THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

COMP5527 Mobile Computing and Data Management

Group Project Report

Group Members:

QING Pei	11500811g
Mei Youzhi	11521683g
SHAO Shuai	11552402g
RAN Jun	09694163g

目录

Overview

In this project, a Mobile Healthcare (MH) application is created.

Required features include:

- Cross-platform.
- Mobile access to healthcare information.
- Mobile management of healthcare data.
- Synchronization between client and server.
- Mobile healthcare services.

What we achieved is an app that can

- run on iOS, Android, Blackberry, Windows Phone and Symbian platforms with the default browser
  (or packaged as an installable native application)
- view and update healthcare data stored on the server
- view or make appointments with doctors
- work in a disconnected environment with cached data

System Design

Client Design

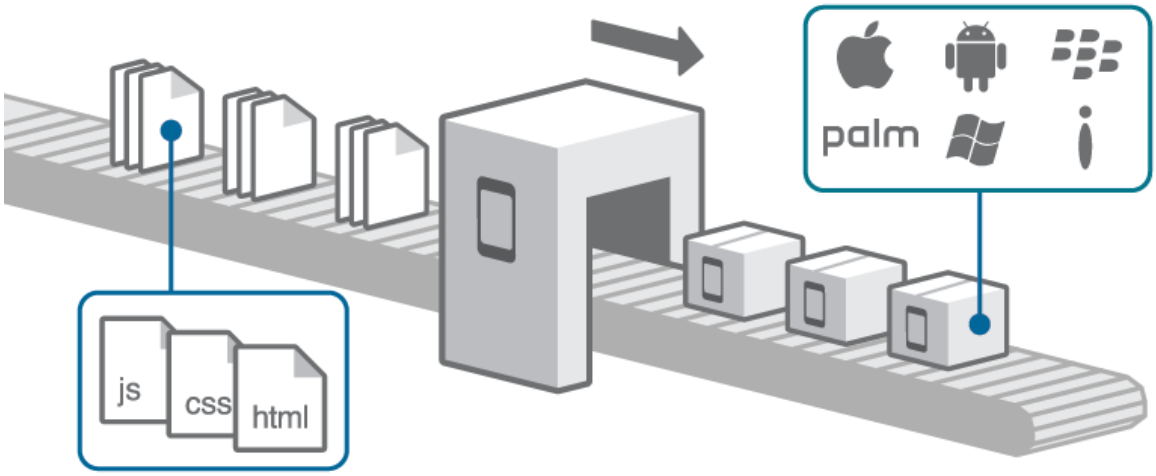
The client is built with HTML5 and javascript. The following frameworks/tools are used.

- PhoneGap
- jQTouch
- Lawnchair

Application Framework

To build a cross-platform app, we adopted PhoneGap framework.

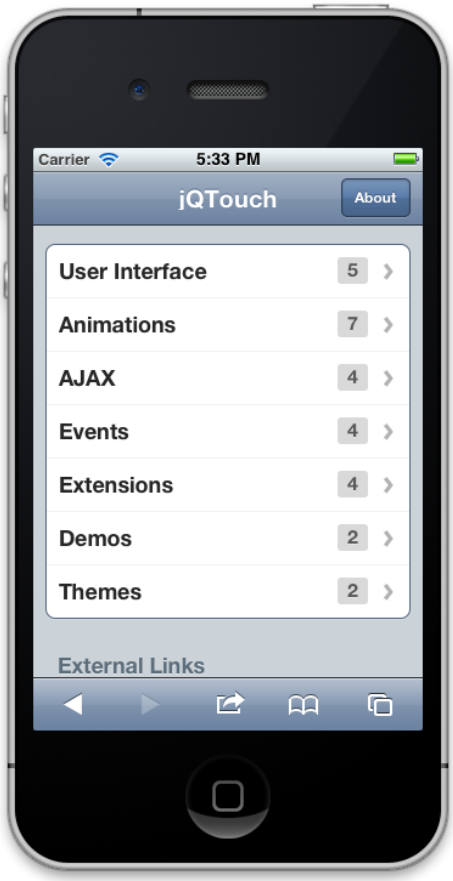
It is an HTML5 app platform that allows us to author native applications with web technologies and get access to APIs and app stores.



The application framework enables us to build the app once with web-standards using HTML5 and javascript, wrap it with PhoneGap and deploy to multiple platforms.

User Interface

Our client app provide a user interface built upon jQTouch. It is a Zepto/jQuery plugin for mobile web development on the iPhone, Android, iPod Touch, and other forward-thinking devices.



With jQTouch, the app UI is a stack of pages. By forward/backward navigation, we push/pop views from the stack. Each view is a element with an unique id attribute. Jumping to another view is done by rendering the correspondent part of the HTML body.

Offline Storage

Lawnchair is used for offline storage. Lawnchair is a lightweight, adaptive, simple and elegant persistence solution perfect for html5 mobile apps. It has the following features.

- super micro tiny storage without the nasty sql: pure and delicious JSON.
- default build weighs in at 3.4k minified; 1.5 gzip'd!
- adaptors for any clientside store.
- designed with mobile in mind.
- clean and simple API.
- key/value store ...key is optional.
- terse syntax for searching/finding.
- battle tested in app stores and on the open mobile web.
- framework agnostic. (if not a framework atheist!)
- MIT licensed.

By default, Lawnchair uses HTML5 localStorage (often called DOM storage). In our application, we will test whether the platform supports indexed-db which is said to be 3 times faster. If so, we use indexed-db, otherwise, localStorage is always a fallback choice.

Application Logic

Patient information management

This function is used for supporting Mobile Access to Healthcare Information (MAHI) and Mobile Healthcare Data Management (MHDM). Depend on user's type (doctor/patient), client side show different contents. For doctor, all patients' name list will appear, and he can notify patient's detail information and all medical records between him and each patients. For patient, his basic information and all medical records can be checked.

Appointment management

This function is used for supporting Mobile Healthcare Service (MHS). For a user as a patient, he can make an appointment with any doctor. Moreover he can update or delete any booked appointments under available situation.

Cache data management

This function is used for supporting mobile computing environment. One basic strategy is running a repeated thread to check any failed data requests existing on cache database. A request list table is used to store any failed user operations' data on cache DB. When client system cannot connect to server side, it will store any user request data to this request list table on cache DB. Another strategy is storing any transmitting data on cache DB. Once failed to connect server side, client system will check any existing data on cache DB and show them to user.

Data synchronization

An attribute name "last_updated_time" is setting on every data record both in client cache and in server database. Client side will check current data with server side, before user does any update operation. If server side found client's current data are not latest, it will response latest data to client. Then, user will be reminded to notice new data before he updates information.

Server Design

Server side function

The server side is for administration and maintenance of user accounts. It is basically the same interface as the client application except that it does not have push notification since the administrator is not supposed to receive any of them. Also, the server side GUI is presented in a mobile Web page instead of an independent application.

As an important function of this application, server side is responsible for data synchronization. So we use attribute *timestamp*, which is *datetime* type, to mark the most recent timing point of update. With the help of this *timestamp*, the decision of which copy of data to use could be facilitated.

The server side has the following modules:

- Registration
    - Register a new patient user
- Logging
    - Log into the system
    - Log out of the system
- User Profile
    - Modify the user's profile
    - Download the patient's record
- Patient's Record
    - Update the patient's record
    - Add a patent's record
- Disease Information
    - Query disease information
    - Add a disease information to the system
    - Delete a disease from the system
- Hospital Information
    - Query the hospital information
    - Add a hospital to the system
    - Delete a hospital from the system
- Schedule List
    - View the schedule list
    - Submit a schedule by a patient
    - Accept a schedule by a doctor
    - Delete a schedule
- Meeting Minutes
    - Add some meeting minutes
    - View the meeting minutes
    - Delete meeting

The above modules as well as their sub-modules are further explained with their functions and applications in the Web interface.

Web Interface

This part will be filled out locally on a MS Word.

Name	register	
Description	Register a new patient user	
Parameters	loginName	The login name, which is picked by patient
	password	The password for the patient
	fullName	Full name of the patient
	email	Email address of the patient
	birthdate	Birth date of the patient, in YYYYMMDD format
	gender	Gender of the patient, “male” or “m” for male, “female” or “f” for female.
	married	Marriage status of the patient. “single” or “s” for single, “married” or “m” for married, “divorced” or “d” for divorced.
	allergies	Description on the patient’s allergies.

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’
		}

	Failed	Refer to the error list
Example	*register*?**loginName**=wen&**password**=123456&**fullName**=Wen%20Li&**email**=li%2ewen%40polyu%2eedu%2ehk&**birthdate**=19300101&**gender**=female&**married**=m&**allergies**=	

Name	Login	
Description	User login	
Parameters	loginName	The login name, which is picked by patient
	password	The password for the patient

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’,
		‘sessionid’ : ‘38fe9023ba0232cd1’
		}

	Failed	Refer to the error list
Example	*login*?**loginName**=wen&**password**=123456 A unique session is assigned for the continuous operations.	

Name	Logout	
Description	User logout, the session is terminated	
Parameters	sessionid	The session id

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’
		}

	Failed	Refer to the error list
Example	*logout*?**sessionid**=38fe9023ba0232cd1	

Name	Modifyprofile
Description	Modify user’s profile

Parameters	sessionid	The session id which is assigned while login
	oldpassword	The old password
	newpassword	The new password
	fullName	Full name of the patient
	email	Email address of the patient
	married	Marriage status of the patient. “true” or “t” for married, “false” or “f” for not married.
	allergies	Description on the patient’s allergies.

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’
		}

	Failed	Refer to the error list
Example	*modifyprofile*?**sessionid**=38fe9023ba0232cd1&**oldpassword**=123456&**newpassword**=654321&**fullName**=Wen%20Li&**email**=li%2ewen%40polyu%2eedu%2ehk&**birthdate**=19800101&**allergies**=green%20bean	

Name	Downloadrecord	
Description	Download the patients records	
Parameters	sessionid	The session id which is assigned while login
	patientName	The patient name, useful when inquiring by a doctor user. Wildcards are applied. “*Li”, “Wen*Li”

Returns	Successful	{
		'return': {
		'code':'0',
		'description':'Success'
		'patient':{
		'userid':'1',
		'name':'WEN LI',
		'record' : {
		'id' : '1',
		'timestamp' : '20120409201918',
		'date' : '20120407162349',
		'doctor' : '2',
		'symptom' : 'cough',
		'diagnosis' : '',
		'treatment' : 'rest',
		'remark' : ''
		},
		}
		}
		}

	Failed	Refer to the error list
Example	*downloadrecords*?**sessionid**=38fe9023ba0232cd1	

Name	Updaterecords	
Description	Update patient's record	
Parameters	sessionid	The session id which is assigned while login
	recordid	The id of a record
	date	The date
	doctorid	The doctor's user id
	sympton	The symptom
	diagnosis	The diagnosis
	treatment	The treatment
	remark	Remark

Returns	Successful	{
		'return' : {
		'code' : '0',
		'description' : 'success'
		}

	Failed	Refer to the error list
Example	*updaterecord*?**sessionid**=38fe9023ba0232cd1&**recordid**=1&**treatment**=intravenous%20drip	

Name	Addrecord	
Description	Add a patient's record, operable by a doctor only.	
Parameters	sessionid	The session id which is assigned while login
	patientid	The id of a patient
	date	The date
	doctorid	The doctor's user id
	sympton	The symptom
	diagnosis	The diagnosis
	treatment	The treatment
	remark	Remark

Returns	Successful	{
		'return' : {
		'code' : '0',
		'description' : 'success'
		}

	Failed	Refer to the error list
Example	*addrecord*?**sessionid**=38fe9023ba0232cd1&**patientid**=1&**date**=20120325&**doctored**=2&**symton**=toothache&**diagnosis**=tooth%20decay&**treatment**=canalis%20radicis%20dentis&**remark**=	

Name	Querydisease	
Description	Query disease information	
Parameters	sessionid	The session id which is assigned while login
	diseasename	The name of a disease, wildcard is applicable, i.g. "tooth*"
	department	The name of department, wildcard is applicable, i.g. "dental*"

Returns	Successful	{
		'return' : {
		'code' : '0',
		'description' : 'success',

		'disease' : {
		'id' : '1',
		'timestamp' : '20120409192132',
		'diseasename' : 'flu',
		'department' : 'Internal Medicine'
		}
		'disease' : {
		'id' : '2',
		'timestamp' : '20120409192132',
		'diseasename' : 'Tooth Decay',
		'department' : 'Dentistry'
		}
		}
		}

	Failed	Refer to the error list
Example	*querydisease*?**sessionid**=38fe9023ba0232cd1&**diseasename**=Fl%2a	

Name	Adddisease	
Description	Add disease information into MHS	
Parameters	sessionid	The session id which is assigned while login
	diseasename	The name of a disease
	department	The name of department

Returns	Successful	{
		'return': {
		'code':'0',
		'description':'Success'
		}
		}

	Failed	Refer to the error list
Example	*adddisease*?**sessionid**=38fe9023ba0232cd1&**diseasename**=fever&**department**=Intern Medicine	

Name	Deletedisease	
Description	Delete a disease information	
Parameters	sessionid	The session id which is assigned while login
	diseaseid	The id of a disease

Returns	Successful	{
		'return': {
		'code':'0',
		'description':'Success'
		}
		}

	Failed	Refer to the error list
Example	*deletedisease*?**sessionid**=38fe9023ba0232cd1&**diseaseid**=1	

Name	Queryhospital	
Description	Query hospital information	
Parameters	sessionid	The session id which is assigned while login
	hospitalname	The name of a hospital, wildcard is applicable, i.g.

		“hospital*”

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’,
		‘hospital’ : {
		‘id’ : ‘1’,
		‘timestamp’ : ‘20120409201554’,
		‘name’ : ‘Prince Wales Hospital’,
		‘email’ : ‘service@abcd.com’,
		‘address’ : ‘Shatin, N.T.’,
		‘tel’ : ‘12345678’,
		‘fax’ : ‘87654321’
		},
		}
		}

	Failed	Refer to the error list
Example	*queryhospital*?**sessionid**=38fe9023ba0232cd1	

Name	Addhospital	
Description	Add hospital information	
Parameters	sessionid	The session id which is assigned while login
	hospitalname	The name of a hospital, wildcard is applicable, i.e. “hospital*”
	email	The email address of the hospital
	address	The address of the hospital
	tel	The tel number
	fax	The fax number

Returns	Successful	{
		'return': {
		'code':'0',
		'description':'Success'
		}
		}

	Failed	Refer to the error list
Example	*addhospital*?**sessionid**=38fe9023ba0232cd1&**hospitalname**=abc&**email**=service@abc.com&**address**=Beijing*%20*Shanghai%20Chengdu&**tel**=88888888&**fax**=77777777	

Name	Deletehospital	
Description	Delete a hospital information	
Parameters	sessionid	The session id which is assigned while login
	hospitalid	The name of a hospital, wildcard is applicable, i.e. “hospital*”

Returns	Successful	{
		'return': {
		'code':'0',
		'description':'Success'
		}
		}

	Failed	Refer to the error list
Example	*deletehospital*?**sessionid**=38fe9023ba0232cd1&**hospitalid**=1	

Name	Viewschedule	
Description	View schedule list	
Parameters	sessionid	The session id which is assigned while login
	scheduleid	The id of a schedule
	doctoreid	The id of a doctor
	begintime	The begin time
	endtime	The end time
	patientid	The id of a patient

Returns	Successful	{
		'return': {
		'code':'0',
		'description':'Success',
		'schedule' : {
		'id' : '3',
		'timestamp' : '20120409202854',
		'doctorid' : '2',
		'doctorname' : '',
		'patientid' : '1',
		'patientname' : 'WEN LI',
		'begintime' : '201204081400',
		'endtime' : '201204081600',
		'status' : 'confirmed'
		},
		}
		}

	Failed	Refer to the error list
Example	*viewschedule*?**sessionid**=38fe9023ba0232cd1&**doctorid**=1	

Name	Submitschedule	
Description	Submit a schedule request	
Parameters	sessionid	The session id which is assigned while login
	doctorid	The id of a doctor
	begintime	The begin time
	endtime	The end time
	patientid	The id of a patient

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’
		}

	Failed	Refer to the error list, also returns the current schedule list of the specified doctor.
Example	*addschedule*?**sessionid**=38fe9023ba0232cd1&**doctorid**=1&**begintime**=20120325%2011%3a30&**endtime**=20120325%2012%3a00&**patientid**=5	

Name	Acceptschedule	
Description	Accept a schedule request	
Parameters	sessionid	The session id which is assigned while login
	scheduleid	The id of the schedule
Returns	Successful	{

		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’
		}

	Failed	Refer to the error list, also returns the current schedule list of the specified doctor.
Example	*changeschedule*?**sessionid**=38fe9023ba0232cd1&**begintime**=20120325%2011%3a30&**endtime**=20120325%2012%3a30&**patientid**=5	

Name	Deleteschedule	
Description	Delete a schedule	
Parameters	sessionid	The session id which is assigned while login
	scheduleid	The id of the schedule

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’
		}

	Failed	Refer to the error list, also returns the current schedule list of the specified doctor.
Example	*deleteschedule*?**sessionid**=38fe9023ba0232cd1&**scheduleid**=3	

Name	Addmeetingminutes	
Description	Add a meeting minutes	
Parameters	sessionid	The session id which is assigned while login
	scheduleid	The related schedule id, if any
	attender	The attender user id list, separated by ‘,’(comma)
	date	The meeting date time
	minutes	The meeting minutes

Returns	Successful	{
		‘return’ : {
		‘code’ : ‘0’,
		‘description’ : ‘success’
		}

	Failed	Refer to the error list.
Example	*addmeetingminutes*?**sessionid**=38fe9023ba0232cd1&**attender**=1,2&**scheduleid**=1&**date**=20120409&**minutes**=make an urgent operation	

Name	Viewmeetingminutes	
Description	View meeting minutes	
Parameters	sessionid	The session id which is assigned while login
	attender	The attender’s id list, separated by comma, wildcard “*” is applicable.
	scheduleid	The related schedule id, if any. Wildcard “*” is applicable.
	date	The specified date in YYYYMMDD format

Returns	Successful	{
		‘return': {
		'code':'0',
		'description':'Success',
		'meetingminutes' : {
		'id' : '1',
		'timestamp' :
		'20120410011723',

		'attenders' : 'attenders',
		'scheduleid' : '0',
		'date' : '20120410',
		'minutes' : 'made an operation'
		},
		'meetingminutes' : {
		'id' : '2',
		'timestamp' : '20120410011834',
		'attenders' : '1,2',
		'scheduleid' : '0',
		'date' : '20120410',
		'minutes' : 'made an operation'
		},
		}
		}

	Failed	Refer to the error list.
Example	*viewmeetingminutes*?**sessionid**=38fe9023ba0232cd1&**attender**=2	

Name	Deletemeetingminutes	
Description	Delete a meeting minutes	
Parameters	sessionid	The session id which is assigned while login
	meetingminutesid	The id of the meetingminutes

Returns	Successful	{
		'return' : {
		'code' : '0',
		'description' : 'success'
		}

	Failed	Refer to the error list.
Example	*deletemeetingminutes*?**sessionid**=38fe9023ba0232cd1&**meetingminutesid**=1	

Error Code	Error Description
0	Success
1	Login name is already taken
2	Incorrect password
3	Wrong full name
4	Invalid email address
5	Wrong date format
6	Invalid schedule id
7	Invalid doctor id
8	Conflict begin time
9	Conflict end time

99	Unknown Error
10000+	Database Error Code, starting from 10000. For example: Code 11062 represents the database error code 1062

SQL Schema

```
DROP DATABASE if exists mhs;

################################################
# Create Database
################################################
CREATE DATABASE mhs;
USE mhs;

################################################
# Table: push
################################################
CREATE TABLE push
(
                needPush                BOOLEAN         NOT NULL DEFAULT
False
);

INSERT INTO push VALUES('FALSE');

################################################
# Table: user
# ---------------------------------------------
# For the user profile data
################################################
CREATE TABLE user
(
                id                      INT             not null auto_increment
primary key,    # primary key
                loginName               VARCHAR(64)     not null unique,
                # login name
                timestamp               DATETIME    not null,
                # last modified time stamp
                password                VARCHAR(64)     not null,
                                        # password
                fullName                VARCHAR(64),
                                        # full name
                userType                CHAR(1)         not null default 'p',
                                        # 'p' for patient, 'd' for doctor, 'b' for both
                emailAddress            VARCHAR(256),
                                        # user email address
```

```
		birthDate		DATETIME	not null,
		# birth date
		gender			CHAR(1)		not null,
		# gender, "M/m" for male, "F/f" for female
		married			CHAR(1)		not null,
		# marriage status, 's' for single, 'm' for married, 'd' for divorced
		allergies		VARCHAR(1024)	default ''
		# allergies
);
DELIMITER |
CREATE TRIGGER push_user BEFORE INSERT ON user
FOR EACH ROW BEGIN
		UPDATE push SET needPush=True;
		SET new.timestamp = now();
END;
CREATE TRIGGER update_user BEFORE UPDATE ON user
FOR EACH ROW BEGIN
		UPDATE push SET needPush=True;
		SET new.timestamp = now();
END;
CREATE TRIGGER delete_user BEFORE DELETE ON user
FOR EACH ROW BEGIN
		UPDATE push SET needPush=True;
END;
|
DELIMITER ;

################################################
# Table: session
################################################
CREATE TABLE session
(
		userId			INT		NOT NULL UNIQUE PRIMARY KEY,
		# user Id
		sessionId		VARCHAR(16)	NOT NULL,
		# session id
		lastAccess		TIMESTAMP  NOT NULL
		# last access time
);

DELIMITER |

CREATE TRIGGER insert_session BEFORE INSERT ON session
FOR EACH ROW BEGIN
		SET new.lastAccess = now();
END;
CREATE TRIGGER update_session BEFORE UPDATE ON session
FOR EACH ROW BEGIN
		SET new.lastAccess = now();
END;
|

DELIMITER ;


################################################
# Table: record
```

```
# ---------------------------------------------
# For healthy record
################################################
CREATE TABLE record
(
                id                              INT             NOT             NULL
AUTO_INCREMENT PRIMARY KEY,     # primay key
                timestamp               DATETIME    NOT NULL,
                # last modified time
                patientId               INT     NOT NULL,                       #
the patient user id
                createDate              DATETIME,
                # date
                doctorId                INT     NOT NULL,                       #
the doctor user id
                sympton                         TEXT,
                                                # symptom
                diagnosis               TEXT,
                # diagnosis
                treatment               TEXT,
                # treatment
                remark                          TEXT
                                                # remark
);

DELIMITER |
CREATE TRIGGER insert_record BEFORE INSERT ON record
FOR EACH ROW BEGIN
                SET new.createDate = IFNULL(new.createDate, now());
                SET new.timestamp = now();
                UPDATE push SET needPush=True;
END;
CREATE TRIGGER update_record BEFORE UPDATE ON record
FOR EACH ROW BEGIN
                SET new.timestamp = now();
                UPDATE push SET needPush=True;
END;
CREATE TRIGGER delete_record BEFORE DELETE ON record
FOR EACH ROW BEGIN
                UPDATE push SET needPush=False;
END;
|
DELIMITER ;

################################################
# Table: disease
################################################
CREATE TABLE disease
(
                id                              INT             not     null    auto_increment
primary key,    # primary key
                diseaseName             TEXT            not null,
                # disease name
                department              TEXT            not null,
                # department
                timestamp               DATETIME    not null                        #
timestamp
);
```

```
DELIMITER |
CREATE TRIGGER insert_disease BEFORE INSERT ON disease
FOR EACH ROW BEGIN
	SET new.timestamp = now();
		UPDATE push SET needPush=True;
END;
CREATE TRIGGER update_disease BEFORE UPDATE ON disease
FOR EACH ROW BEGIN
	SET new.timestamp = now();
		UPDATE push SET needPush=True;
END;
CREATE TRIGGER delete_disease BEFORE DELETE ON disease
FOR EACH ROW BEGIN
		UPDATE push SET needPush=True;
END;
|
DELIMITER ;

################################################
# Table: organization
################################################
CREATE TABLE organization
(
		id			INT		not null auto_increment primary key,	# primary key
		orgName			varchar(256)	not null,
					# organization name
		orgType			char(1)		not null default 'h',
					# "h" for hospital, 'o' for organization
		email			varchar(256),
					# email address for the orgnization
		address			text(1024),
					# address of the orgnization
		tel			varchar(256),
					# phone number
		fax			varchar(256),
					# fax number
		timestamp		DATETIME
		# time stamp for modification
);

DELIMITER |
CREATE TRIGGER insert_organization BEFORE INSERT ON organization
FOR EACH ROW BEGIN
	SET new.timestamp = now();
		UPDATE push SET needPush=True;
END;
CREATE TRIGGER update_organization BEFORE UPDATE ON organization
FOR EACH ROW BEGIN
	SET new.timestamp = now();
		UPDATE push SET needPush=True;
END;
CREATE TRIGGER delete_organization BEFORE DELETE ON organization
FOR EACH ROW BEGIN
		UPDATE push SET needPush=True;
END;
|
```

```
DELIMITER ;

################################################
# Table: schedule
################################################
CREATE TABLE schedule
(
                id                              INT             not     null    auto_increment
primary key,    # primary key
                doctorId                INT     not null,                                       #
doctor user id
                patientId               INT     not null,                                       #
patient user id
                status                          varchar(16)     not null default 'new',
                                        # status, "new", "confirmed"
                beginTime               DATETIME    not null,
                # begin time
                endTime                         DATETIME        not null,
                                        # end time
                timestamp               DATETIME    not null                                    #
last updated timestamp
);

DELIMITER |
CREATE TRIGGER insert_schedule BEFORE INSERT ON schedule
FOR EACH ROW BEGIN
        SET new.timestamp = now();
                UPDATE push SET needPush=True;
END;
CREATE TRIGGER update_schedule BEFORE UPDATE ON schedule
FOR EACH ROW BEGIN
        SET new.timestamp = now();
                UPDATE push SET needPush=True;
END;
CREATE TRIGGER delete_schedule BEFORE DELETE ON schedule
FOR EACH ROW BEGIN
                UPDATE push SET needPush=True;
END;
|

DELIMITER ;

################################################
# Table: meetingminutes
################################################
CREATE TABLE meetingminutes
(
                id                              INT             NOT                     NULL
AUTO_INCREMENT PRIMARY KEY,     # primary key
                attenders               TEXT            NOT NULL,
                # attender user Id list, separated by ','
                scheduleId              INT,                                                    #
related schedule, if any
                timestamp               DATETIME    NOT NULL,
                date                            DATETIME        NOT NULL,
                                        # meeting time
                minutes                         TEXT
                                        # meeting minutee
```

```
);
DELIMITER |
CREATE TRIGGER insert_meetingminutes BEFORE INSERT ON meetingminutes
FOR EACH ROW BEGIN
                SET new.timestamp = now();
                UPDATE push SET needPush=True;
END;
CREATE TRIGGER update_meetingminutes BEFORE UPDATE ON meetingminutes
FOR EACH ROW BEGIN
                SET new.timestamp = now();
                UPDATE push SET needPush=True;
END;
CREATE TRIGGER delete_meetingminutes BEFORE DELETE ON meetingminutes
FOR EACH ROW BEGIN
                UPDATE push SET needPush=True;
END;
|
DELIMITER ;
```

Data synchronization

According to the last updated time, server system can compare every client request data with corresponded data in server database, and store latest updated data. Moreover, as we said in client's data synchronization, server will make sure user get latest data, before they do any operation.

Push

As an Apple Push Notification Service(APNS) is required to support server positively sending notifications to mobile side which is based on mobile device UDID.

Reference:
http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html

Contribution List

- QING Pei
    - Client Side
- Mei Youzhi
    - Client Side
- SHAO Shuai
    - Server Side
- RAN Jun
    - Server Side

Deployment Guide

For Installation

1. Make sure that MySQL5.1 or above, PHP5.3 is ready
2. Install the necessary packages
   sudo apt-get install php5-mysql php5-curl
3. Execute below command to build up the database.
   mysql --user=root --password=****** < db/schema.sql
4. copy all the files onto webserver mhs/
   cd php
   cp -r * /var/www/mhs/
   chown -R go+rw /var/www/mhs

For Pushing

Before using, the following needs to be done:
1. install the curl module for PHP. In Ubuntu:
   sudo apt-get install php5-curl
2. Re-construct the database
   mysql --user=root --password=your-password < db/schema.sql
3. Test the push function via:
http://ipaddr/mhs/push.php?registration=RegistrationIDByGoogle&auth=YourGoogleAuth

Making some changes to the database is necessary, i.e. addrecord, addhospital. Otherwise, PUSH is not triggered if there is no change to the data.

Get Started

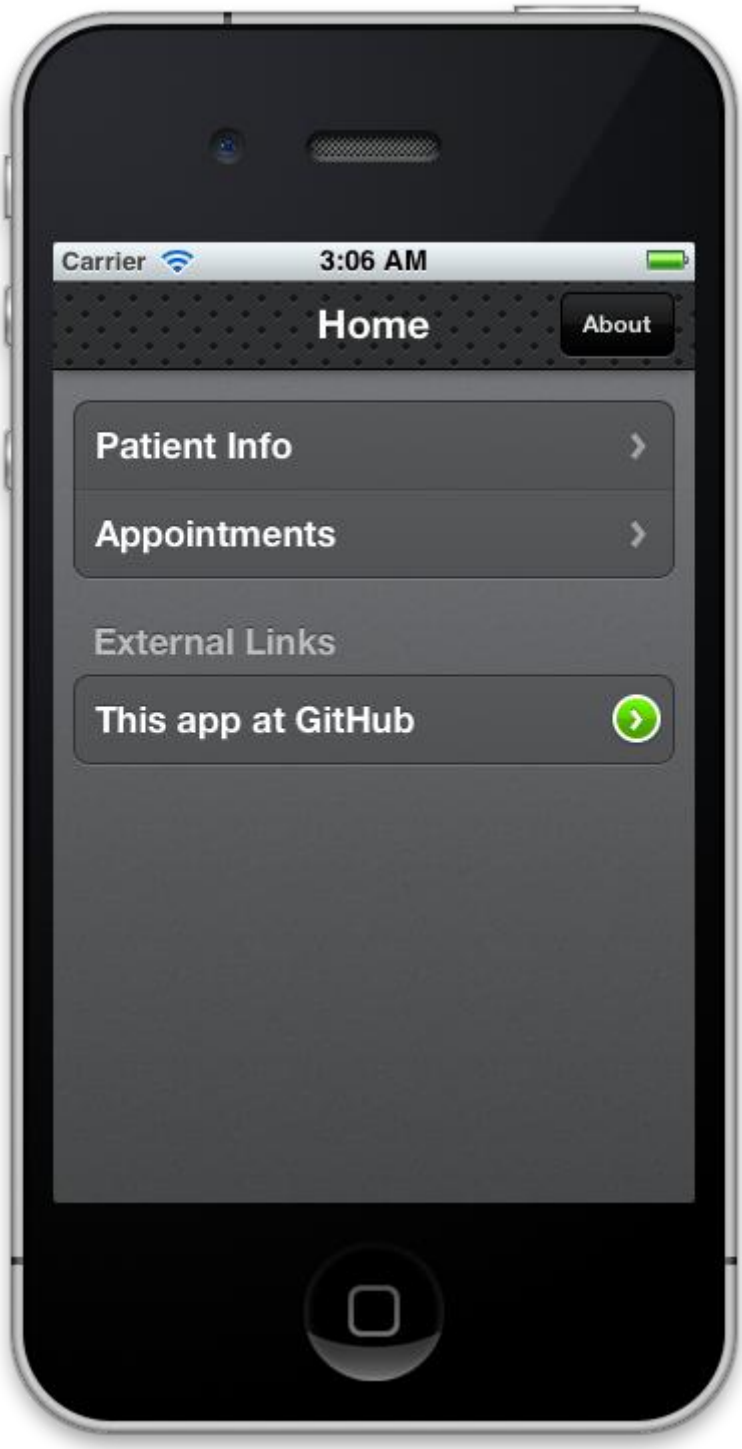
Get the app installed

So far, we do not have a iOS Developer membership to distribute the app. The following is based on iPhone Simulator 5.1.

Alternatively, you may use this as an web app. Just visit this link.

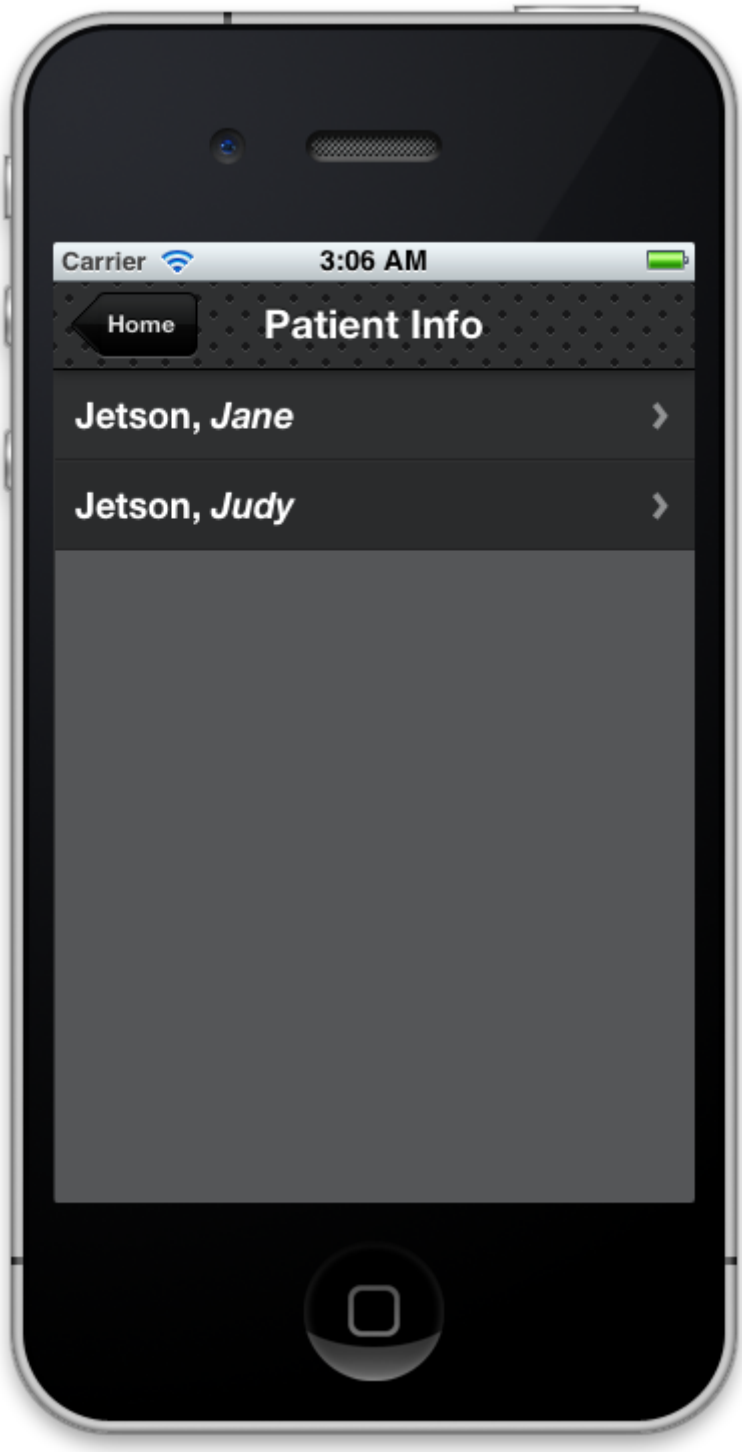
Starting point

Start the app and you will be shown the homepage.



Patient List

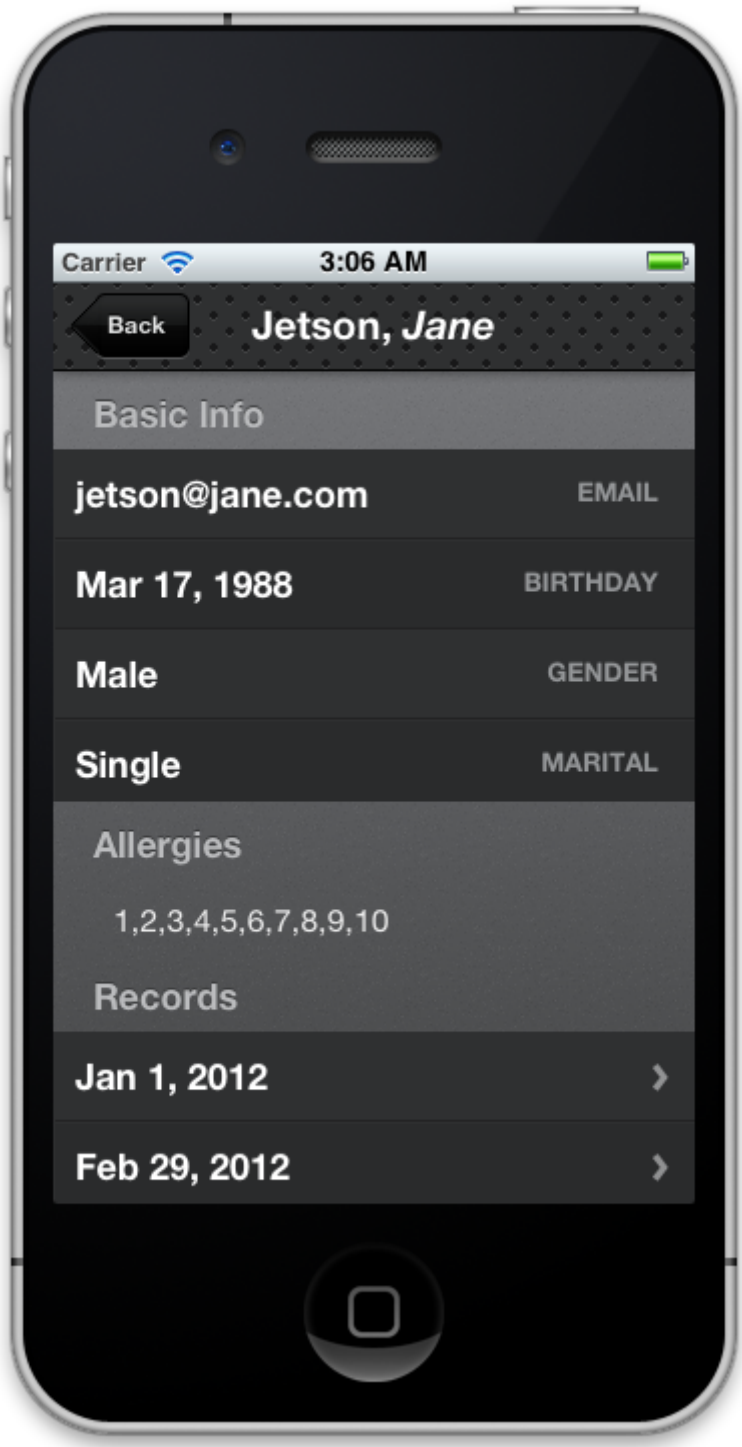
Tap “Patient Info” will lead you to a list of patient names.



Patient Information

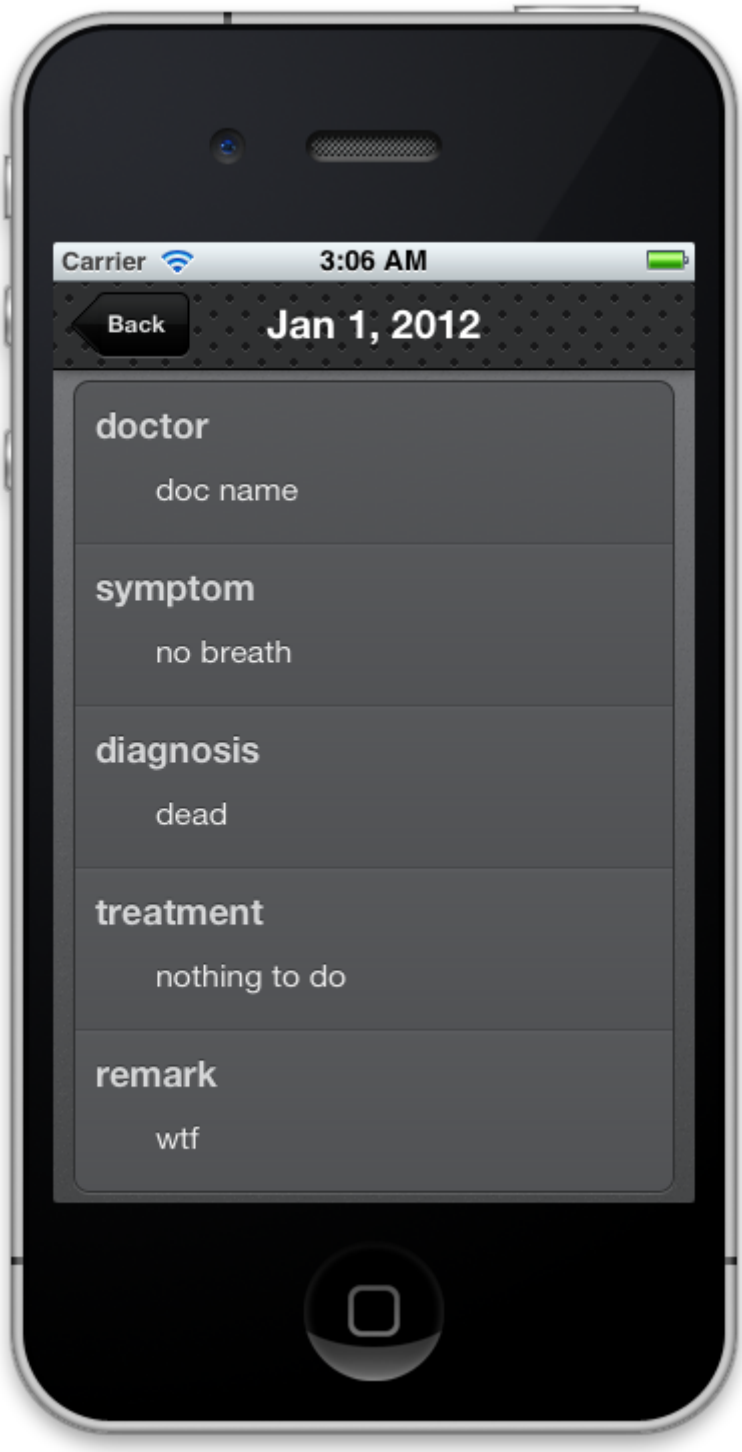
Tap a name and you will see the details.

Healthcare records are listed after some basic information.



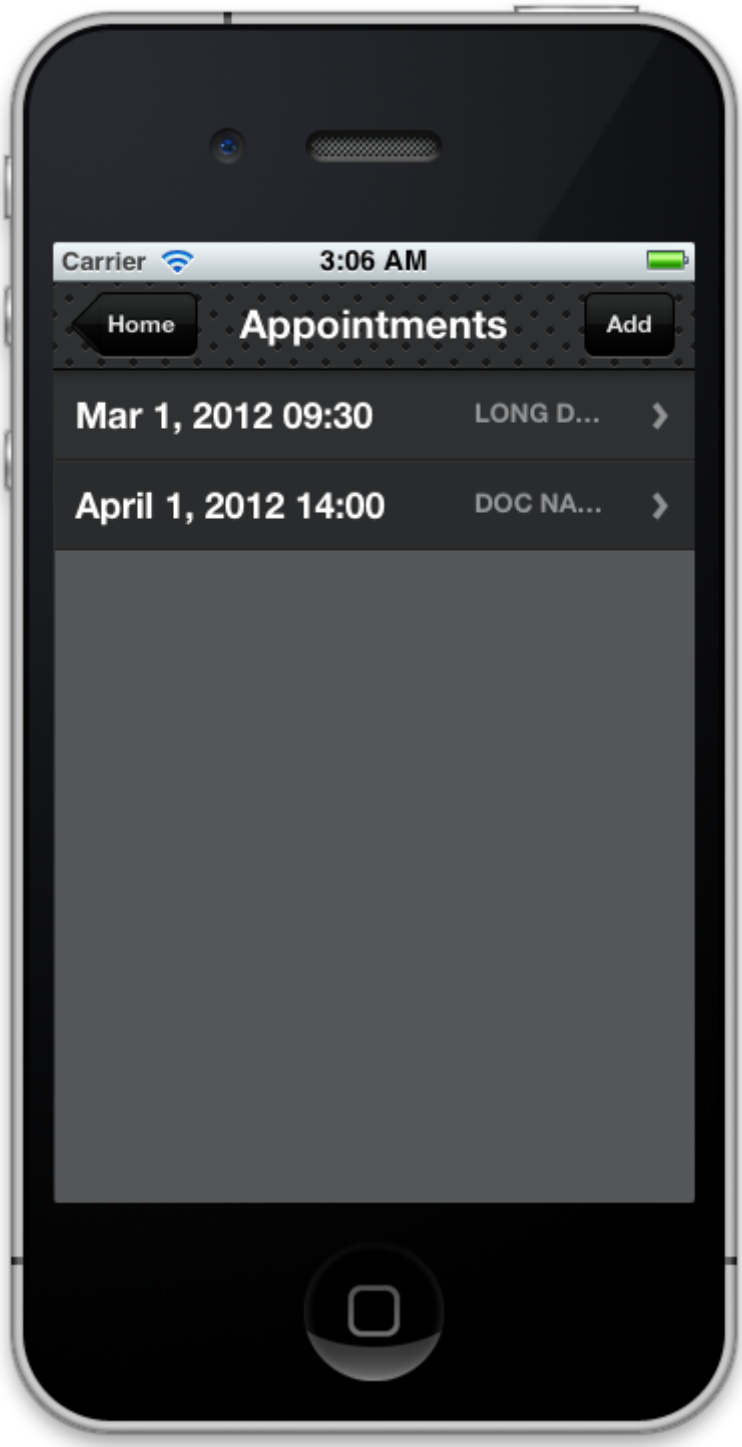
Healthcare Record

By tapping the record date, you can view the healthcare record information.



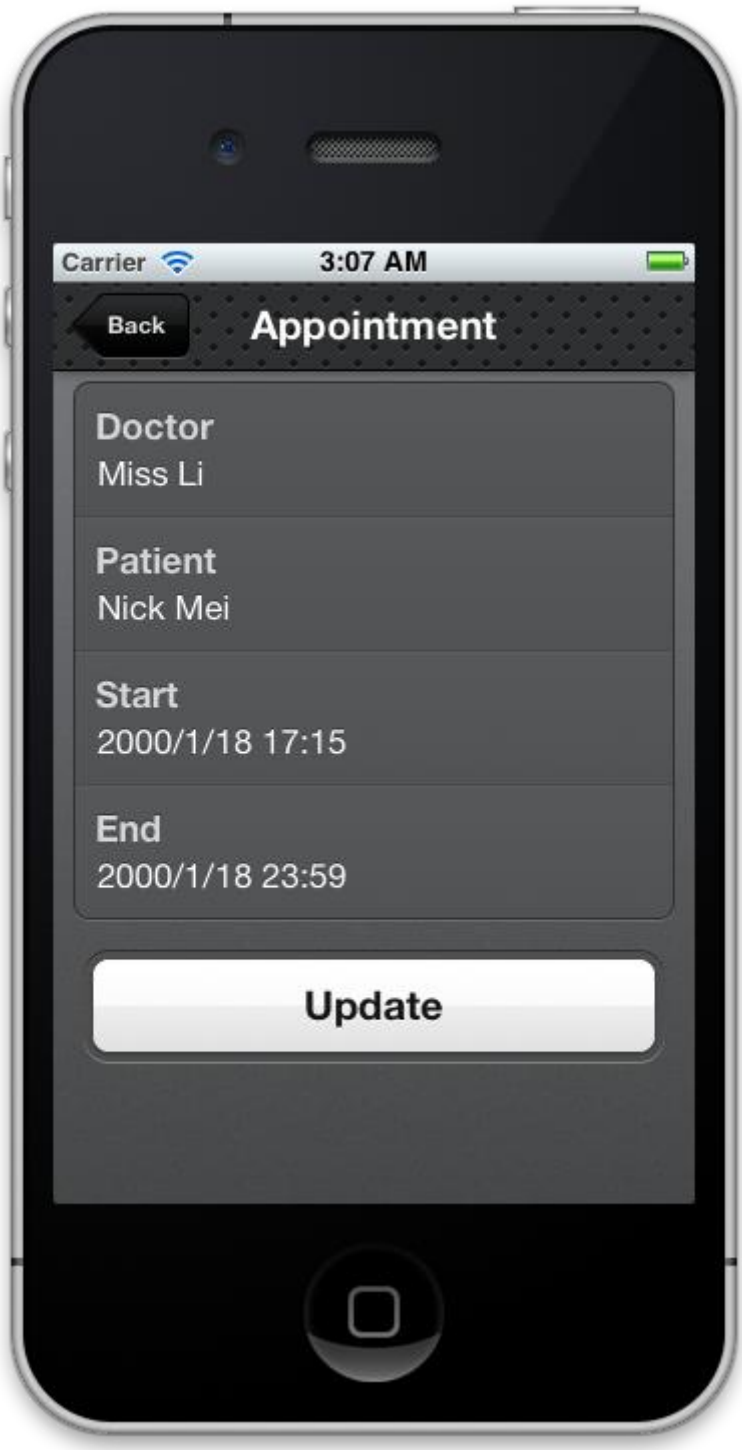
Appointments

Tapping “Appointments” in the homepage, you will see a list of appointments.



Appointment view

You can check the appointment details by tapping on one of the appointments.
You may make changes to it and tap “Update”.



New appointment

By clicking the “Add” button at the top right corner in Appointments view, you can create a new appointment.
Type in the data and tap the “Add” button.