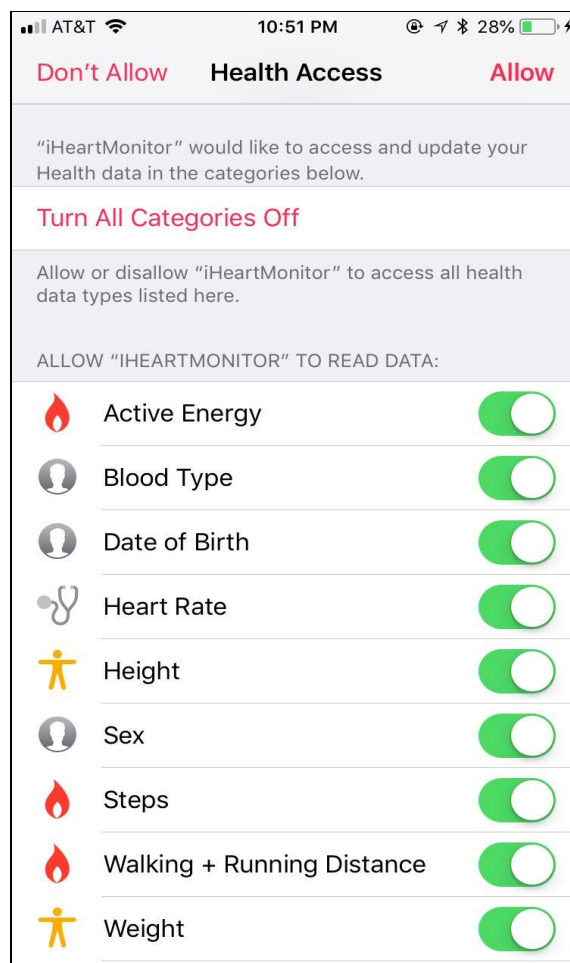
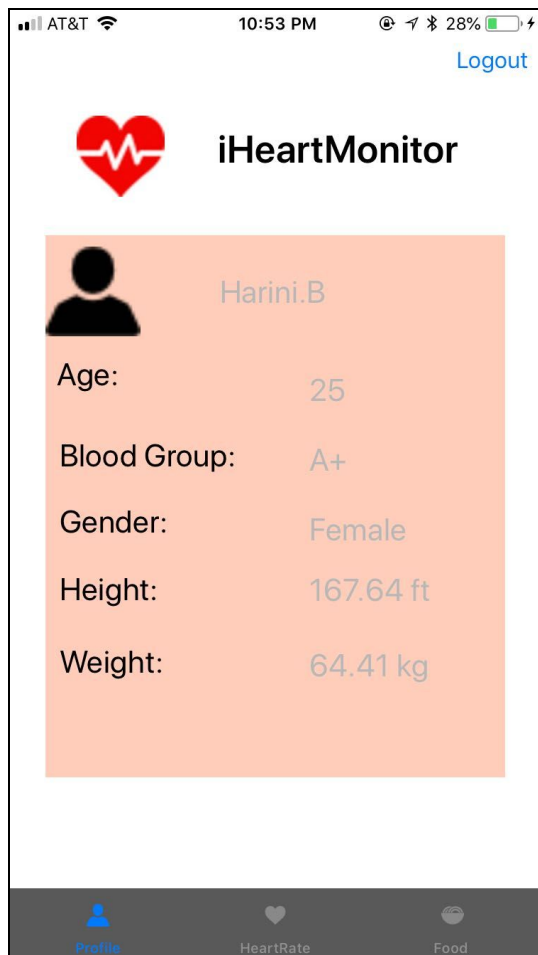
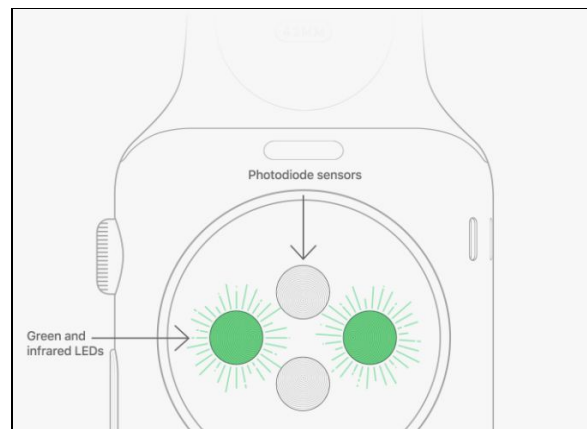


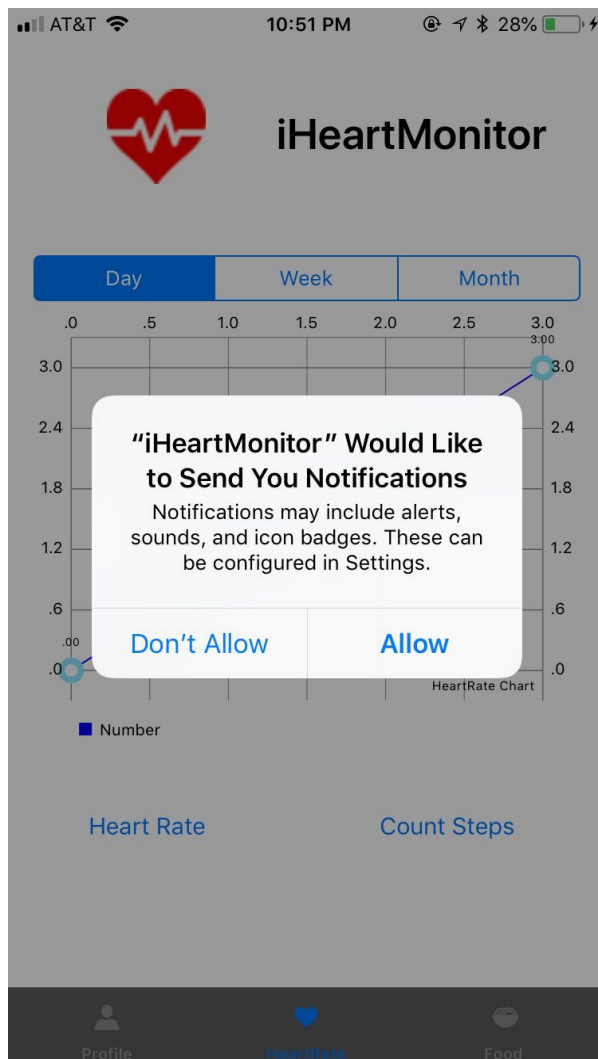
Figure 11: Healthkit data authorized



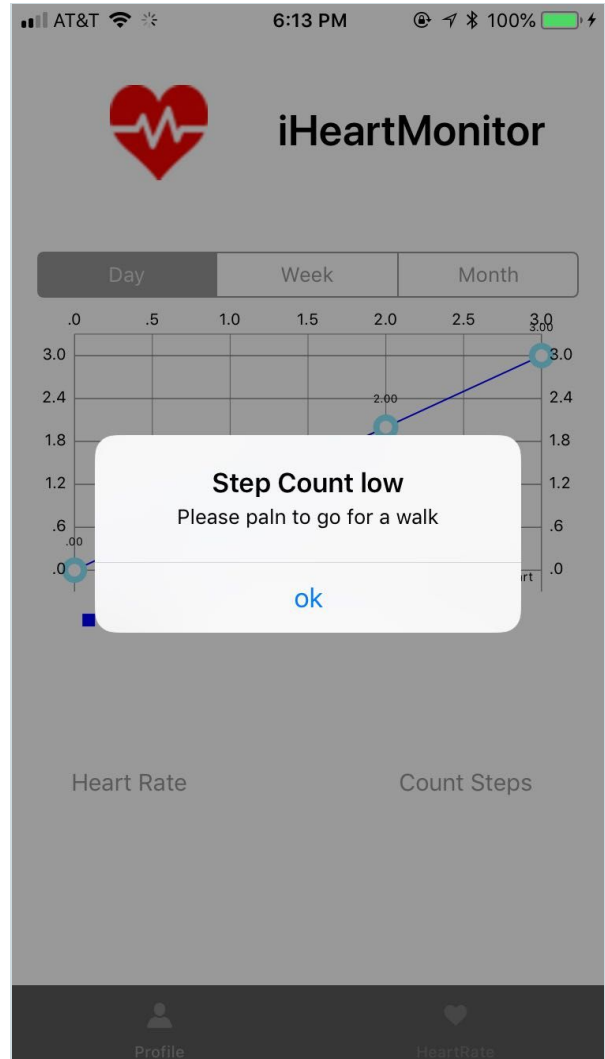
**Figure 12: User information from Healthkit**



**Figure 13: Apple Watch**



**Figure 14: Requesting access for alerts count**



**Figure 15: Alert generated on low step count**

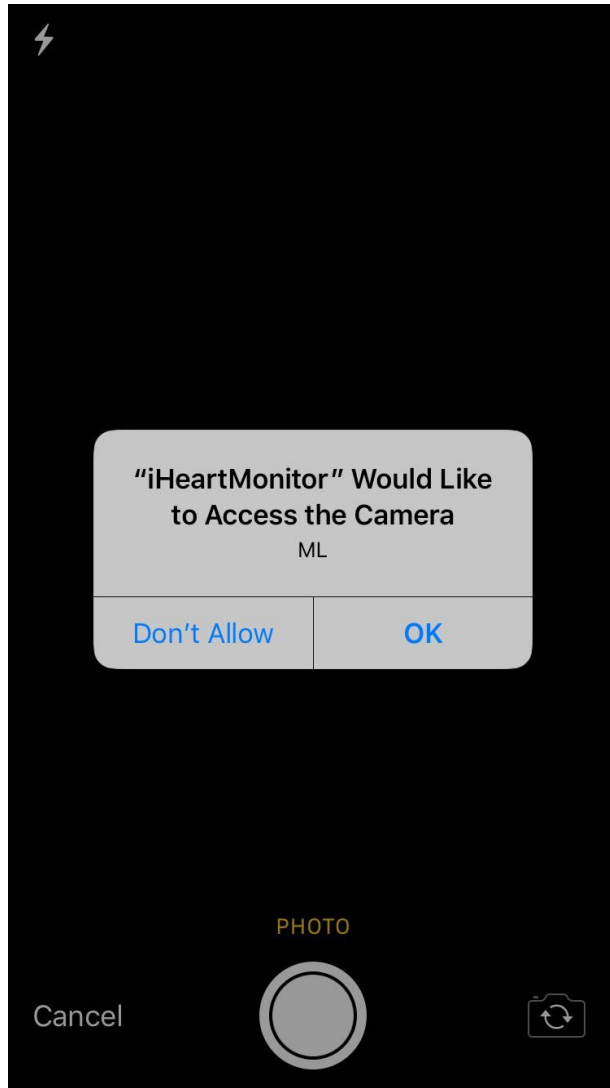


Figure 16: Requesting access for camera

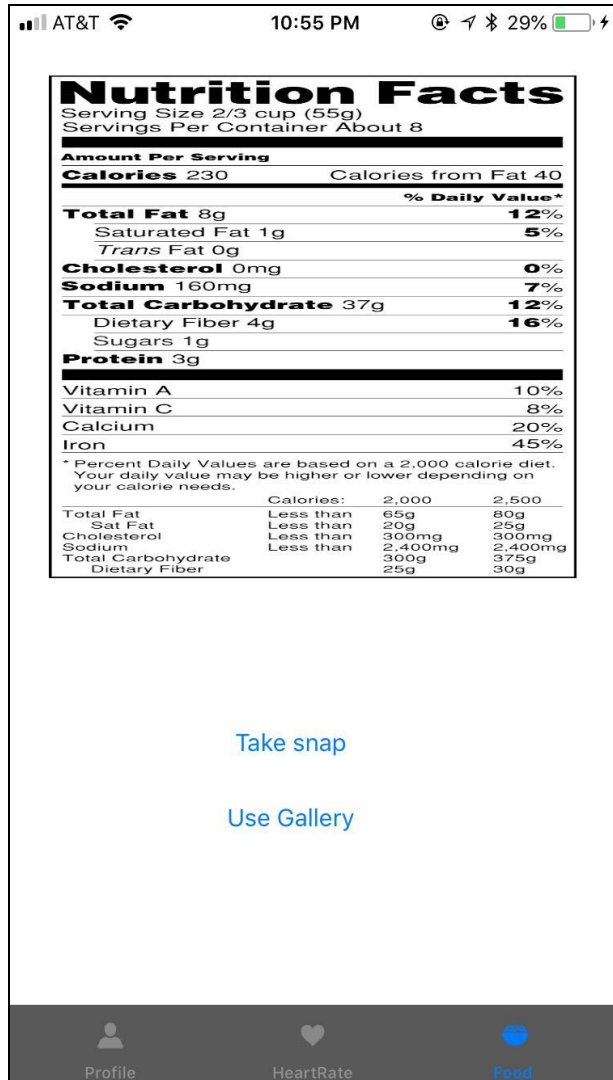


Figure 17: Takes picture of the food facts

## 8.2 Image scanning using Firebase MLKit

Firebase on May 8th 2018 announced Firebase MLKit which allows users to use standard ML based APIs to do things like labeling, text recognition and custom machine learning models on the device. We have taken the text recognition APIs and built a text recognition utility that can read food labels and get the text from it. The API used is called Vision Text Detector. The API scans the image and returns a string from the text it was able to detect. Once received, we try to get information from the label like calorie count, trans fat, sodium etc. Using this, we can alert

the user on unhealthy food consumed and enter the nutrition information back into the HealthKit APIs. The code for recognizing text from the image is as follows:

```
guard let image = imageView.image else {
    return }
    textDetector =
self.vision.textDetector()
    let visionImage =
VisionImage(image: image)

    textDetector.detect(in:
visionImage) { (features, error) in
        guard error == nil, let
features = features, !features.isEmpty
        else {
            print("Text detection
failed with error: \(error)")
            return
        }
    }
```

### 8.3 Anomaly detection using Firebase MLKit custom models

Since we have all of the user's health related information, it is easy to perform more complex machine learning model based predictions from it. We had taken a subset of the heartbeat information and tried to train an AR regression algorithm from tensorflow. We had also introduced artificial spikes in the heart rate to symbolize exercise periods. When the trained model was requested to predict the next 100 heartbeat values, it was able to provide a fairly accurate pattern that denoted exercising intervals as per schedule. The below diagram conveys the prediction done by the AR regression:

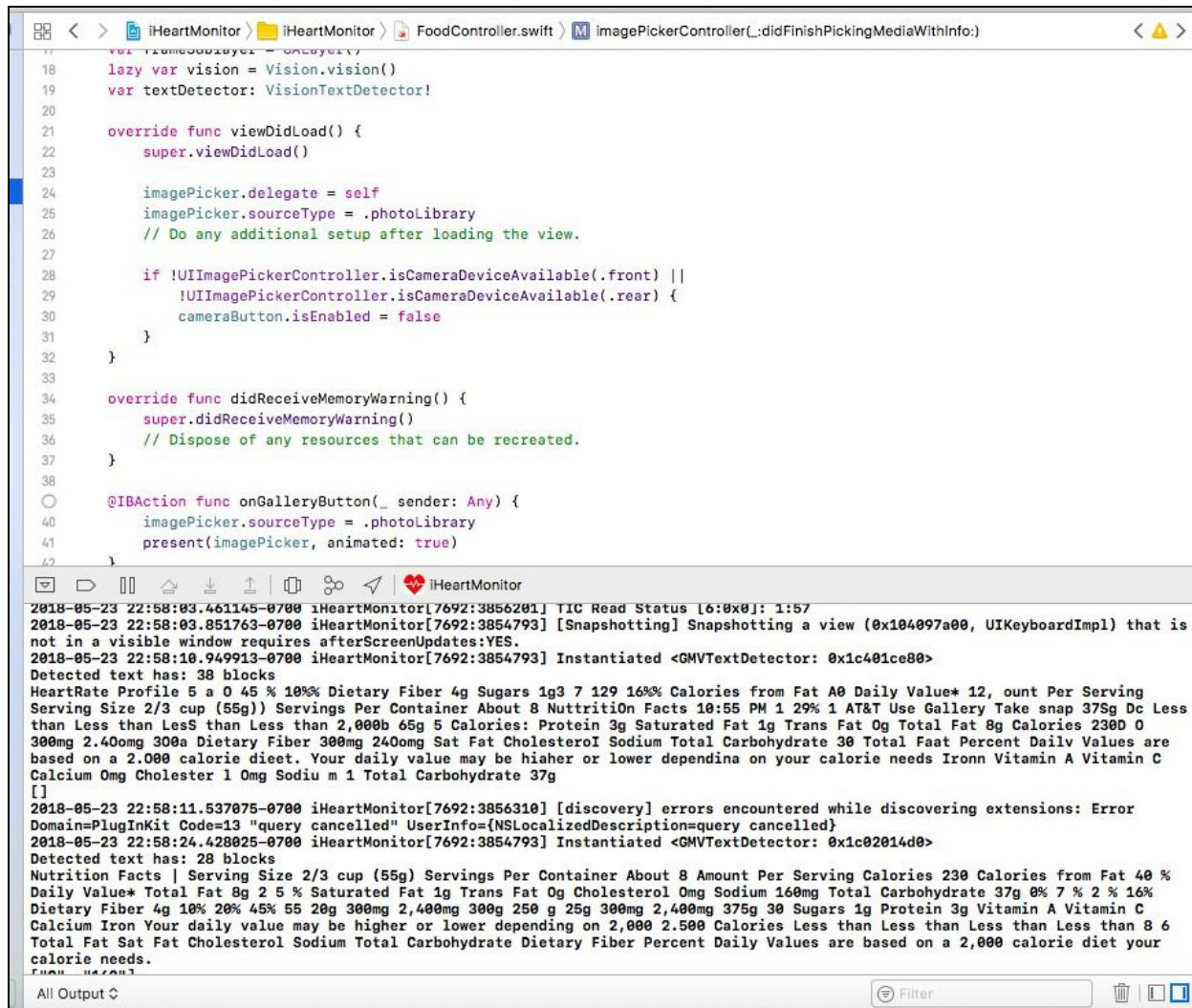
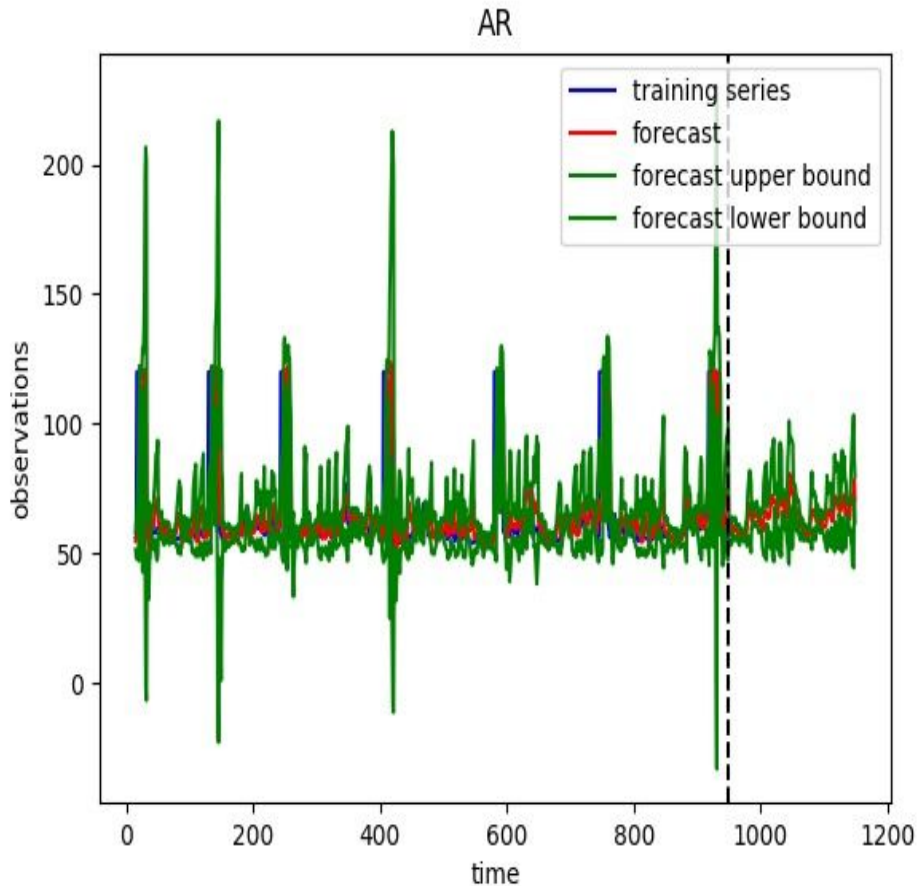


Figure 18: Code snippet for UIImagePickerController



**Figure 19 : Heart rate Prediction Model**

Firebase MLKit offers the capability to load models into the phone once a tensorflow lite file is generated from a tensorflow frozen graph. However since ML Kit is still very new, tensorflow lite still doesn't support the conversion of complex regression models like AR regression. Hence we were not able to load the model into the phone. It was still possible to load the model into Google Cloud's Cloud ML engine but that would require us to upload user heartbeat information into Google Cloud which goes against the privacy features that we want to enable in the application. A user must be able to keep all of his/her information safely on the phone without having to put it on the cloud. Hence we deferred this feature until MLKit supports more complex models like regression and LSTM. This will allow us to predict if the user's heartbeat pattern is deteriorating over a period of time or if the user did not exercise in a given day.



## 9. CLIENT SIDE DESIGN

### 9.1 Firebase Authentication

#### 9.1.1 Login and Registration

Firebase authentication using email/password is used for login and signup.

In the pod file add 'Firebase/Core' and 'Firebase/Auth' pods.

In the below figure 20: code to add firebase email/password authentication is mentioned. An alert is created to notify the user about wrong credentials.

While registration the email id provided should be valid, if not alert will be generated. Password should be more than six characters otherwise generated and alert. The code snippet for registration can be seen in figure 20.

```
@IBAction func login(_ sender: UIButton) {  
    if let email = userName.text, let password = userPassword.text{  
  
        Auth.auth().signIn(withEmail: email, password:password, completion: {(user, error) in  
            if let firebaseError = error{  
                // create the alert  
                let alert = UIAlertController(title: "Wrong Credentials", message: firebaseError.localizedDescription , preferredStyle:  
                    UIAlertControllerStyle.alert)  
  
                // add an action (button)  
                alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.default, handler: nil))  
  
                // show the alert  
                self.present(alert, animated: true, completion: nil)  
                return  
            }  
            //self.presentLoggedInScreen()  
            //print("sucess")  
            self.goToHome()  
        })  
    }  
}
```

**Figure 20: Code snippet for adding email and password authentication using Firebase**



```
if let email = userName.text, let password = userPassword.text{
    Auth.auth().signIn(withEmail: email, password:password, completion:{(user, error) in
        if let firebaseError = error{
            // create the alert
            let alert = UIAlertController(title: "Wrong Credentials", message:
                firebaseError.localizedDescription , preferredStyle: UIAlertControllerStyle.
                alert)

            // add an action (button)
            alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.default,
                handler: nil))

            // show the alert
            self.present(alert, animated: true, completion: nil)

            return
        }
        //self.presentLoggedInScreen()
        //print("sucess")
        self.goToHome()
    })
}
```

**Figure 21: Code snippet for adding email and password registration using Firebase**

### 9.1.2 Logout

```
//logout
@IBAction func Logout(_ sender: Any) {

    try! Auth.auth().signOut()
    if let storyboard = self.storyboard {
        let vc = storyboard.instantiateViewController(withIdentifier: "ViewController") as! UINavigationController
        self.present(vc, animated: false, completion: nil)
    }

}
```

**Figure 22: Code snippet for logging out using Firebase**

## 9.2 Healthkit - Heart rate

Heart rate data can be obtained from the Apple Healthkit. The user needs to authorize the health store to read users' vitals. The heart rate is collected throughout the day and the results can be obtained from the health kit as time series data by using health kit queries. We obtain the heart rate value series for day, week and month by corresponding healthkit queries. The heart rate values is stored in HKQuantitySample subclass of HkSample class in Healthkit store. We use a

HKSampleQuery query with particular conditions. We need the predicate field for setting the time interval for which we need to obtain the heart rate values. Say for example if we want heartbeat values for a week, the start date would be time difference value of 7 days and end date would be current time. The following code is used for fetching the heart rate values for a week. The observer query can be used to continuously monitor the vitals and statistics queries are used to provide minimum, maximum and average calculations for a period of time.

```
func fetchWeeklyHeartRate(completionHandler: @escaping (_ sample: HKQuantitySample?) -> Void) {
    print("Fetching weekly heart rate data")

    let quantityType : Set = [HKObjectType.quantityType(forIdentifier: HKQuantityTypeIdentifier.heartRate)!]

    //Fetch the last 7 days of HEARTRATE.

    let startDate = Date.init(timeIntervalSinceNow: -7*24*60*60)
    let endDate = Date()
    let predicate = HKQuery.predicateForSamples(withStart: startDate,
                                                end: endDate,
                                                options: .strictStartDate)
    let sortDescriptor = NSSortDescriptor(key: HKSampleSortIdentifierStartDate, ascending: false)
    let sampleQuery = HKSampleQuery(sampleType: quantityType.first!,
                                    predicate: predicate,
                                    limit: HKObjectQueryNoLimit,
                                    sortDescriptors: [sortDescriptor]) { (_, results, error) in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }
        completionHandler(results?[0] as? HKQuantitySample)
        print("bfr start")
        for iter in 0..

```

**Figure 23: Code snippet for fetching weekly heart rate**

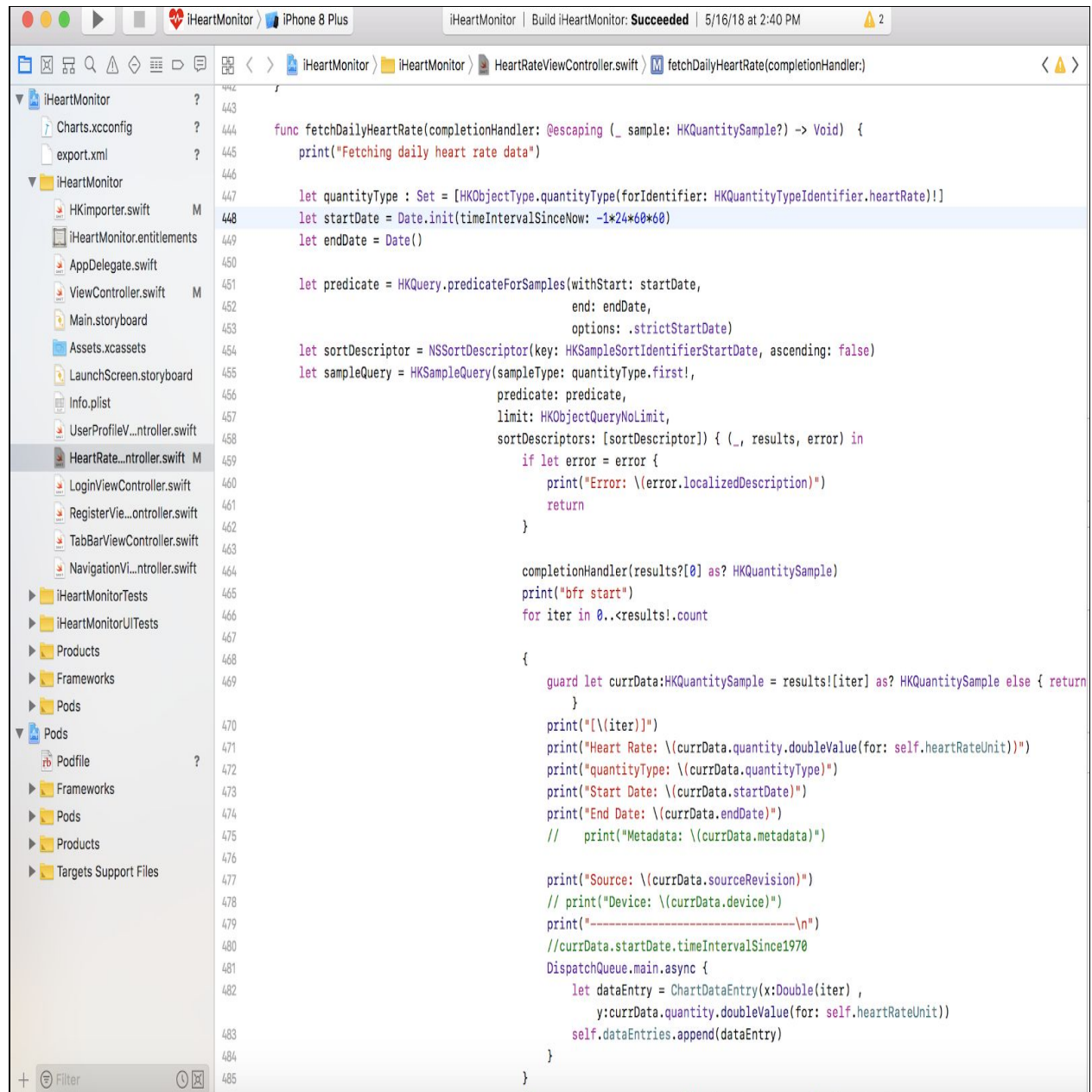


Figure 24: Code snippet for fetching daily heart rate

```

func fetchMonthlyHeartRate(completionHandler: @escaping (_ sample: HKQuantitySample?) -> Void) {
    print("Fetching monthly heart rate data")

    let quantityType : Set = [HKObjectType.quantityType(forIdentifier: HKQuantityTypeIdentifier.heartRate)!]

    //Fetch the last 30 days of HEARTRATE.

    let startDate = Date.init(timeIntervalSinceNow: -30*24*60*60)
    let endDate = Date()

    let predicate = HKQuery.predicateForSamples(withStart: startDate,
                                                end: endDate,
                                                options: .strictStartDate)
    let sortDescriptor = NSSortDescriptor(key: HKSampleSortIdentifierStartDate, ascending: false)
    let sampleQuery = HKSampleQuery(sampleType: quantityType.first!,
                                    predicate: predicate,
                                    limit: HKObjectQueryNoLimit,
                                    sortDescriptors: [sortDescriptor]) { (_, results, error) in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }

        completionHandler(results?[0] as? HKQuantitySample)
        print("bfr start")
        for iter in 0..

```

**Figure 25: Code snippet for fetching monthly heart rate**

The values range between 60 to 100 counts/ min. We represent the heart rate values in the form of line graph so that the end user can see how the heart rate has varied for the past day, week, month. The charts are added as cocoa pod dependency in Xcode. The following screenshot represents the heart rate graph plot for a month



**Figure 26: Heart rate line chart for the month**

### 9.3 Healthkit - Step Count

Using HKStatisticsQuery provided by Healthkit we are calculating the average number of step count per day for a specific time slot. An alert is generated if the number of steps are goes down below a certain limit. Figure\_ shows the code for step count.

```
//step function
func getTodaySteps(completion: @escaping (Double) -> Void) {

    let stepsQuantityType = HKQuantityType.quantityType(forIdentifier: .stepCount)!

    let now = Date()
    let startOfDay = Calendar.current.startOfDay(for: now)
    let predicate = HKQuery.predicateForSamples(withStart: startOfDay, end: now, options: .strictStartDate)

    let query = HKStatisticsQuery(quantityType: stepsQuantityType, quantitySamplePredicate: predicate,
    options: .cumulativeSum) { (_, result, error) in
        var resultCount = 0.0
        guard let result = result else {
            print("Failed to fetch steps rate")
            completion(resultCount)
            return
        }
        if let sum = result.sumQuantity() {
            resultCount = sum.doubleValue(for: HKUnit.count())
        }

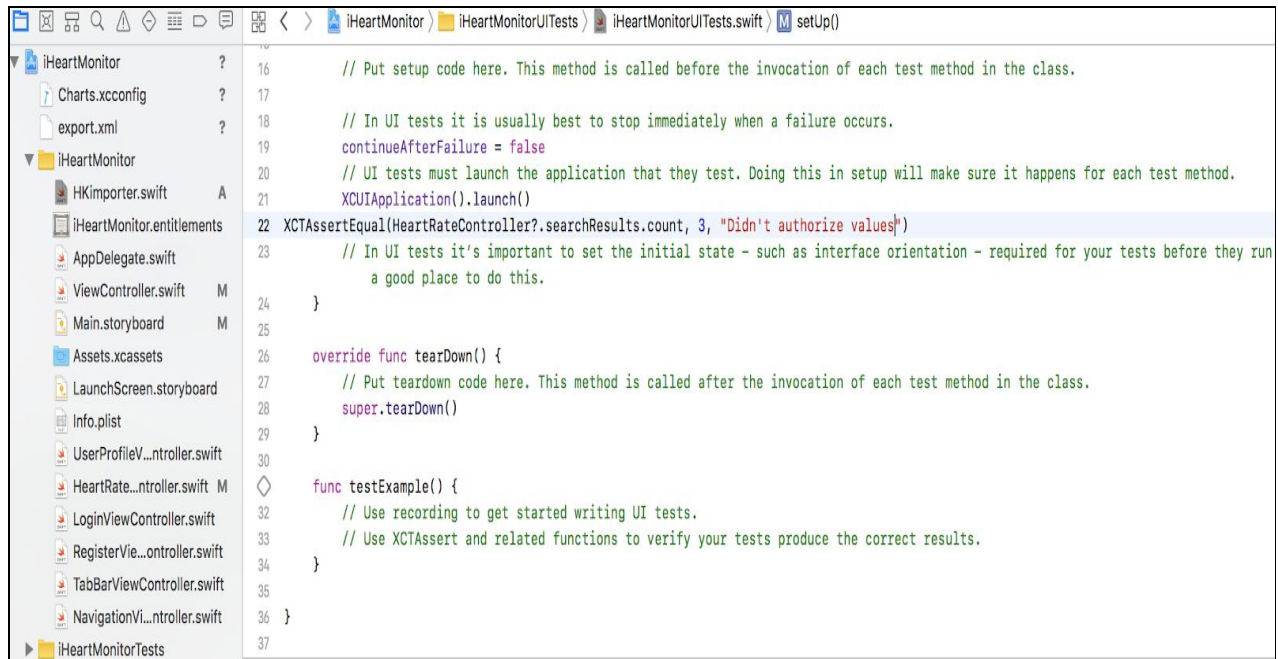
        DispatchQueue.main.async {
            completion(resultCount)
        }
    }
    healthStore.execute(query)
    //print("sucess")
}
```

**Figure 27: Code snippet for fetching daily step count**

## 10. TESTING

Xcode Unit testing framework helps in testing the code coverage by using Xcode Test Navigator . XCTAssert helps in testing of the main functionality of the code. The test failure breakpoint is used for finding the failure test cases. The following figure represents the code snippet for unit testing. Xcode allows to build UI test cases using recording rules. On clicking the record button in the bottom bar, the testing framework opens the app in which one can do a variety of actions on the app and the framework records code to do repetitive testing. An example looks as follows





**Figure 28: Code snippet for unit testing using Xcode Test Navigator**

```

func testExample() {
    let app = XCUIApplication()
    app.textFields["Enter username"].tap()

    let enterPasswordSecureTextField =
app.secureTextFields["Enter password"]
    enterPasswordSecureTextField.tap()
    app.buttons["Login"].tap()
    app.tabBars.buttons["HeartRate"].tap()
    enterPasswordSecureTextField.tap()
    // Use recording to get started writing UI tests.
    // Use XCTAssert and related functions to verify your
    tests produce the correct results.
}

```

**Figure 29: Code snippet for unit testing using UI testing**

As seen the test case has recorded actions for typing username, password and hitting login button.

## 11. DESIGN PATTERNS USED □

### 11.1 Storyboard

UI Storyboards are used to sketch the user interface of iHeartMonitor application. It provides a detail flow of the app in a white board. We have sketched the graphical elements like buttons, views, icons using libraries such as UIKit. By working on the storyboard, a lot of time is saved and helps to do the backend controller coding with less confusions. Storyboards make it easy for the app to run with less/no complexity. Using the Interface Builder component of Xcode, it is easy to drag and drop the UI components like navigation controllers on the whiteboard and design the user interface of each view in an independent manner. Xcode by default generates all the code necessary to implement the behavior of these UI components.

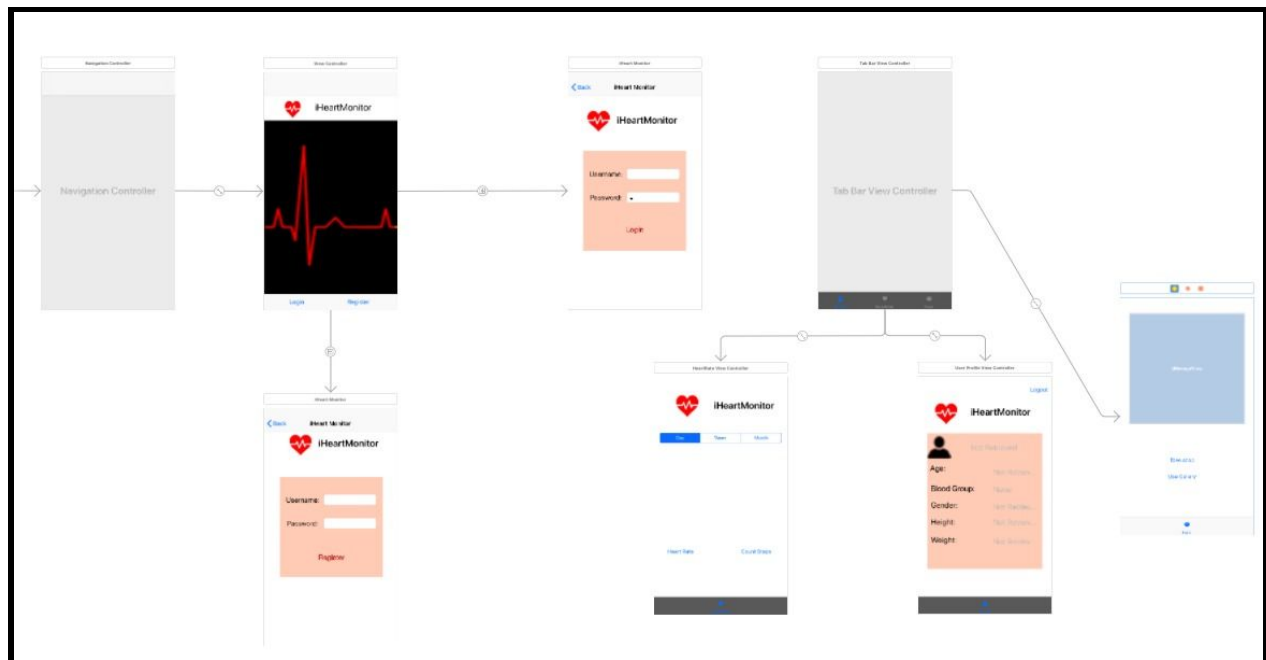


Figure 30: Application Storyboard



## 11.2 Navigation controller

As our application in more than one content view hierarchy, we decided to use **UINavigationController** that provides a navigation controller that manages the transition between series of view controllers backward and forward. A set of view controllers manage the navigation stack. Thus we never used segue to move backward over the storyboard. Using too many segue made it complex for the UI to understand the user events and system events. Navigation controller provides the necessary bar on the top of each page with a back icon. We can customize the icon.

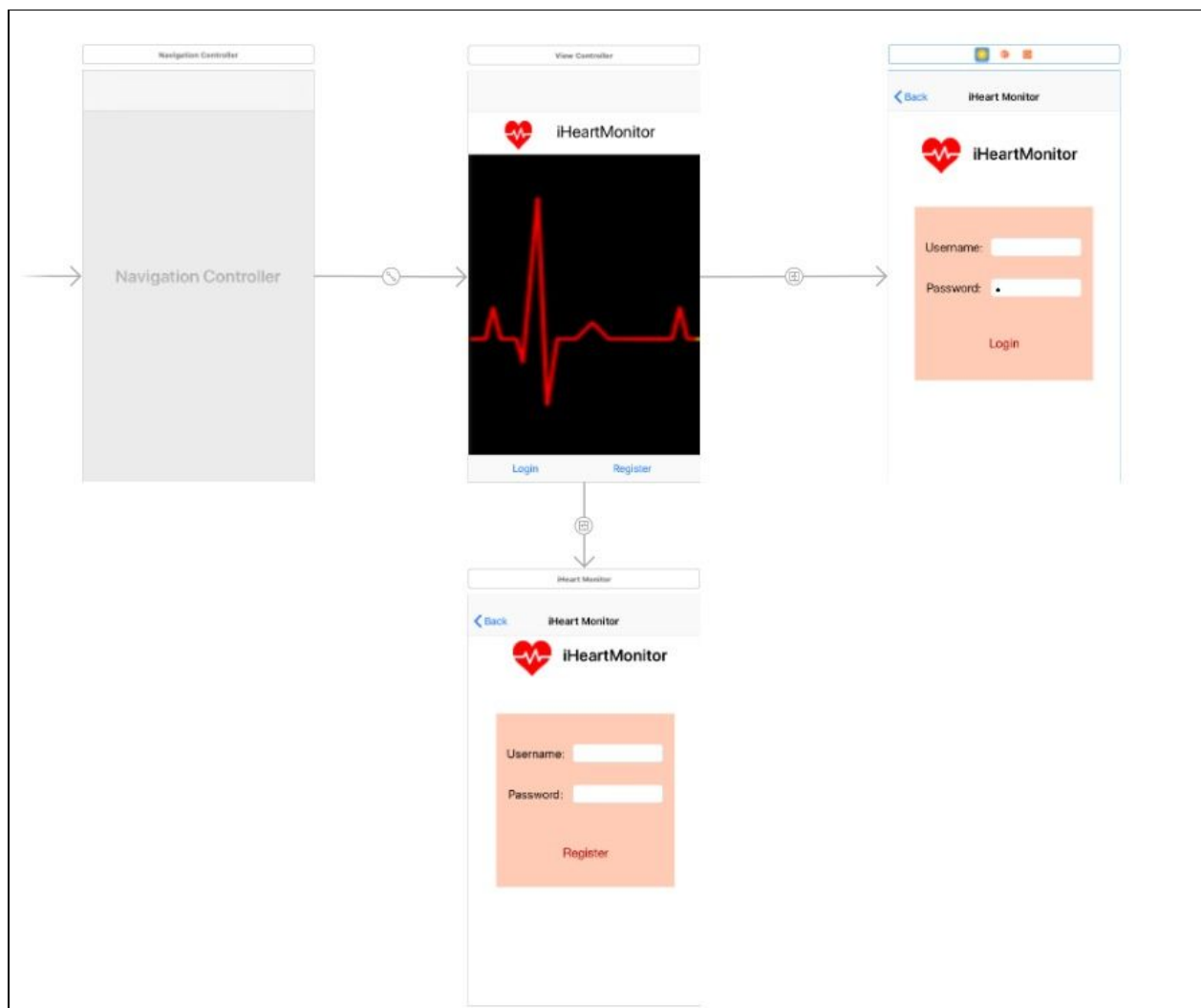


Figure 31: Navigation Controller flow

### 11.3 Tab Bar Controller

A tab bar controller helped us to divide our app into three sections with different modes of operations. It is an instance of `UITabBarController` and contains view controller. Thus selecting on one of the tab items, the user can easily navigate different views and relative view controllers.

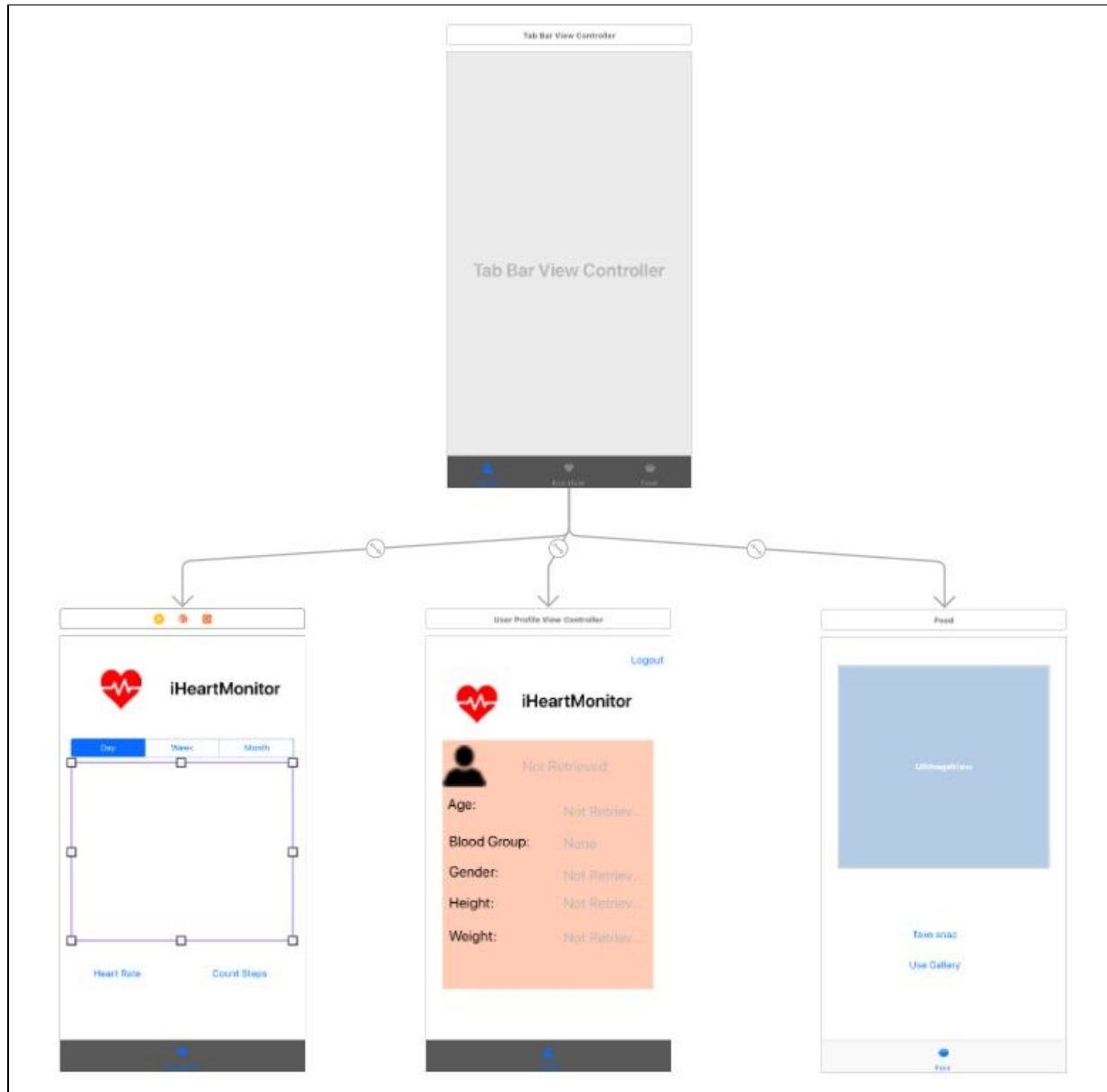


Figure 32: Tab View Controller flow

## 11.4 Segue

Segue is defined as the flow of the entire application. It is used to transfer from one view controller to another view controller. The starting point of a segue is always a button or a table row or a gesture recognizer. The end point is a view controller that controls the functionality of the display. In general a segue presents a view controller. **Unwind** segue can be used to dismiss a view controller. The **UIKit** loads the segue along with the view controller it is associated with and connects both of them to the corresponding trigger relation. Thus if we delete a segue, we need to manually delete the relationship of that segue with the view controller components.

The **UIStoryboardSegue** object is used to perform the transition between the view controllers. Segue objects are used to help the **UIStoryboardSegue** to perform the transition. **Segue objects** allow to pass data among the view controllers using **prepareForSegue: sender** method. **Actions** are piece of code that is linked to the event (any kind). We used **IBAction** to create and implement the method to perform the action on an event in our application. All the objects referred to the UI are controlled as outlets in the view controller using **IBOutlet** property.

There are several types of segue like push, modal, custom and Unwind. We have used push/show, popover and show details segue in this application.

## 11.5 Segmented control

A control in a user interface is an object such as slider, button or switch that helps the users to manipulate the contents and interact with the application. When the user interacts with a control, an event is created and respective response object is called. The events can happen by three ways: by touching the screen or dragging items on the screen, editing the events or changing the values related to the events.

## 11.6 View Controllers

The view controllers are used to manage the presentation of the application content on the screen. A view controller provides facilities to manage a single view and its collection of subviews. The `UIViewController` class is known as the base class for all the view controller objects. UIKit framework uses the view controller classes to implement the system interfaces like image picker, navigation interface and tab bar interface. The view and control objects provide the visual representation with familiar interfaces such as buttons, text fields, toggle switches and so on. The UIKit framework provides a custom views that layers the main view controller object. The responder chain begins with these inner views and navigates to the view controller to respond to the events triggered by these interfaces on the view. From the view controller the response object will be navigated to the window and then to the application framework. Thus on a button click, we call the method that calls the application to get the heart rate from the Healthkit library. Then we computed the necessary calculations in the view controller and responded with a graphical view on the UI.

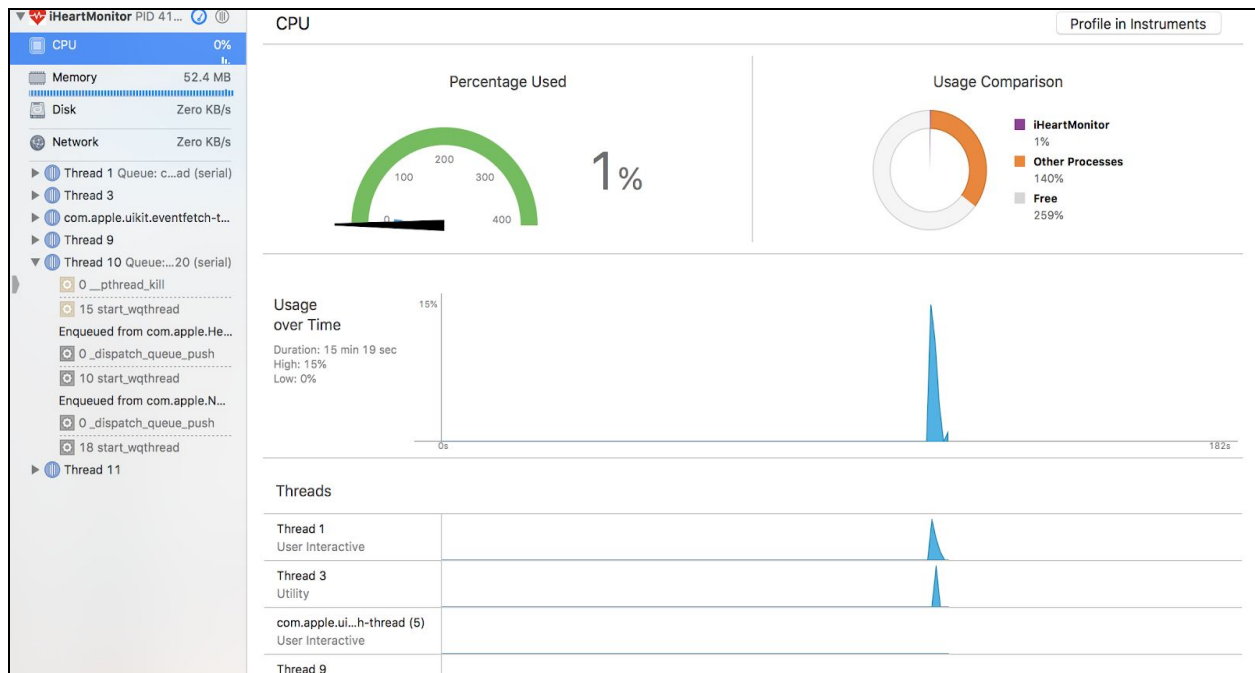
## 11.7 Charts

iOS has beautiful set of libraries like Charts which was developed by Daniel Gindi. Charts is an amazing feature that helped us to provide a clear graphical representation of the user heart rate. One of the essential need in iOS to install libraries is the library manager. CocoaPods are used to manage the iOS libraries. We installed the Chart library which has 8 different chart types. It can scale the axes and allows to zoom in and zoom out. We can also drag the lines to view highlighted values with predefined color templates. We used line chart with three x value ranges.

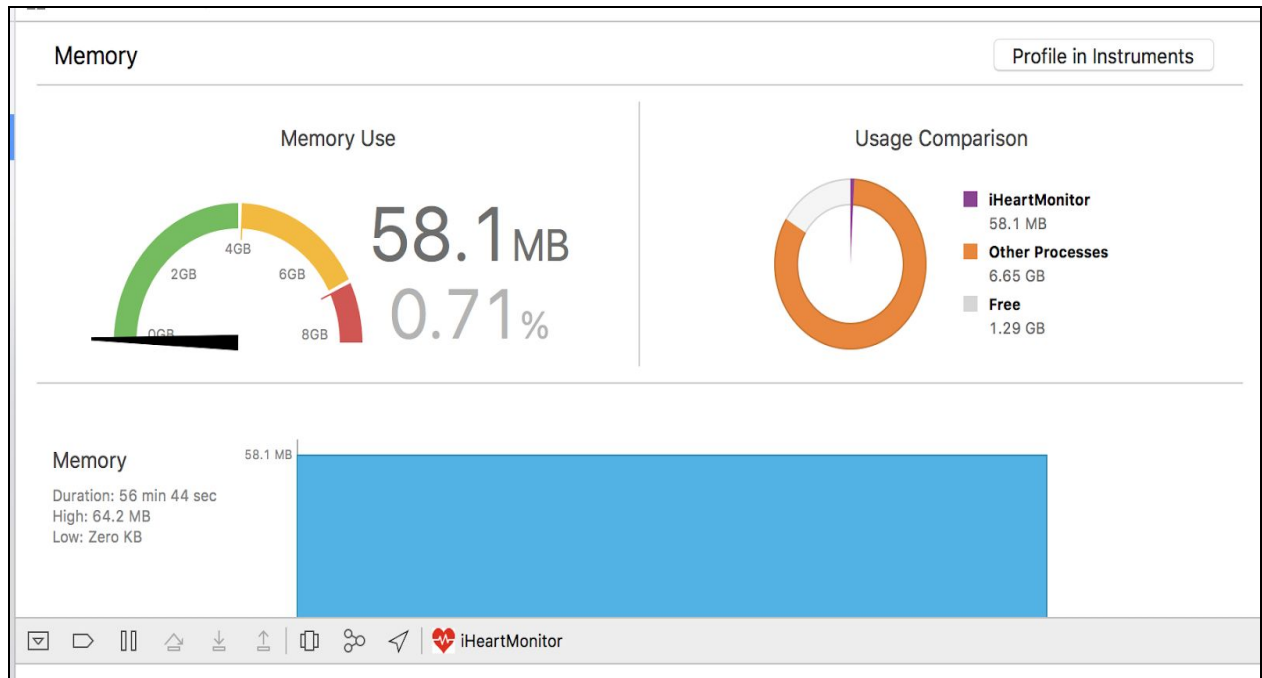
## 12. PROFILING

### 12.1 Xcode Profiler

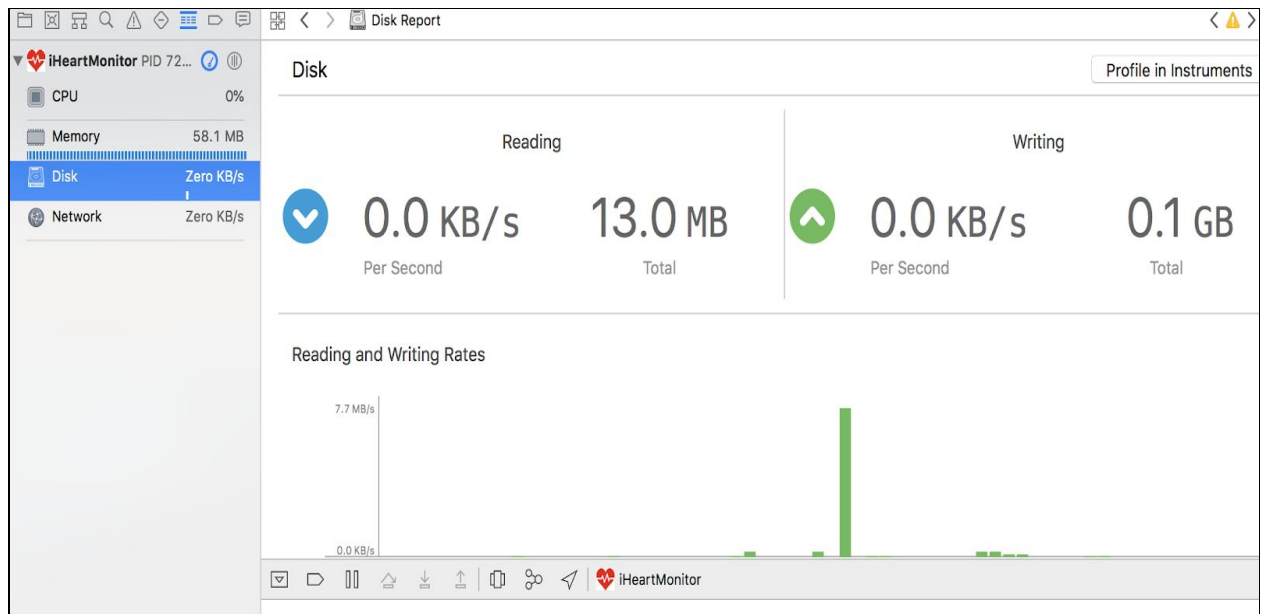
The Xcode provides a toolkit called Instruments for analysis of performance and testing. To improve the app performance and understand their behaviour with respect to memory, CPU, disk usage and network usage. We can find out the system resources consumed by the application from the instruments profiler and plan to reduce the resources. The below screenshots represent the CPU ,Memory, Disk and Network profiling information.



**Figure 33: CPU usage in Xcode Instrument profiler**



**Figure 34: Memory usage in Xcode Instrument profiler**



**Figure 35: Disk usage in Xcode Instrument profiler**

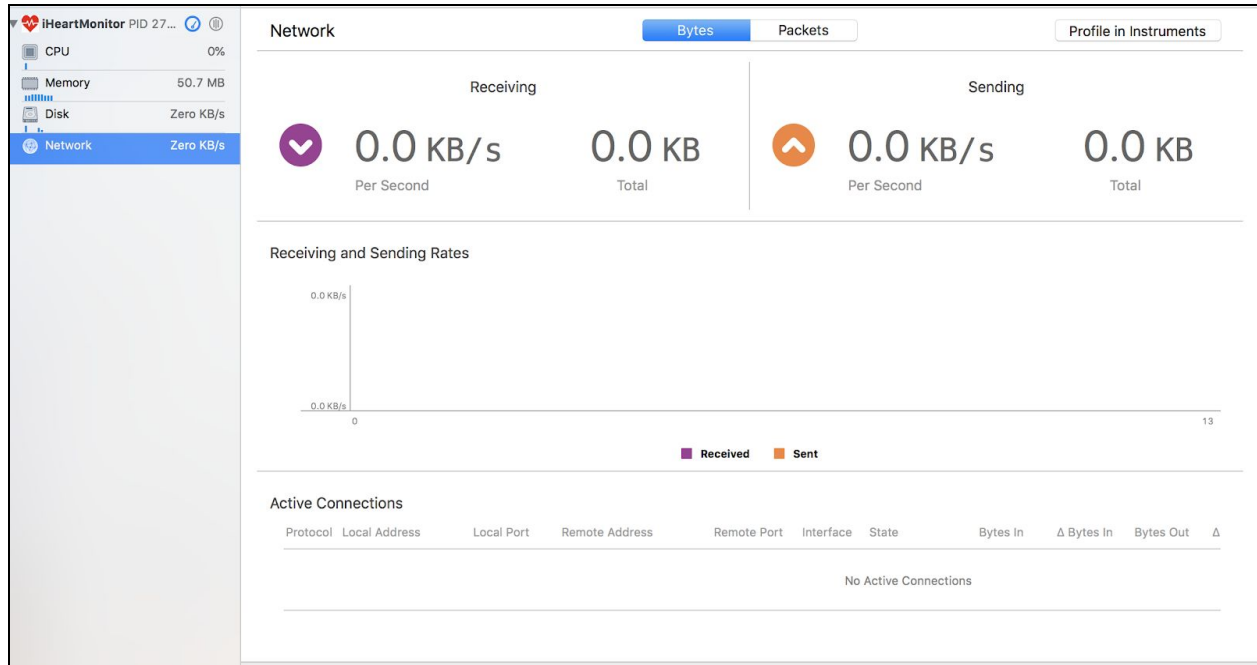


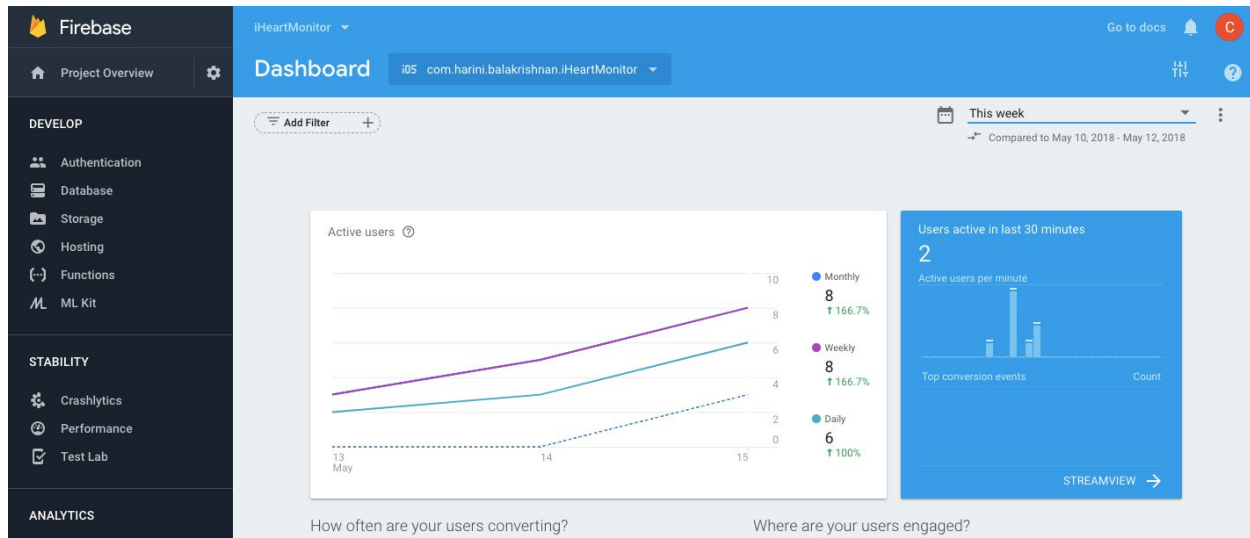
Figure 36: Network usage in Xcode Instrument profiler

## 12.2 Firebase Profiling

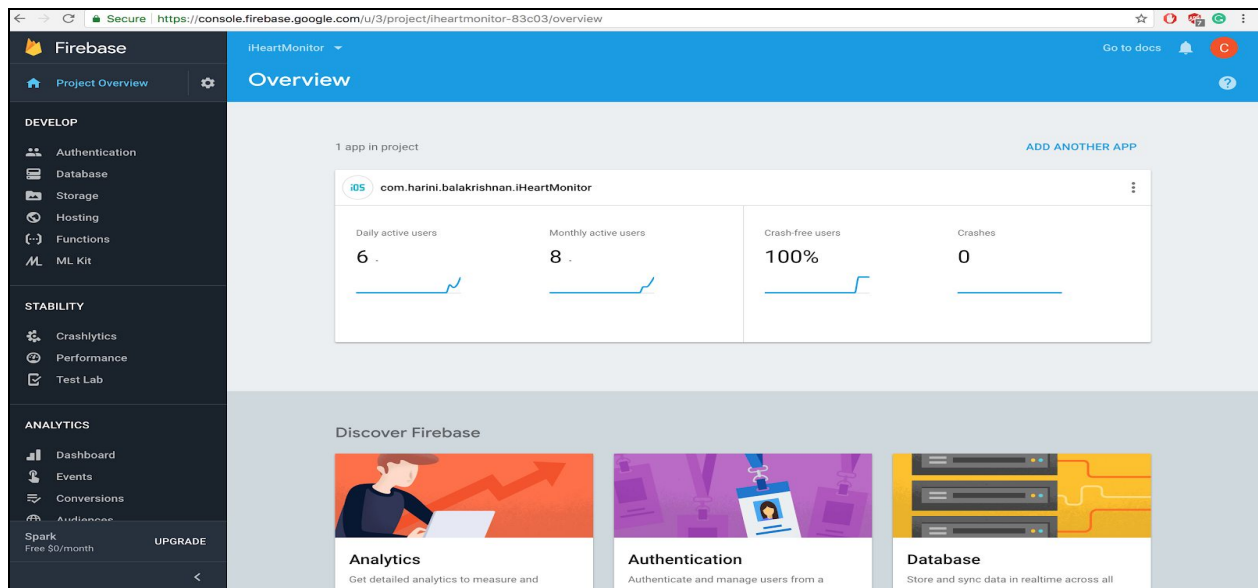
The screenshot shows the Firebase console's Authentication section. A table lists user sign-in details. The table has columns for Identifier, Providers, Created, Signed in, and User UID. There are 7 users listed.

Identifier	Providers	Created	Signed in	User UID
test2@email.com	Google	May 13, 2018	May 13, 2018	CzmpvZJHurXzfrgh059NcvOuNd42
vjsamuel1990@gmail.com	Google	May 15, 2018	May 16, 2018	LDKQV5ROJJJeAv57qMyYwAE4J...
harini1593@gmail.com	Google	May 16, 2018	May 16, 2018	V70siXfz3bWmgOBUMlmMRjThd...
harini.balakrishnan@sjsu.edu	Google	May 16, 2018	May 16, 2018	Yqx2SQX4kjRtY1MEIbc5zIByMbv1
test@email.com	Google	May 12, 2018	May 16, 2018	Zhw8wNThfCXvf1PPVpqw6f2Nbiq1
test11@email.com	Google	May 14, 2018	May 16, 2018	dfSRKgeUyuNDKCxLmCM7N6q9...

Figure 37: App users sign-in details on firebase console



**Figure 38: Active users monitoring on firebase**



**Figure 39: Users activity and App crash monitoring on firebase console**



## **13. Future work**

We are planning to extend our application functionality by adding machine learning to it. We plan to build a heart rate prediction model using tensorflow and detect deviations in heart rate. We also want to add a module that send notification to the doctor when it detects abnormality in heart rate. To augment the user's interest , a feature to chat and share user's activity to friends will be added.

## 14. References

- [1] M. Allan, "A Beginner's Walk Through HealthKit," 5 October 2016. [Online]. Available: <https://medium.com/@missyalienn/a-walk-through-healthkit-10434d33ff87>.
- [2] Apple Inc., "Apple Developer," [Online]. Available: <https://developer.apple.com/documentation/healthkit/hkstatisticsquery>.
- [3] C. Esplin, "What is Firebase?," 24 October 2016. [Online]. Available: <https://howtofirebase.com/what-is-firebase-fcb8614ba442>.
- [4] Firebase, "Firebase," 2018 May 2018. [Online]. Available: <https://firebase.google.com/docs/ios/setup>.
- [5] Instruments User Guide: About Instruments. (2016, September 13). Retrieved from <https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/index.html>
- [6] HealthKit | Apple Developer Documentation. (2018). Retrieved from <https://developer.apple.com/documentation/healthkit>
- [7] Your heart rate. What it means, and where on Apple Watch you'll find it. (2018). Retrieved from <https://support.apple.com/en-us/HT204666>
- [8] Recognize Text in Images with ML Kit on iOS | Firebase. (2018). Retrieved from <https://firebase.google.com/docs/ml-kit/ios/recognize-text>

## TEAM MEMBER CONTRIBUTION

NAME	FUNCTIONALITY WORKED
<b>Harini Balakrishnan</b>	Storyboard UI development, Healthkit integration, Charts, Documentation
<b>Vidhi Sharma</b>	Firestore authentication, Step count , Documentation
<b>Vijay Samuel</b>	Firestore ML Kit, Apple Watch Configuration, Documentation
<b>Yamini Muralidharen</b>	Heart Rate Collection, Charts, Testing, Documentation