

Physao Companion App

Hao Li, Weiqi Wei, Emmanuel Shiferaw

1.Introduction

1.1 Physao

Physao, which is a DUhatch startup, has created a portable spirometer which collects data on a user's respiratory capacity. Users blow into the device, which advertises results via BlueTooth.

1.2 Desired application

Companion App needed with device for storage and display of test results. It also has simple UI/animation for tests. Meanwhile, the graph will show the historical data collected for health tracking. The Physao team also wants us to design some parts to motivate users to do the blow tests. So we developed the social part to fulfill their needs.

2. Data model

We use three data stores for our different features: SQLite, HealthKit, and Parse. The data like the basic information of users and historical data are saved in SQLite database. In the social part, the information of users are loaded from and saved to Parse.com.

2.1 Data from the SQLite database

There are three tables in this database: userTable, friendsTable and dataTable1.

In the userTable, the userName and password are saved. All of friends' info from the social part are saved in friendsTable. The historical data are all saved in dataTable1. If you want to see all of the historical data, you need to log in with UserName: **Weiqi Wei**, Password: **11**. If you create a new

account and log in, you will see an empty graph. You can see the design of DataTable1 in Figure 2.3 to get more details.

TABLE	userTable	Search	Show All	Add	Duplicate	Edit	Delete
rowid	username	password					
3	Hao Li	11					
4	WeiQi Wei	11					

Figure 2.1 userTable

TABLE	friendsTable	Search	Show All	Add	Duplicate	Edit	Delete
id	hostName	friendName	times	untilDate			
0	WeiQi Wei	Hao Li	1	11/29/2015			
1	WeiQi Wei	Yebin Han	2	11/24/2015			
2	Hao Li	WeiQi Wei	3	11/27/2015			
3	Hao Li	Mingming Lu	4	11/28/2015			

Figure 2.2 friendsTable

TABLE	DataTable1	Search	Show All	Add	Duplicate	Edit	Delete
rowid	date	fvc	fev1	userName			
1	11/30/2015	10	10	WeiQi Wei			
2	12/1/2015	20	20	WeiQi Wei			
3	12/2/2015	19	19	WeiQi Wei			
4	12/3/2015	20	29	WeiQi Wei			
5	12/4/2015	20	27	WeiQi Wei			
6	12/5/2015	38	27	WeiQi Wei			
7	12/6/2015	34	25	WeiQi Wei			

Figure 2.3 DataTable1

2.2 Data in Parse.com

There are two classes which are equal to tables in databases in this cloud. The first one is called friendsInfo, which has three main columns called name, times and untilDate.

<input type="checkbox"/>	objectId String	name String	times String	untilDate String
<input type="checkbox"/>	ULrWYRVrCZ	Emmanuel	0	12/01/2015
<input type="checkbox"/>	4zBnnWlics	WeiQiww	0	12/01/2015
<input type="checkbox"/>	4c3iVZ4VPQ	WeiQiwww	0	11/29/2015
<input type="checkbox"/>	900gMKIgKI	Wenwen	0	11/29/2015
<input type="checkbox"/>	9QV9KB3sqS	WeiQiiff	0	11/29/2015
<input type="checkbox"/>	9Uav0IDYLZ	Hehe	0	11/29/2015
<input type="checkbox"/>	Fh6qcojoEq	Haha	0	11/29/2015
<input type="checkbox"/>	98VgcYcqD4	Wenchen Ning	5	11/29/2015
<input type="checkbox"/>	M9lxdQyXMs	Shuai Fu	4	11/28/2015
<input type="checkbox"/>	H5wfEkNaFE	Mingming Lu	3	11/27/2015
<input type="checkbox"/>	NYpKGSzQzr	Yebin Han	2	11/26/2015
<input type="checkbox"/>	gGKqoYH0xk	Guoshan Liu	1	11/25/2015
<input type="checkbox"/>	PynpgCc3iY	WeiQi Wei	0	11/24/2015
<input type="checkbox"/>	2gIwnuj6hI	Hao Li	3	11/26/2015

Figure 2.4 friendInfo class


Another class is called `InvitationObject`. All of the invitations from users are saved in this class. This class has columns: `hasAdded`, `nameFrom`, `nameTo`, `timesFrom`, `untilDateFrom`, `timesTo`, `untilDateTo`. The reason to save last 4 column is to reduce the time to search `friendInfo` class when adding new friends. The `hasAdded` column is to determine whether the invitation has been confirmed. If confirmed, it will be changed to 1. Otherwise, it will be 0.

<input type="checkbox"/>	objectId String	hasAdded String	nameFrom String	nameTo String	timesFrom String	untilDateFrom String	timesTo String	untilDateTo String
<input type="checkbox"/>	0rq1DgFI9y	0	Weiqi Wei	Hao Li	0	12/02/2015	3	11/26/2015
<input type="checkbox"/>	1Lv1GXuW2d	0	Weiqiww	Hao Li	3	11/26/2015	3	11/27/2015
<input type="checkbox"/>	sGBX3dI6ai	0	Weiqi Wei	Shuai Fu	4	11/28/2015	4	11/25/2015
<input type="checkbox"/>	R9fobpVSTa	0	Weiqi Wei	Weiqiff	0	11/29/2015	2	11/27/2015
<input type="checkbox"/>	MsF8qyJ2uU	0	Wenwen	Weiqiff	0	11/29/2015	4	11/28/2015
<input type="checkbox"/>	0as7hgJo1i	0	Weiqi Wei	Mingming Lu	0	11/24/2015	2	11/24/2015
<input type="checkbox"/>	kpbNgTAq6G	0	Weiqi Wei	Wenchen Ning	5	11/29/2015	6	11/30/2015
<input type="checkbox"/>	RZimYoA4qp	0	Weiqi Wei	Yebin Han	5	11/29/2015	5	11/29/2015
<input type="checkbox"/>	RYNboY33oV	0	Shuai Fu	Hao Li	5	11/29/2015	1	11/22/2015

Figure 2.5 `InvitationObject`

2.3 HealthKit

For the actual test results collected from the device, Physao uses Apple's

HealthKit . In the HealthKit API, singular entries of any value measured are known as “Samples” (abstract class `HKSample`). These can be things like steps (pedometer), vital signs, fitness/workout tracking events, and in the case of Physao, spirometer results. The specific subclass of `HKSample` we use, called `HKQuantitySample`, consists of a ‘quantity’ (abstract class `HKQuantity`) of a certain ‘type’ (abstract class `HKQuantityType`) as well as a timestamp (consisting of a start date and end date). The metrics we collect from the Physao spirometer (FEV, FVC1, and PEFR) are all found in the HealthKit library as built in data types.

These are constants included in the HealthKit API which can be used to generate the necessary/relevant subclasses of `HKQuantityType`.

```
(FVC) NSString * const HKQuantityTypeIdentifierForcedVitalCapacity;  
(FEV1) NSString * const HKQuantityTypeIdentifierForcedExpiratoryVolume1;  
(PEFR) NSString * const HKQuantityTypeIdentifierPeakExpiratoryFlowRate;
```

As such, upon reading data from the device and extracting the relevant values, the process for saving the data essentially involves building an HKSample object for each metric (with the above identifiers) with the collected value and current time as the start/end date.

Physao has one class that encapsulates all the information for a single query of a *single data type* (FVC, FEV, PEFR, Ratio, communicated through enum type `PhysaoSampleType`). This is called `PhysaoQuery`, with constructor:

```
init(start:NSDate, end:NSDate, typeToReturn: PhysaoSampleType)
```

`PhysaoQuery` returns a list of `PhysaoDataPoint` objects that correspond to the individual samples. `PhysaoDataPoint` is a simple wrapper for a single point to be graphed simply by the graphing portion of this application.

3. Login Part and Dashboard

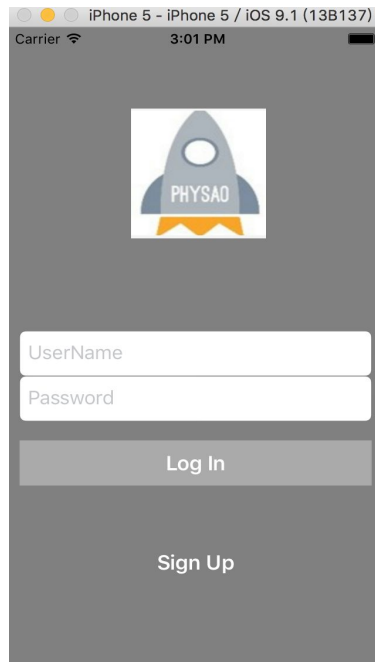


Figure 3.1 Log in view

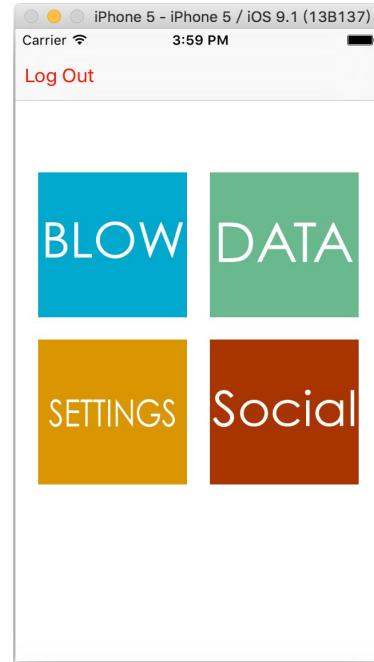


Figure 3.2 Dashboard

The first part of our app is the login viewcontroller. If you are a new user, you can input your userName and password, then press sign up. Your information will be stored in the SQLite database.

If you are not a new user, you can type in your userName and password, then press login button. This app will check whether the pair of username and password is correct by comparing the data in SQLite database. Then you will see the dashboard viewcontroller.

4. Settings Part

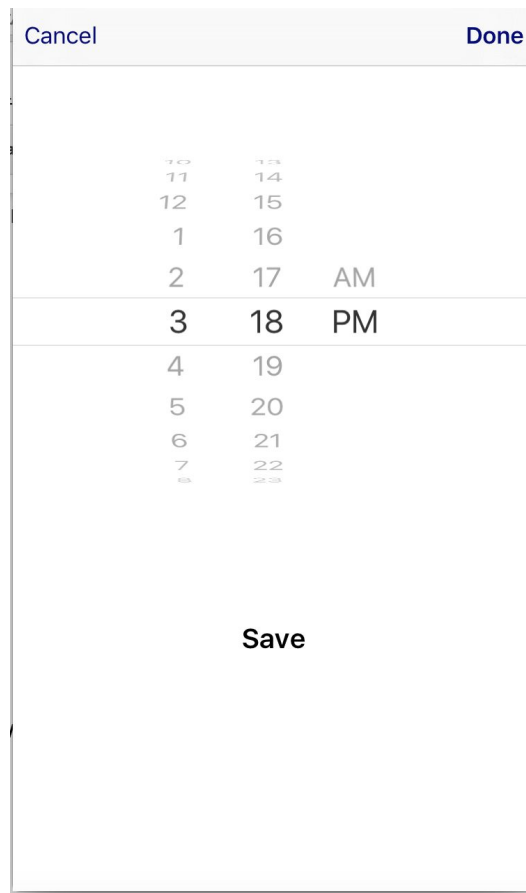


Figure 4.1 Settings Part

This part is mainly about creating notifications to remind the user to use the medical device on time. Basically, the user needs to use the device everyday, so we set the mode of `UIDatePicker` to “time” mode, which only shows the hours and minutes.

```
notifyPicker.datePickerMode = UIDatePickerMode.Time
```

After the user selects the time, the “Save” button needs to be pushed. In order to let the user know that the notification is created successfully, we built a `UILabel` part which fades in and then fades out to tell the user that the notification has been created successfully. Once we push the “Save”

button, it will register and create the notification. We have the code both in AppDelegate.swift and SettingsViewController.swift. In case the notifications we have set before still work, which is annoying. Everytime we set a new notification, we do delete all the notifications we've registered before.

```
UIApplication.sharedApplication().cancelAllLocalNotifications()
```

5. Blow Part

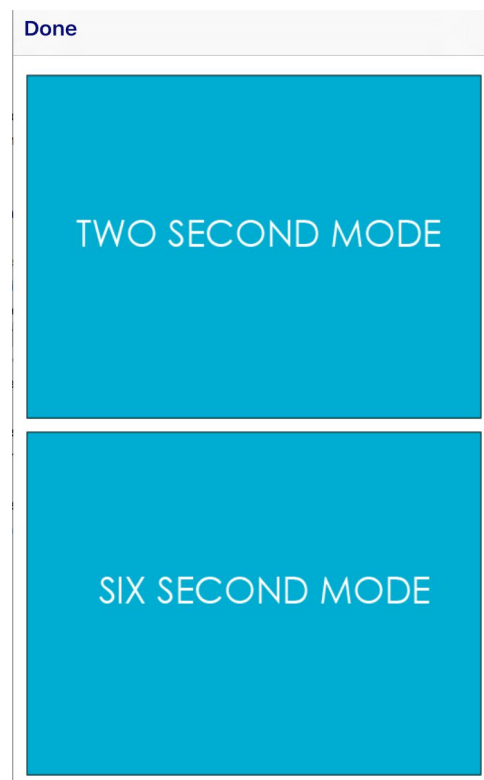


Figure 5.1 Mode Choose View

Once we push the “Blow” button in the Dashboard View, we go to this Mode Choose View first. Basically we have two different modes: two sec mode and six sec mode. We can swipe down the two sec mode button and

swipe up the six sec mode button to go to each mode.

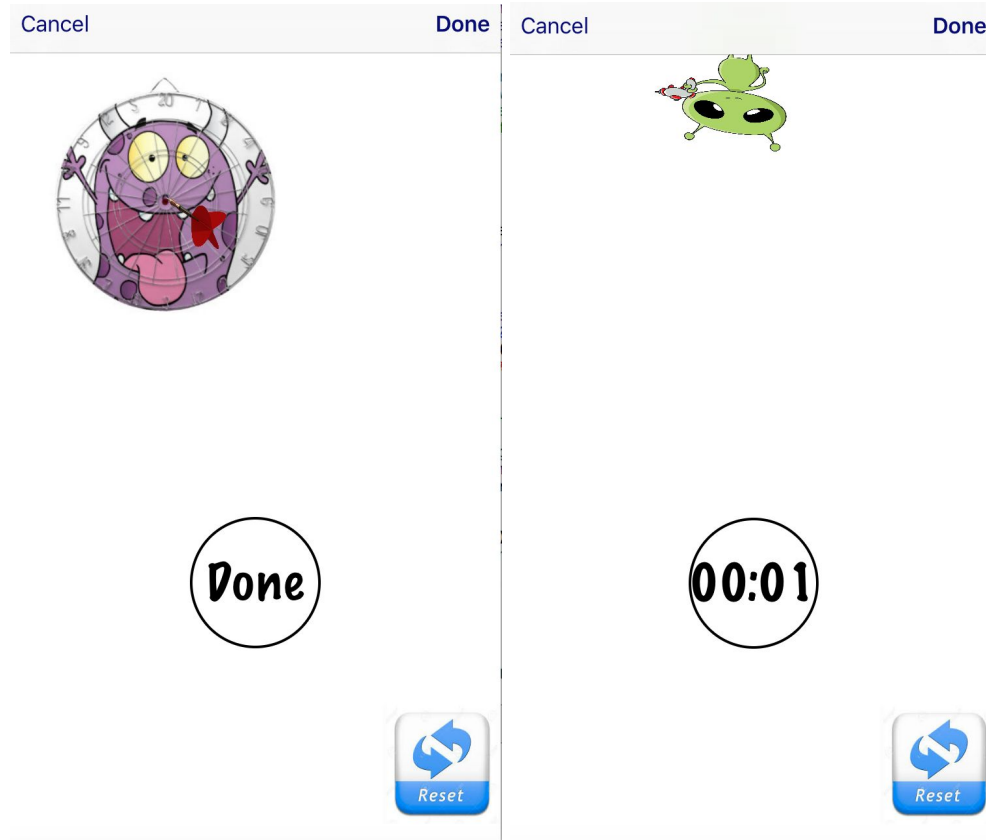


Figure 5.2 two sec mode

Figure 5.3 six sec mode

In these two modes, we both built a countdown timer, one is two second, and the other is six second. It shows clearly how long the user needs to use the device. If the user feels that he/she doesn't use the device for enough time according to the timer, he/she can do that again.

And we've also implemented different animations in these two modes. According to Physao's demands, we've implemented shooting dartboard and blowing alien away in two sec mode and six sec mode respectively. During the animation, the size of dart and alien both shrink.

For now, we can trigger both the animation and the countdown timer using

the reset button. As we cannot have the full access to the medical device from the Physao team. But we do leave the interface about triggering the animation and countdown timer to them.

```
func startAnimation() {}
```

```
func startTimer() {}
```

```
func resetTimer() {}
```

They can do whatever they like to trigger the animation and the countdown timer. The general idea is that when user blows to the device, the device will transmit the data to the phone via bluetooth. Once the app starts receiving the data, the animation and the countdown timer will be triggered.

In func peripheral(peripheral: CBPeripheral, didUpdateValueForCharacteristic characteristic: CBCharacteristic, error: NSError?) {}, we can trigger the animation and the timer once the app starts receiving the data.

That delay can be ignored due to the speed of connecting to the phone and transmitting data. In that case, the animation, countdown timer and user blowing to the device can happen simultaneously. And then the reset button can only reset the display of the countdown timer.

About the bluetooth part, according to the information and the bluetooth module from the Physao team (and thanks Physao for that), we connect the bluetooth module using the module's unique id,

```
service.UUID == CBUUID(string: "FFE0")
```

```
characteristic.UUID == CBUUID(string: "FFE1")
```

and also successfully receive the random string from that module. And we strongly believe that after the Physao team finish the calculation part

themselves, the app can have reasonable data from their medical device.

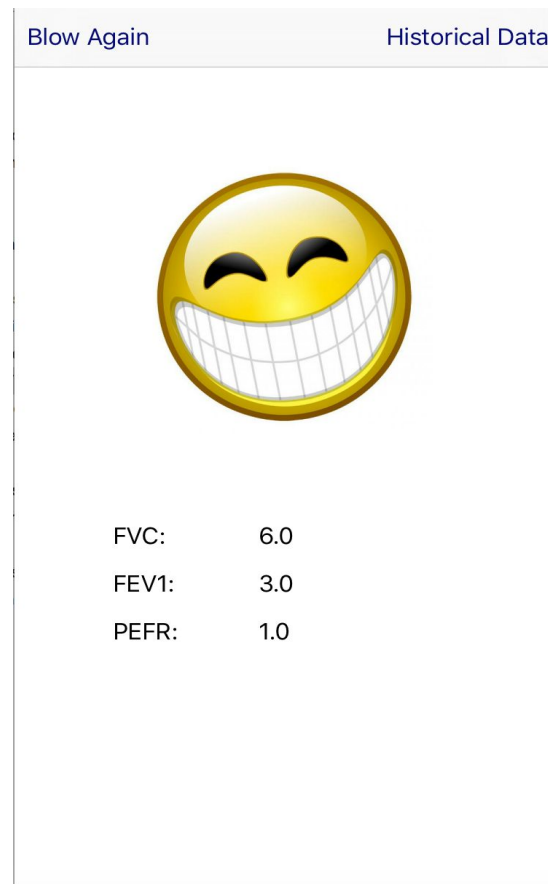


Figure 5.4 Immediate Feedback View

We've also built an immediate feedback view after user finishes blowing to the device. We've set three different kinds of faces: smile face, neutral face, and the sad face, based on their suggestion. For now, in the immediate feedback view, we show the smile face by default. And we also leave the interface to them, we've already finished the data transmitting part from the two sec and six sec mode to the feedback view. After they finish the calculation part, it will show different faces in the feedback view based on their criteria. And for the clearness, we chose to display the three important values they've pointed out below the image.

6. Graph Part

The graph in the right is the historical graph to show the user's data. It can show two kinds of data: FVC and FEV1. There are three types of dates. Days means to shows data in last 7 days. Weeks will shows data in last 3 weeks. Months will present historical data in last 3 months.

The whole process is below:

Data from database -> save in [String: userData] -> make the arrays of dates -> get the corresponding data -> draw graph

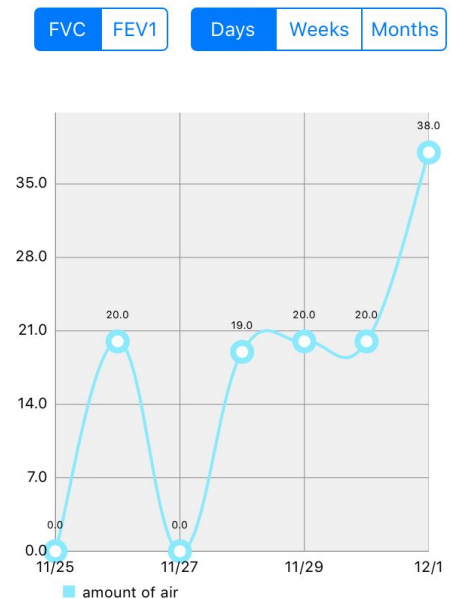


Figure 6.1 Graph

6.1 Get data from database

First, in HistoryDataViewController.swift, the app loads all of the historical data from SQLite database.

I designed a class called userData, which has 4 variables: userName, saveDate, FVC and FEV1. I use the function below to load data from database. The return value is a dictionary type. The key is the date and the value is the userData type.

```
func getUserHistoryData(userName: String)->([String: userData])
```

6.2 Calculate three x-value arrays

Then I need to get the accurate data in last 7 days, last 21 days and last 3 months. In order to draw the graph, I need to make x values in one array

and make y values into another array.

I design three modes: 0, 1, 2. Mode 0 means Days. Mode 1 means Weeks. Mode 2 means Months. When the user press the Button Days, Weeks and Months. The data will be gotten and draw the graph.

The most hard part is how to find the array of dates, which type is string DD/MM. The original type is DD/MM/YYYY. I need to change the values to the type DD/MM first. Then calculate the range of 7 days, 21 days and 3 months and save them into arrays.

6.3 Get the corresponding data and draw graph

Now I have gotten the dates arrays. Because the dictionary type [String: userData], the key is the date. I can input a date and then the value will be the user data. So I can find all of data using the data array I've got and used these data to draw graph.

I use the third-party library called IOS-Charts. Below is the link of this library.

<https://github.com/danielgindi/ios-charts>

7. The social part

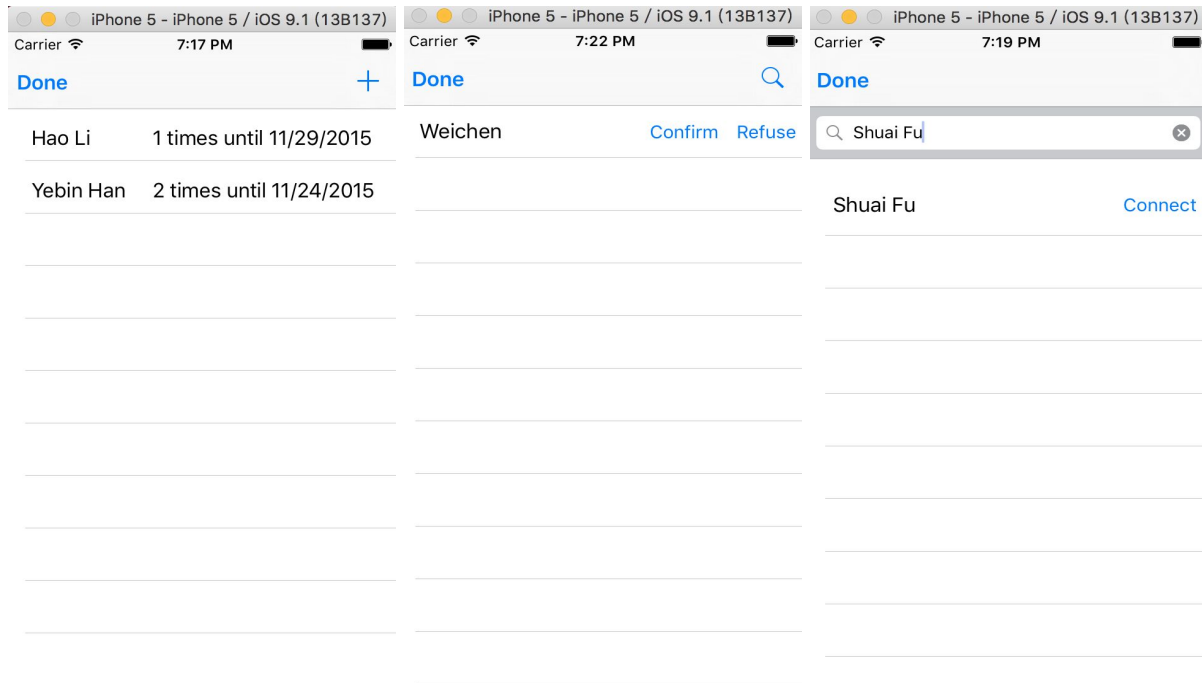


Figure 7.1 Friends list

Figure 7.2 Invitations list

Figure 7.3 Search friends

7.1 Introduction

In this social part, the user can see how often his/her friends use their medical device. The friends list view shows their friends' name and how many times they have used their devices. The invitation list view presents all of the invitation this user have got and the user can choose to confirm the invitation or refuse it. In the add friends view, the user can search new friends and connect with them.

7.2 Friends list

The friends data are saved in SQLite database. Each time the app loads this view, the data will be gotten from database and shown in the table view. If the user accepts the invitation in the invitations list ViewController,

the new friend will be added to this table view.

7.3 Invitation list

Each time the app loads this viewcontroller, it will get data from Parse.com to see whether there are some invitations there. If there are some invitations, it will get them and shows in this table view. If the user confirms the invitation, it will change one column called hasAdded in Parse.com. So the sender will know the invitation has been accepted. After accept, the friends info will be presented in friends list.

7.4 Search friends

If the user types a name in the search bar at the top of this viewcontroller and press search button, it will search on Parse.com to find whether there is a person called that name. However, if the user types an uncompleted name or a name starting with a lowercase letter, it will also search the name containing these letters and present them on this table view.