

Wireless ECG Monitor System Design—Web Service and Android Client Development

by Weizheng Li

Bachelor of Engineering, Southeast University, 2014

A Report Submitted in Partial Fulfillment  
of the Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

© Weizheng Li, 2016  
University of Victoria

All rights reserved. This report may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.

## **Supervisory Committee**

Wireless ECG Monitor System Design—Web Service and Android Client Development

by

Weizheng Li

Master of Engineering, Southeast University, 2014

### **Supervisory Committee**

Dr. Xiaodai Dong, Department of Electrical and Computer Engineering  
**Supervisor**

Dr. Hongchuan Yang, Department of Electrical and Computer Engineering  
**Departmental Member**

## Abstract

The heart attack has killed millions of people all over the world. Although the medical level is experiencing a high speed progress, there are increasing amount of sudden deaths caused by heart disease every year. One of the reasons is that, normally, there are no obvious symptoms in advance and patients can hardly get first ad treatment timely.

However, if patients and doctors can monitor patients' electrocardiogram (ECG) signal more frequently and timely, many heart problems can be captured by ECG. Proper treatment can be conducted, which reduces the chance of more severe heart events. In the case of heart attack, ECG monitoring can immediately identify the dangerous situation, and give out alarm signal to emergency, further reducing the chance of sudden cardiac death. Due to this motivation, we developed a wireless and wearable ECG monitoring system which enable patients to obtain their ECG data anywhere and anytime. The system would be used not only for sudden death but also to help doctor's treatment.

The system consists of a wireless ECG sensor, mobile APP, web server and web sites. Patients use the ECG sensor to test their ECG data and the data would be transmitted to mobile APP. On the mobile APP, a user can view the real-time ECG data and the APP would upload ECG data to a web server. On the web site, patients and their authorized doctors can view and comment the ECG data.

This project focuses on the development of Android mobile APP and web service.

## Table of Contents

Supervisory Committee .....	I
Abstract .....	II
Table of Contents .....	III
List of Tables .....	V
List of Figures .....	VI
Acknowledgments.....	VII
Chapter 1 Introduction .....	1
1.1 Background.....	1
1.2 Introduction to Cloud Web Service .....	2
1.3 Introduction to Mobile Client APP .....	2
1.4 Outline of report.....	3
Chapter 2 Cloud Web Service Design and Implementation .....	4
2.1 Introduction.....	4
2.1.1 Concept of Web Service and REST (Representational State Transfer).....	4
2.1.2 Architecture of web service .....	6
2.1.3 Technology Stack.....	7
2.2 Cloud Web Service Configuration.....	8
2.2.1 Jersey Restful Web Service Configuration .....	8
2.2.2 Tomcat Server Configuration .....	10
2.2.3 Database Configuration .....	11
2.2.4 Firewall (Iptables) configuration .....	12
2.3 Interface Data Format .....	13
2.3 Database Design and Architecture.....	14
2.4 Patient/Doctor authentication.....	16
2.5 Patient/Doctor relation and permission management .....	17
2.6 ECG data storage and management .....	18
2.8 Doctor Comments & Patient Notes.....	21
2.8.1 Patient Notes .....	21
2.8.2 Doctor Comments .....	23
2.9 Search API .....	24
2.10 Overview of APIs .....	25
Chapter 3 Mobile Client App Design and Implementation .....	28
3.1 Introduction.....	28
3.2 ECG Data Receiving via Bluetooth Low Energy .....	29
3.2.1 Bluetooth Low Energy .....	29
3.2.2 Android API for BLE.....	30
3.2.3 Data Format and Data Rate.....	31
3.3 Real Time ECG Signal Displaying.....	33
3.3.1 ECG Data Parsing and Displaying.....	33
3.3.2 Single/Double Channel Displaying.....	34
3.3.3 Real Time Heart Rate and Ventricular tachycardia / Ventricular fibrillation monitoring.....	36
3.4 Local Storage Management .....	36

3.5 ECG Test.....	38
3.5.1 Quick Test.....	39
3.5.2 Long Term Monitor .....	41
3.6 History ECG Data Management .....	43
3.6.1 Quick Test Report History Management .....	43
3.6.2 Long Term ECG Data Management .....	46
3.7 Doctor / Doctor's Comment Management.....	49
3.7.1 Doctor Management.....	49
3.7.2 Doctor Comment Management.....	50
Chapter 4 Conclusion and Future Work .....	52
4.1 Conclusion .....	52
4.2 Future Work .....	52
Bibliography .....	54
Appendix A: Structure of Database Tables .....	55
Table patient_info .....	55
Table doctor_info.....	56
Table relation .....	57
Table access_request.....	57
Table doctor_comment .....	57
Table comment_relation .....	58
Table mynote .....	58
Table note_relation .....	58
Table ecgtestinfo.....	58
Table ecgdatainfo.....	59
Appendix B: Server Configuration .....	61
Step One—Update and install SSH .....	61
Step Two—Install Nginx .....	61
Step Three—Install tomcat7 .....	61
Step Four—Install MySQL.....	61
Step Five—Import ecgdatabase .....	62
Ctrl+C to exit .....	62
mysql -u root -p -h localhost ecgdatabase<ecgdatanase.sql .....	62
Step Six—Import our ECG WebService servlet.....	62

## List of Tables

Table 1 Technology stack of ECG Web Service .....	7
Table 2 APIs for Patient Notes .....	22
Table 3 APIs for Doctor Comments .....	23

## List of Figures

Figure 1 Structure of the ECG system .....	2
Figure 2 Architecture of Web Service .....	6
Figure 3 Jersey Framework Jar Files .....	9
Figure 4 Web.xml Configuration .....	10
Figure 5 JDBC import.....	11
Figure 6 Work flow for Patient/Doctor Relation Management .....	18
Figure 7 ECG Data Storage .....	19
Figure 8 ECG Test List on Web Site .....	20
Figure 9 ECG Data Displaying on Web Site .....	21
Figure 10 ECG Test Search Function on Web Site .....	25
Figure 11 Work Flow of the ECG Monitor System.....	28
Figure 12 ECG Sensor in BLE Device List .....	31
Figure 13 Channel Selection Menu.....	34
Figure 14 Single and Double ECG Data Displaying .....	35
Figure 15 Local Storage Folder Structure.....	37
Figure 16 ECG Test Selection .....	39
Figure 17 Quick Test Count Down.....	40
Figure 18 A Example of Quick Test Report .....	41
Figure 19 Data Management Option.....	43
Figure 20 Examples of Quick Test Report .....	45
Figure 21 ECG Parameter Trend Line Chart .....	46
Figure 22 Long Term ECG Data Management.....	47
Figure 23 ECG Record List .....	48
Figure 24 History ECG Data Plot .....	48
Figure 25 Doctor Management .....	50
Figure 26 Doctor Comment .....	51

## Acknowledgments

I would like to thank:

Dr. Xiaodai Dong for trusting me and giving me this practice opportunity. During this project, I have learnt a lot regarding programming and system design under the instruction of Dr Xiaodai Dong.

Weizheng Li

April, 2016

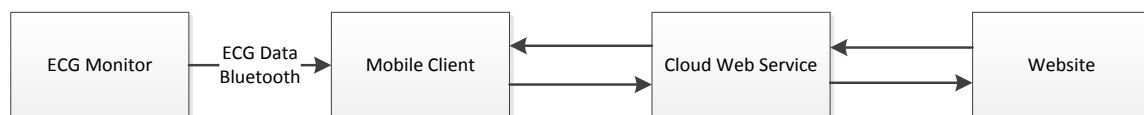
# Chapter 1 Introduction

## 1.1 Background

Heart Disease is one of the most challenging disease human are facing nowadays. There are a large number of people died from heart attack everyday in every corner of the world. A wireless and wearable ECG monitor system can be useful to detect a patient's heart problem and can send out alarm when heart attack happens so that many of the sudden deaths may be avoided.

Currently, besides the traditional hospital ECG tests, the holter monitor has also been used for ECG data monitoring. Holter is a portable ECG monitor which a patient can wear for one or two days. Then the data would be collected after the monitoring period. However, one of the most severe weakness of the holter is that the ECG data is not real time. That means, the patients and the doctors can only get the ECG data after the monitoring is finished and the holter be sent to the hospital. Typically the analysis of holter data takes a long time before the report is available to the patient.

The wireless ECG monitor system consists of four part: 1) Wearable and wireless ECG monitor which can be worn by patients to collect and transmit patients ECG data, 2) Mobile client which can receive ECG data from the ECG monitor via Bluetooth, manage and view history ECG data, analyse ECG data and upload ECG data to cloud, 3) Cloud server which supports mobile application as well as web front end, 4)Website which is used for both users and doctors to view and manage the history ECG data and write/read comments.



**Figure 1 Structure of the ECG system**

## 1.2 Introduction to Cloud Web Service

In our ECG monitor system, we have a web server to support mobile APP and web site. The web server have a database storing all the information of our ECG monitor system and the web service provides APIs as the interface so that both mobile APP and web site can contact with the server.

The web service is based on Wanbo Li, a previous Masc student's work. Comparing with Wanbo Li's work, current web service now can support web site and new features are added in order to support new functions on the mobile client side.

The details of web service will be introduced in Chapter 2.

## 1.3 Introduction to Mobile Client APP

In the wireless ECG monitor system, the mobile APP plays a central role to connect the ECG monitor to the cloud server. Patients use a wireless ECG monitor to detect their ECG data and the data is transmitted to the mobile APP via Bluetooth Low Energy.

Our APP is now built on Android platform and designed for patient use. The features of the mobile APP include:

- Receiving ECG data from an ECG monitor
- Parsing data and real-time display
- Upload ECG data to server
- Real-time ECG data analysing
- One minute Quick ECG data analysis
- History ECG data management
- Patient login and profile management

The details of mobile APP will be introduced in Chapter 3.

#### **1.4 Outline of report**

The rest of this report is organized as follows:

Chapter 2 introduces the design and development of the Cloud Web Service. The Web Service's architecture, database design, API design and server configuration & implementation will be included.

Chapter 3 describes how the mobile (Android) APP is designed and built. It elaborates how the APP receive, display and store ECG data and ECG quick test system.

Chapter 4 concludes the report and suggests future work.

## Chapter 2 Cloud Web Service Design and Implementation

### 2.1 Introduction

In this chapter, we will introduce the design and implementation of ECG cloud web service.

The ECG cloud web service follows the rules of restful web service style which provides API for clients including web site and mobile APP. As this project is based on Wanbo Li, a previous Masc student's work, we will only introduce the functions implemented after Wanbo's work.

#### 2.1.1 Concept of Web Service and REST (Representational State Transfer)

A Web Service [1] is a service offered by an electronic device to another electronic device, communicating with each other via the World Wide Web. In a web service, web technology such as the HTTP, originally designed for human-to-machine communication, is utilized for machine-to-machine communication, more specifically for transferring machine readable file formats such as XML and JSON. In a real project, the web service is typically sets of interface (API) to a database server, utilized for example by another web server, or by a mobile application, that provides a user interface to the end user. Another common application offered to the end user may be a mashup, where a web server consumes several web services at different machines, and compiles the content into one user interface.

There are many kinds of web service frameworks and design patterns including Spring based on Java, Django based on Python, Rail on Ruby. Nowadays, a new design pattern

of web Service that becomes increasing popular is restful Web Service. To some extent, restful web service has become the default for most of internet companies, while its importance is likely to continue growing quickly as all technologies move towards an API orientation. In this report, we choose to use Jersey framework based on Java and follow the rules of restful web service which is also called as Representational State Transfer (REST). Although some of the existing framework also support restful style, Jersey a pure restful framework which is focused on supporting restful web service.

According to [2], the REST is an architectural style that specifies constraints, such as the uniform interface. This architectural style can enable services to work best on the Web with good performances on key properties such as performances, scalability, and modifiability.

According to the REST architectural style, all the data and functionality are regarded as resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources [2] are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

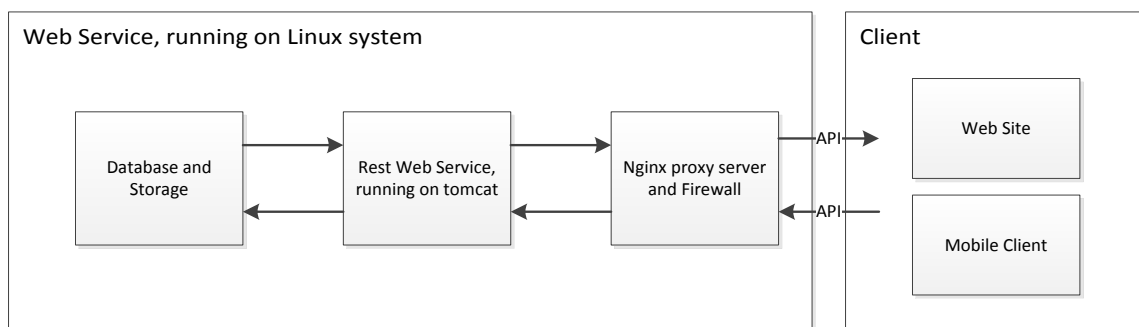
For all the APIs provided by the web service in this project, they are totally stateless which means no matter when and who is calling, the response will always be irrelevant with time and the state of the client.

There is another concept related to the web service which is the ‘web server’. Generally, the server always refers to the physical machine which our web service runs

on. Nowadays, most IT companies would not have the physical server machine in their office. Instead, people often buy the cloud server service such as Amazon's AWS and Microsoft's Azure. In our ECG system, we use a machine in the Lab as the server.

### 2.1.2 Architecture of web service

As shown in Figure below, the service consists of database, rest web service and a proxy server which is used as a port switcher. There is an iptable served as the firewall at the outermost layer of the server.



**Figure 2 Architecture of Web Service**

The clients of our ECG system includes web site and mobile APP. The client can visit the web service to read or write data via the API provided by the web service. If we consider the server as a box, the APIs are the interfaces provided by the server through which the client is able to visit the server.

Within the web service, the outermost layer is the firewall which can block the requests according to the request's IP address. Then we have a proxy server which can transmit the request from the 80 port to the real port of our server.

The innermost layer of the web service is the database and hard disk which store all the data. The web service framework runs between the database and the proxy server. The web service would have functionality to support all the logistics.

### 2.1.3 Technology Stack



**Table 1 Technology stack of ECG Web Service**

As shown in the Table 2, the Cloud Web Service's technology stack consists of Linux System, Tomcat 7, MySQL database, Jersey Restful Web Service framework and Java programming language.

Linux system: Linux system is always used by server in all kinds of Internet companies including Google, Amazon, Uber because of its high efficiency, stability and open source. Our ECG Web Service currently runs on the Ubuntu 14 system.

Tomcat container: Tomcat is an application server from the Apache Software Foundation that executes Java servlets and renders Web pages that include Java Server Page coding. Described as a "reference implementation" of the Java Servlet and the Java Server Page specifications, Tomcat is the result of an open collaboration of developers and is available from the Apache Web site in both binary and source versions. Our web service is deployed on the tomcat7.

MySQL database: MySQL is one of the most popular database among today's Internet companies including Youtube, Facebook, LinkedIn. In this report, MySQL is used as the backend database to store data including user's information and ECG data information.

Jersey restful Web Service framework [3]: Jersey RESTful Web Services framework is open source, production quality, framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs and serves as a JAX-RS (JSR 311 & JSR 339) Reference Implementation. Jersey provides its own API that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development. Jersey also exposes numerous extension SPIs so that developers may extend Jersey to best suit their needs.

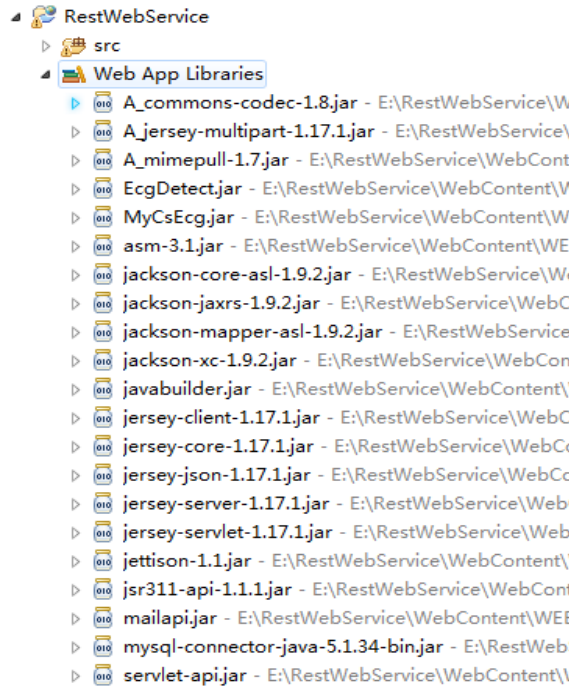
## **2.2 Cloud Web Service Configuration**

In this section, we describe how the server is setup and how each part of the server is connected.

### **2.2.1 Jersey Restful Web Service Configuration**

As introduced in Section 2.1, our web service is built upon Jersey framework. We can download Jersey framework from <https://jersey.java.net>. In this project, we use MyEclipse as the IDE for server development.

After download, we can import the jar file into the Web Service project. Shown in the figure below:



**Figure 3 Jersey Framework Jar Files**

Next configuration is to set up servlet to listen to request in web.xml file, telling the web service that we use Jersey Rest Service to handle requests. The web.xml file is derived from the Servlet specification, and contains information used to deploy and configure the components of web applications.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:
  <display-name>RestWebService</display-name>

  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>
      com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>robertlee.restwep.resource</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>

```

**Figure 4 Web.xml Configuration**

Then we can start to write the APIs using Jersey Framework. For example, the code below is the most basic API with GET method and accepting Text\_plain request.

```

@GET
@Produces(MediaType.TEXT_PLAIN)
public String runningTest() {
    return "EcgCloudDatabase Web Service is running...new";
}

```

## 2.2.2 Tomcat Server Configuration

After the configuration of Jersey Framework, we then need to set up tomcat as server container to deploy our web service. Firstly, we can download tomcat from the official website of Apache or install with the command *sudo apt-get install tomcat7*.

To get tomcat7 running, type

```
sudo service tomcat7 start
```

```
sudo adduser ecg tomcat7
```

To confirm tomcat7 running properly, type `//localhost:8080`, we will see tomcat7 welcome page.

### 2.2.3 Database Configuration

MySQL database can be downloaded from <https://www.mysql.com/> freely or with command `sudo apt-get install mysql-server`.

After downloaded, we can setup user and password to login and create database and tables. However, as we use Java to develop server, we need write codes in Java to visit database. Thus, we need to setup Java Database Connectivity (JDBC) to link the server with database.

We can use JDBC by downloading and importing the Jar file into the server program. Then, in the server program, we can import the related packages, shows in the figure below:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
```

**Figure 5 JDBC import**

Then we can read/write the database via Java in our Web Service. The code below shows how to connect to the database and make a basic select query.

```
Class.forName("com.mysql.jdbc.Driver");
Connection connection = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/ecgdatabase", "root", "ecg1234");
Statement statement = connection.createStatement();
String SQL= "select *";
statement.executeQuery(SQL);
```

## 2.2.4 Firewall (Iptables) configuration

The firewall is the most common network security appliance deployed at the edge of two networks. In this project, a standard Linux firewall—Iptables is used.

Iptables is a user-space application program that allows a system administrator to configure the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. To set up the Iptables, we can use the following commands:

Clear all the settings

```
1. iptables -L -n
```

Allow data input from port 22 and 80. 22 is the port for SSH while 80 is the port for our APIs.

```
1. iptables -A INPUT -p tcp --dport 22 -j ACCEPT
2. iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT
3. iptables -A INPUT -p tcp --dport 80 -j ACCEPT
4. iptables -A OUTPUT -p tcp --sport 80 -j ACCEPT
```

Block all other ports' input and output

```
1. iptables -P INPUT DROP
2. iptables -P FORWARD DROP
3. iptables -P OUTPUT DROP
```

Allow local visit

```
1. iptables -A INPUT -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
```

Allow all the visits called by the server and save all configuration

```
1. iptables -A OUTPUT -j ACCEPT
```

```
2. service iptables save
```

## 2.3 Interface Data Format

In order to make sure server and client can communicate, the first thing is to make sure both sides use the same data format.

There are several data format that be used to generate and parse the transmitted data. Server and client should parse the transmitted data into the same data format. The most two commonly used data formats are EXtensible Markup Language (xml) and JavaScript Object Notation (JSON). In this project, we use JSON which is provided by Jersey Framework as the data format.

JSON (JavaScript Object Notation) [4] is a lightweight data-interchange format which is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON[4] is built on two structures:

1. JSON Object: A collection of key/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. JSON Array: An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

In our web service, we take advantages of both JSONObject and JSONArray in different scenarios. JSONObject consists of sets of key-value pairs which are easy to serialize/deserialize and change. Here is an example of JSONObject:

```
{“username” : “liweizheng”,  
  “password” : “123456”}
```

However, JSONArray can be better to use in some situations. As JSONArray do not have keys, it can reduce the size of the transmitted package. For example, when the data is highly regular and we would use JSONArray to reduce the transmitting load.

In this project, we normally use JSONObject. However, for some APIs which transmits larger scale and repeated data, we store JSONArray in the JSONObject. Below is an example of the JSON package of API patients/gettests which is a JSONObject within which contains a JSONArray.

```
{ "result": "success",  
  "total_results": 2,  
  "results": [{  
    "id": 314,  
    "total_notes": 13,  
    "total_records": 9,  
    "created": "2015-06-07 13:30:00"}]}
```

### **2.3 Database Design and Architecture**

As we described previously, we choose to use MySQL as the backend database for the server. There are totally 9 tables in the database of the server which are patient\_info, doctor\_info, relation, access\_request, doctor\_comment, comment\_relation, ecgdatainfo, ecgtestinfo, mynote and note\_relatoin respectively.

We will discuss the role and relation of each tables here and the details of each table's structure are listed in Appendix A.

- Table patient\_info

This table stores the information of every specific user including the user's profile like username, user's phone number, user's email and user's doctor number, user's doctor request number and etc.

- Table doctor\_info

Like table patient\_info, this table stores the information of doctors. Each doctor has one row in the table.

- Table relation

This table stores the relation between doctor and patient. When a patient and a doctor are connected, this relation information will be stored in this table. The details of the relation management will be introduced in Section 2.5 Patient/Doctor relation and permission management.

- Table access\_request

This table stores the request information. When a doctor wants to build the relation with a patient, before being approved, the request information will be stored in this table.

- Table doctor\_comment

This table stores the information of all the doctor's comment including the comment content, comment time, doctor's id, patient's id and etc. Details will be introduced in Section 2.8 Doctor Comments.

- Table comment\_relation

As a doctor's comment may be related to a specific ECG record, this table stores the information of relationship between doctor's comment and ECG record.

- Table ecgdatainfo

This table stores the information of each ECG data. As our ECG raw data is stored in the hard disk, the position of each ECG data is stored in this table.

- Table ecgtestinfo

This table stores the information of each ECG test.

- Table mynote

This table stores the information of patient's note. Details of this part will be introduced in section 2.9 Patient Notes.

- Table note\_relatoin

As a patient's note is related to several ECG records or a specific time, we need this table to store the relationship between them.

## **2.4 Patient/Doctor authentication**

Authentication is the act of verifying a user's identity. User's reliable and safe authentication is important to prevent the system from being attacked or hacked. For our system, we verify user's identity by username and password via login interface. After receiving user's username and password, the server will first confirm the correctness of

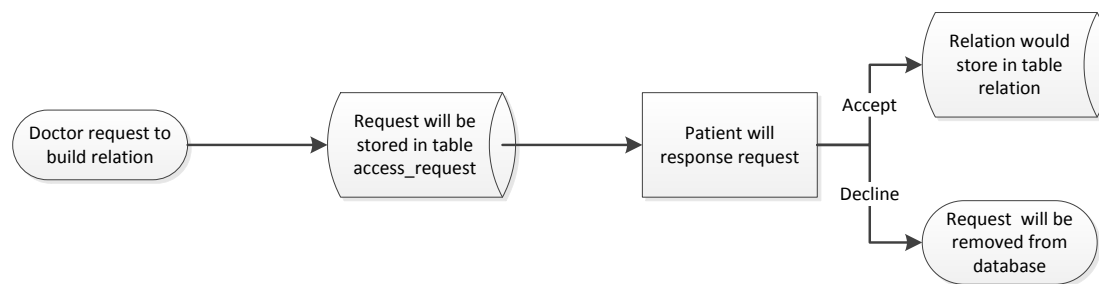
username and password. If the matching of username and password failed, the server will return the failure information.

For web site user, after the verification of the username and password, the server will send back a unique session\_id which can be used for all the further authentication. The session\_id and the userid are stored in a Hashmap so that server can get user's userid with session\_id. The session\_id is stored in the memory of the server and would expire in a given time. With the unique session\_id and expire time, it's reliable and efficient to identify a user's identification without passing password anymore.

For mobile APP user, we use the similar method as session\_id. However, instead of having an expiration time, mobile users may be logged in for a long time period. Thus, after verification of username and password, we provide a dynamic\_id which is a 64 bits UUID to specific users. This dynamic\_id will be updated every logins.

## **2.5 Patient/Doctor relation and permission management**

In the ECG System, as the role of patients and doctors have been involved, it is necessary to have logics to handle the relation between patients and doctors. To protect the privacy of patients, only after the built of relation between patient and doctor, doctor can have permission to access the patient's ECG data and comment on patient's ECG data. The workflow of patient/doctor's relation management is described as Figure6:



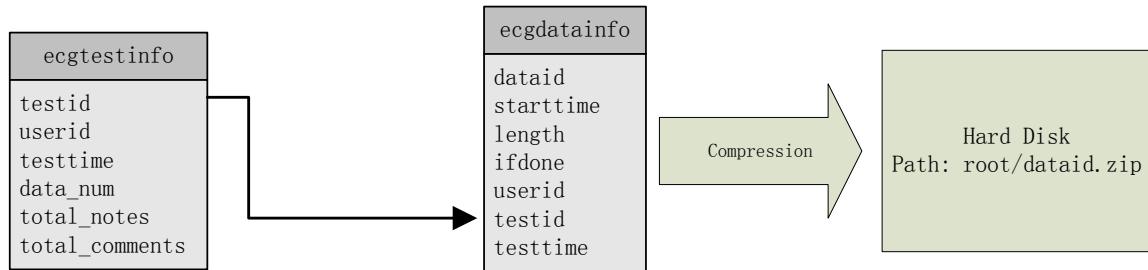
**Figure 6 Work flow for Patient/Doctor Relation Management**

To be better managed, only doctors are permitted to request to build the relation. By sending the request through API root/doctor/accessrequest, the server will receive and store the request information. Next time the user login, the user can choose to accept/decline the request through API root/doctor/confirmrequest. If accepted, the information of user and doctor' relation will be stored in table relation. Otherwise, if the user declines the request, the requests will be deleted from database.

## 2.6 ECG data storage and management

To allow doctors or patients to monitor patient's ECG signal constantly, long term ECG monitoring (will be introduced in detail in Chapter 3) has been implemented on the Mobile APP. However, with the increase of the length of ECG data, the size of the ECG data would be very large. A one hour ECG data can be more than 500KB which will results in data loading unacceptably slow. Also, too long ECG data are not easy to read. Thus, in our ECG system, we introduce the concept of test and record. Each long term ECG monitoring is called one test while every test is separated into several records which are up to 10 minutes each.

The logic of ECG data storage is described in Figure7:



**Figure 7 ECG Data Storage**

Each test would have a row in table `ecgtestinfo` including information like the `testid` which is a unique identifier for each test, `data_num` which specifies how many record (record is the same concept of data in database).

All the ECG data/record will be uploaded from Mobile APP to Web Service through API `root/mobile/update`. After receiving data, if this data/record is the first of a test, the Web Service will create a new row in `ecgtestinfo`. Otherwise, the Web Service will update the `data_num` of regarding `testid`. After that, the Web Service will create a new row storing the information of the ECG data in `ecgdatainfo`, setting `testid` to be the `testid` we got from `ecgtestinfo` so that the user can retrieve the all the ECG data belonging to a specific test.

After storing information in the database, to make the database more efficient, we will store the raw data into the hard disk of server. As ECG data is highly regular, some compression packages can have compression rate more than 70% on the ECG data. Thus, we store the ECG data in compressed file name with `dataid.zip` so that it can be easy to retrieve. The file name of the ECG data is `'data_id.bin'` so that server can retrieve the ECG data file according to the ECG data id.

Besides storing the ECG data locally, the server would run an ECG analysis program, calculating the important parameter of the ECG data including heart rate, QRS duration, QT interval, PR interval and QTC's value. Those data are also saved in the `ecgdatainfo` table for further retrieval.

When a patient or the connected doctor wants to view this patient's history data, firstly the `/gettests` API will be called which returns the list of ECG tests of the specific patient.

The figure below is the test list on the web site.

Show <input type="text" value="10"/> entries						Search: <input type="text"/>
ID	Records	Comments	Notes	Length	Created	
41	1	0	1	1 minute 23 second	2016-03-24 01:39:20	
40	1	0	0	33 second	2016-03-21 06:01:41	
39	1	0	3	2 second	2016-03-17 03:09:59	
38	1	0	0	4 second	2016-03-17 03:00:33	
37	1	0	1	3 second	2016-03-17 02:39:25	
36	1	0	0	36 second	2016-03-14 11:26:41	
35	1	0	10	41 second	2016-03-14 03:30:02	
34	2	0	0	3 minute 3 second	2016-01-08 03:09:04	
33	1	0	0	15 second	2015-12-17 11:24:52	
32	1	0	0	9 second	2015-12-17 11:23:26	

Showing 1 to 10 of 43 entries

Previous      Next

**Figure 8 ECG Test List on Web Site**

If user selects one test, the web site or mobile APP would visit the `/test` API with the id of the specific test to get the ECG record list of this test.

After getting the list of all the records of the test, the user may want to view a record's ECG data. Then the web site or mobile APP would visit the `/record` API with the id of the specific ECG record to get the content of the ECG data. With the ECG data, web site and mobile APP will be able to display it. The Figure 9 is an example of ECG data displaying on the web site.



**Figure 9 ECG Data Displaying on Web Site**

## 2.8 Doctor Comments & Patient Notes

Communications between patients and doctors are important to let doctor have better control of patient's situation and let patients know more about their own condition. Based on the ECG monitoring, our system provides users with doctor comment function and patient notes function.

### 2.8.1 Patient Notes

For our ECG monitoring system, we want to provide the function that a patient can write some notes when testing the ECG data or feeling uncomfortable. For example, when a user feels very uncomfortable and want to record his/her ECG data, this user may

also want to write down a note describing when and how this symptom happens. It is also possible that users may want to add some supplementary information the next day. Thus, the patient's note is related to some ECG records or a specific time.

In the database, there are two tables storing the patient note, `mynote` and `note_relation`. The table `mynote` stores the note's content, note's creation time and the event's time. As the relationship between patient note and record is one to multiple, we need another table, `note_relation` to store the relationship between the note and the ECG record.

Shown in Table 2, the Web Service provides four APIs related to the patient notes to support the clients.

Description	API URL	Method
Leave a note	<code>root/note</code>	POST
Get a note	<code>root/note/{id}</code>	GET
A patient's all notes	<code>root/patient/{id}/getnotes</code>	GET
The notes of all the patients who are connected to a specific doctor	<code>root/doctor/{id}/getnotes</code>	GET

**Table 2 APIs for Patient Notes**

When a patient wants to write a note, the client would call the URL `root/note`, providing the content of the note, `patient_id` and the `event_time`. And the API `root/note/{id}` can enable the client to find a note's information with the id of the note.

The API `root/patient/{id}/getnotes` can be used to get all notes of a patient by providing the patient's id. What needs to be explained is that, only the patient him/her self or his/her

doctors have permission to get data from this API call. Otherwise, the patient's privacy would be invaded.

The API *root/doctor/{id}/getnotes* returns all of a doctor's patients' notes.

### 2.8.2 Doctor Comments

For doctors, leaving some comments after reviewing a patient's ECG data can be really useful for the patient's further treatment. Also, as some patients would write something about their symptoms in patient's notes, doctors may want to write some suggestions back. Thus, the software adds the functionality that the doctor can write comments to either ECG record or a patient note.

In the database, we have two tables storing the doctor comment's information, table *doctor\_comment* and table *comment\_relation*. The table *doctor\_comment* stores the information of the comment itself including the content, doctorid, userid. And the table *comment\_relation* stores the one to multiple relation between doctor comment, patient note and ECG record.

To support this function, as shown in Table 3, the ECG Web Service provides four APIs related to this function:

Description	API URL	Method
Leave a comment	root/comment	POST
Get a comment	root/comment/{id}	GET
A patient's all comment	root/patient/{id}/getcomments	GET
A doctor's all comment	root/doctor/{id}/getcomments	GET

**Table 3 APIs for Doctor Comments**

When leaving a comment, the client should visit *root/comment* and provides the server with the content of the comment, the id of the doctor, the id of the patient and the id of the ECG's record which the comment is related to. By visiting the API *root/comment/{id}*, the client can get the comment's detail of a specific id.

The API *root/patient/{id}/getcomments* returns all the comments given to a specific patient while *root/doctor/{id}/getcomments* returns all the comments given by a specific doctor. All those three GET methods have proper permission management to protect user's privacy and security.

## 2.9 Search API

In our web site, we support the search function which enable patients or doctors to search for ECG tests with some optional limits. The search test API takes several optional parameters including the time period, whether the test should be with doctor comment, whether the test should be with patient note. These optional parameters can be taken at same time to limit the range of result. The server responses with all the ECG tests that meet the requirement.

This API is used for both doctors and patients. The search API for the patient and doctor is *root/search/gettests*. The identity and the role can be determined by *session\_id*. If the doctor visit the API, the doctor can choose to get the ECG tests of all of his connected patients or a specific patient. For patient users, they can only retrieve their own ECG tests. This design is actually the typical rest style as the API is only related to the resource (ECG tests) rather than who visits or when the visit is.

The Figure 10 shows the search panel for patients on web site.

**Figure 10 ECG Test Search Function on Web Site**

If the user is a patient, then the server will call to search for the ECG test of the patient with the given limitation in the database. If the user is a doctor, the server will search for all of his patients or the specific one in database with given limitation.

## 2.10 Overview of APIs

Here is the overview of all the APIs provided by the Web Service and their functions.

All the interfaces are based on the root URL: `ecg.ece.uvic.ca/rest/` and the port is 80.

API	Description	Method
<code>/patient/login</code>	For patient's login	POST
<code>/patient/register</code>	Let patient sign up account	POST
<code>/patient/logout</code>	Let patient log out	POST
<code>/patient/confirmrequest</code>	Let patient response to a doctor's access request	POST

/patient/viewed	Update patient's view status	POST
/patient/{id}	Get the basic information of the patient account. Permission only for patient and his/her doctors.	GET
/patient/{id}/gettests	Get all tests of a patient. Permission only allowed for patient and his/her doctors.	GET
/patient/{id}/getdoctors	Get all doctors' info of a patient. Permission only allowed for the patient him/her self.	GET
/patient/{id}/getcomments	Get all the comments given to a patient. Permission only allowed for the patient him/her self.	GET
/patient/{id}/getnotes	Get all the notes of a patient. Permission only allowed for the patient him/her self.	GET
/doctor/{id}/login	For doctor's login	GET
/doctor/register	Let doctor sign up account	POST
/doctor/logout	Let doctor log out	POST
/doctor/accessrequest	Send an access request to a patient	POST
/doctor/viewed	Update doctor's view status	POST
/doctor/{id}	Get the basic information of a doctor. Permission only allowed for doctor him/her self and their patients.	GET
/doctor/{id}/getpatients	Get all the patients of a doctor. Permission only allowed for doctor him/her self.	GET

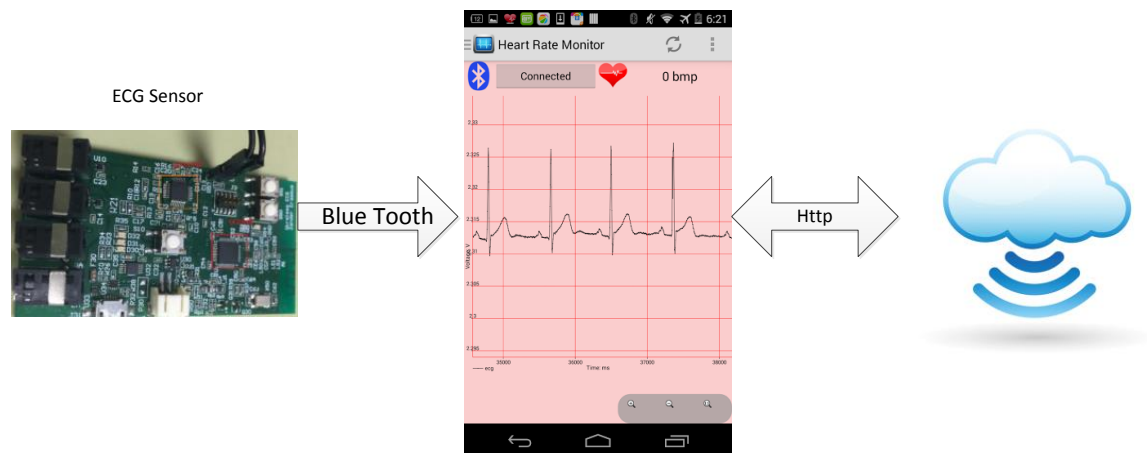
/doctor/{id}/gettests	Get all ECG tests of the patients of a doctor. Permission only allowed for doctor him/her self.	GET
/doctor/{id}/getcomments	Get all comments given by a doctor. Permission only allowed for doctor him/her self.	GET
/doctor/{id}/getnotes	Get all the notes of connected patients of a doctor. Permission only allowed for doctor him/her self.	GET
/test/{id}	Get the detail of an ECG test. Permission allowed for the patient and his/her doctors.	GET
/record/{id}	Get the detail of an ECG record. Permission allowed for the patient and his/her doctors.	GET
/note	Let patient leave a note.	POST
/note/{id}	Get the detail of a patient note. Permission allowed for the patient and his/her doctors.	GET
/comment	Let doctor to leave a comment	POST
/comment/{id}	Get the detail of a doctor comment. Permission allowed for the doctor and the patient.	GET
/search/gettests	Search for the ECG tests with given limitations.	GET
/updata	Upload ECG data received from the ECG monitor via BLE	POST

## Chapter 3 Mobile Client App Design and Implementation

### 3.1 Introduction

In this chapter, we elaborate the mobile APP's design and implementation.

The most important function of mobile APP is to receive ECG data from the ECG sensor and transmit the ECG data to the Web server. The Figure 11 illustrates the work flow.



**Figure 11 Work Flow of the ECG Monitor System**

The mobile APP contact the ECG sensor through Bluetooth Low Energy which is the most ideal protocol of sensor-smart phone systems. After connected, the ECG sensor will send data package in every fixed time period. The mobile APP can receive the data and, after parsing data, display the data on the mobile phone. Along with ECG data displaying, the mobile APP would also calculate real time heart rate and detect the Ventricular tachycardia / Ventricular fibrillation in real-time.

Besides displaying ECG data, the mobile APP provides two types of ECG tests. The first type is long term ECG monitoring which can record a patient's ECG data with no time limits. Those ECG data are stored locally and uploaded to the server once the WIFI is connected. Once the data is uploaded to the Web server, the patient and his/her doctors can view the ECG data on web site. The long term ECG monitoring is very useful for those diseases which happen randomly and is hard to capture when patients test ECG data in the hospital. The second type is quick test which is designed for patient's self-detection. The quick test function allows patients to record their ECG data for one minute and give back a quick test report after one-minute test which have the important parameters of this one-minute ECG data. This can help patients to be quickly aware of their heart condition.

Next we discuss the details of each function on mobile APP. As the work is based on a previous student's work, we only introduce the new functions.

## **3.2 ECG Data Receiving via Bluetooth Low Energy**

### **3.2.1 Bluetooth Low Energy**

BLE is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group (SIG) for short-range communications [5]. The advent of BLE has occurred while other low-power wireless solutions, such as ZigBee, WiFi, have been already gaining a certain momentum in the application domain. In the

following few subsections, a brief introduction of BLE protocols is firstly presented, which are based on the popular handbook of BLE [6].

Android 4.3 [7] (API Level 18) introduces built-in platform support for Bluetooth Low Energy in the *central role* and provides APIs that apps can use to discover devices, query for services, and read/write characteristics. In contrast to Classic Bluetooth, Bluetooth Low Energy (BLE) is designed to provide significantly lower power consumption. This allows Android apps to communicate with BLE devices that have low power requirements, such as proximity sensors, heart rate monitors, fitness devices, and so on.

In our project, we use Bluetooth Low Energy to transmit ECG data from ECG monitor and Android mobile phone.

### 3.2.2 Android API for BLE

Android has provided the API of Bluetooth Low Energy for developers after API 18 so that we can use the BLE on Android phone.

There are several steps showing how Android APP works with other BLE device:

1. Add permission to APP manifest so that the APP can visit BLE service.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-feature android:name="android.hardware.bluetooth_le"
android:required="true"/>
```

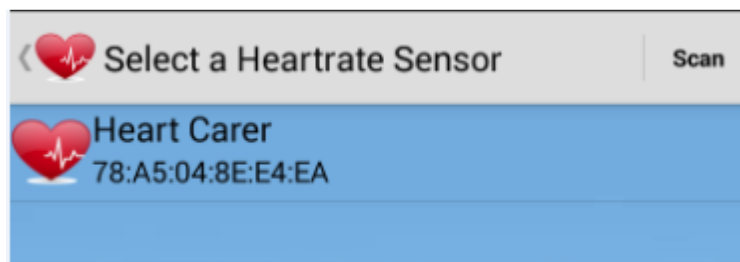
2. Scan BLE devices with function: `BluetoothAdapter.startLeScan(LeScanCallback)`.

There are three modes of scanning which are low power, balanced and high power.

In our project, we use the default mode which is low power model.

3. Select a BLE device from the device list and connect with function `connectGatt()`.

As shown in the Figure 12, the ECG sensor was detected by the Android phone and we can select it to connect the sensor.



**Figure 12 ECG Sensor in BLE Device List**

4. Then we can read from and write to the ECG sensor.

### 3.2.3 Data Format and Data Rate

The maximum transmitting rate for BLE is 2k bytes/second. However, after testing, when setting the transmitting rate to be 2k bytes/second, it's highly likely to lose the packages. If the package loss happens, it would be a disaster for our ECG system as we can hardly determine whether the abnormality of an ECG signal is caused by the patients' heart disease or the package loss. Thus, we finally use 1k bytes/second as the transmitting rate. The ECG monitor sends to the mobile phone every 20 millisecond while each package contains 20 bytes ECG data.

In our ECG system, the ECG monitor can support at most two channels. Thus, the received ECG data contain two channel's data as well as lead off status. There are four sample points of ECG data in each package and each point is five bytes. The first byte indicating the lead-off status. The second byte and the third byte represent the first channel's ECG data and the last two byte form the second channel's ECG data. For each

channel, the first byte is the first eight bits of ECG data and the second byte is the lower eight bits. The transmission can be described by the formula:

$$\text{value} = (\text{first byte} \ll 8) + \text{second byte}$$

An example of an ECG data package can be seen in the table below:

Byte Number	Value	Description
0	00	Lead-Off status
1	D2	ECG Sample1: Channel 1 ADC High byte
2	C0	ECG Sample1: Channel 1 ADC Low byte
3	D0	ECG Sample1: Channel 2 ADC High byte
4	39	ECG Sample1: Channel 2 ADC Low byte
5	00	Lead-Off status
6	D2	ECG Sample2: Channel 1 ADC High byte
7	6D	ECG Sample2: Channel 1 ADC Low byte
8	CF	ECG Sample2: Channel 2 ADC High byte
9	D1	ECG Sample2: Channel 2 ADC Low byte
10	00	Lead-Off status
11	D2	ECG Sample3: Channel 1 ADC High byte
12	78	ECG Sample3: Channel 1 ADC Low byte
13	CF	ECG Sample3: Channel 2 ADC High byte
14	F2	ECG Sample3: Channel 2 ADC Low byte
15	00	Lead-Off status
16	D2	ECG Sample4: Channel 1 ADC High byte
17	CC	ECG Sample4: Channel 1 ADC Low byte

18	D0	ECG Sample4: Channel 2 ADC High byte
19	76	ECG Sample4: Channel 2 ADC Low byte

### 3.3 Real Time ECG Signal Displaying

#### 3.3.1 ECG Data Parsing and Displaying

According to Android BLE API, every time the mobile APP received a new ECG package transmitted via BLE, the callback function `onCharacteristicChanged` in class `BluetoothGattCallback` would be triggered. The code below shows how we parse the received ECG data package into the data structure needed. As introduced before, each ECG data package is consists of 20 bytes which are actually four ECG points of channel one and four ECG points of channel two.

```
public void onCharacteristicChanged (BluetoothGatt gatt,
    BluetoothGattCharacteristic characteristic){

    byte[] notiValue = characteristic.getValue(); // notiValue is received
package

    int[] multiValue = new int[8]; // store the parsed data into a int array

    for(int i=0;i<8;i=i+2){

        int index = i*5/2;

        index++;

        multiValue[i] = notiValue[index++]; // higher 8 bites of channel one

        multiValue[i] = multiValue[i] << 8;

        multiValue[i] += (notiValue[index++] & 0xFF); // the ECG point's value
of channel one

        multiValue[i+1] = notiValue[index++]; //higher 8 bits of channel two

        multiValue[i+1] = (multiValue[i+1] << 8);
```

```

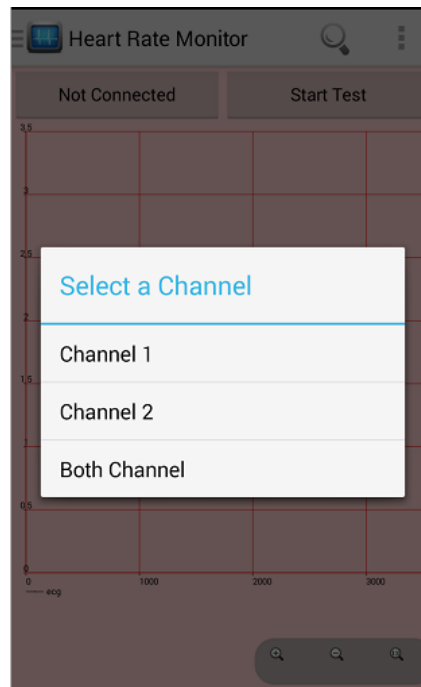
        multiValue[i+1] += (notiValue[index++] & 0xFF);} // the ECG point's
value of channel two
    }

```

After parsing, we now get an int array which contains the ECG points' value of two channel. Then this array is sent to the ECG signal displaying fragment.

### 3.3.2 Single/Double Channel Displaying

Generally, the ECG test data in hospital have 12 channels which are collected from the electrodes placed on different parts of a human body. As a wearable monitor, our ECG monitor system supports up to two channels which can have most of the information of a patient's ECG data. Because our ECG data have two channels, our mobile APP support single channel as well as two channels display. Patients can choose to view both channels as well as a specific channel when testing. The Figure 13 shows the channel selection menu.

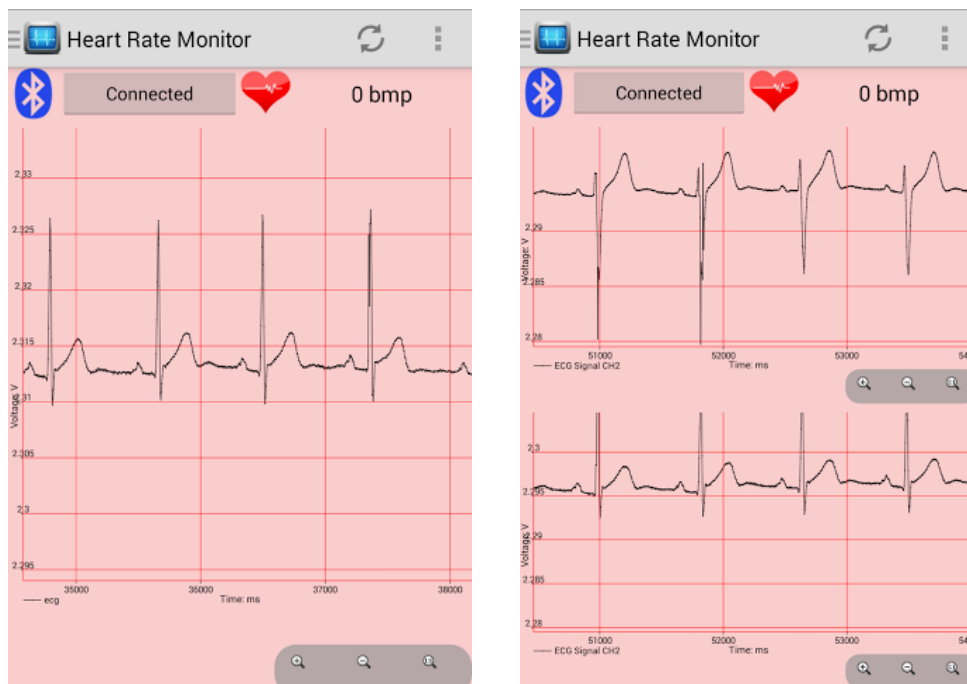


**Figure 13 Channel Selection Menu**

To display ECG signal on a smartphone, we need to use the chart engine. There are many chart engines for Android. In our project, we choose to use AChartEngine which is a well optimized open source framework to draw line chart. There two important classes to use AChartEngine, `XYMultipleSeriesDataset` and `XYMultipleSeriesRenderer`. The first class defines the data source of the chart and the second class defines how the chart should be like.

When we initialize the chart, we need to use `XYMultipleSeriesRenderer` specifying the format of the chart including the chart's size, color, line width, display range and etc. After the chart has been initialized, each time we receive a new data, we add the new data into `XYMultipleSeriesDataset` and call the `repaint()` function to update the new data.

Figure 14 shows the example of signal channel and two channels display.



**Figure 14 Single and Double ECG Data Displaying**

### **3.3.3 Real Time Heart Rate and Ventricular tachycardia / Ventricular fibrillation monitoring**

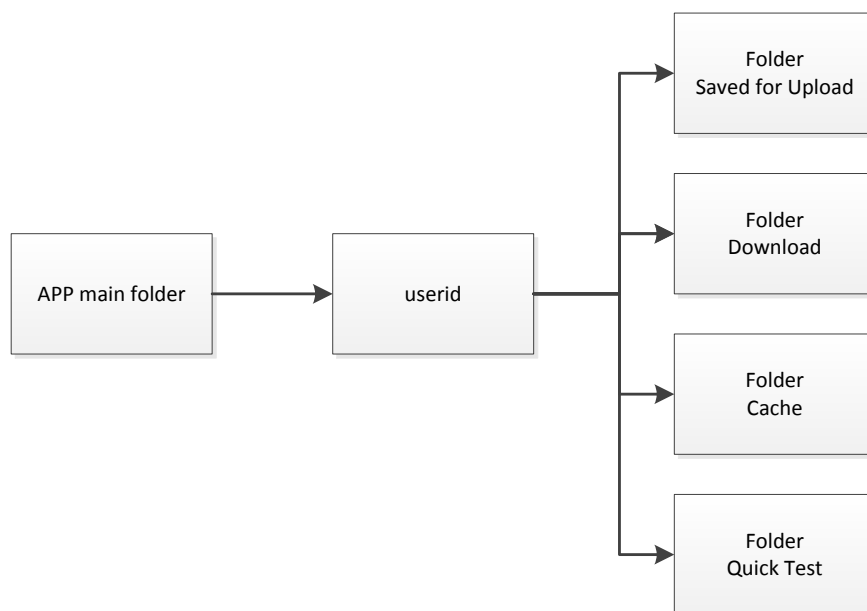
Along with the real time ECG data display, the mobile APP provides the real time heart rate detection and Ventricular tachycardia (VT) / Ventricular fibrillation (VF) monitoring. VT and VF are two types of heart tachycardia which are extremely dangerous events for people.

The algorithms of the real time heart rate calculation is based on Tianyang's program who is a Master's student in the ECG development groups. The VT/VF detection algorithm is based on Ping Cheng's program who is a PhD student in the group.

The real time detection is based on 15 seconds' ECG data. In the APP, there is a cache storing the ECG data for detection. When the mobile APP has collected ECG up to 15 seconds which are 3750 ECG data points, the APP will run the heart rate detection functions and clear the cache. The value of the heart rate will be updated on the screen. And when the VT/VF is detected, there will be a notification message displaying on the screen as well.

### **3.4 Local Storage Management**

In the mobile phone's local memory, we store some ECG data, as shown in the Figure 15. There is a folder for the mobile APP. Inside the main folder, we create a single folder for each user specified by the user's userid. However, in most cases, a mobile phone may have only one user.



**Figure 15 Local Storage Folder Structure**

In the folder of each user, there are four folders storing ECG data: “Saved for Upload”, “Download”, “Cache” and “Quick Test”.

Folder “Saved for Upload” is used for storing the ECG data received from the ECG monitor but has not been uploaded to the server.

Folder “Download” is used for storing downloaded the ECG data from the server.

Folder “Quick Test” is used for storing the ECG data for quick test.

Folder “Cache” is used to store the cache file when receiving ECG data from the ECG monitor. These data will be moved to “Quick Test” or “Saved of Upload” once the transmission finished.

Every time a user login, the APP firstly calculates the size of the APP’s whole folder. Once the APP found that the APP takes more than 500 MB in mobile storage space, it would trigger the cleaning program. The cleaning program firstly deletes all the files in the “Download” folder which is the most unimportant data as those data can be

downloaded again from the server in the future. Next, the data in the “Quick Test” will be deleted. At last, if the storage usage is still higher than 500 MB, we start to delete ECG data in “Saved for Upload”.

During the cleaning processes, once the cleaning program find that the usage of mobile local storage takes smaller than 500 MB, it will stop the process and load to the main activity.

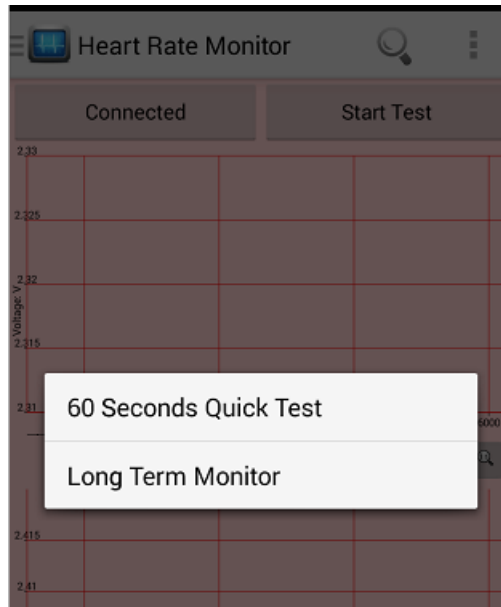
### **3.5 ECG Test**

To better help patients and doctors, our mobile APP provide two kinds of ECG tests, quick test and long term test.

Quick test is one minute testing and a report will be generated from the ECG signal processing algorithm on the APP. Quick test can let patient to know their ECG basic condition in one minute.

However, the more significant use of the system is that doctor can know more about the patient’s long term ECG changes. Thus, long term monitor is included. Long term test has no time limit and all the ECG data are saved locally or on cloud. The patient’s doctor has permission to view those data on their end.

After the ECG sensor is connected, we can start a test by clicking button ‘Start Test’. Then we can choose to start a ‘60 Seconds Quick Test’ or a ‘Long Term Monitor’.

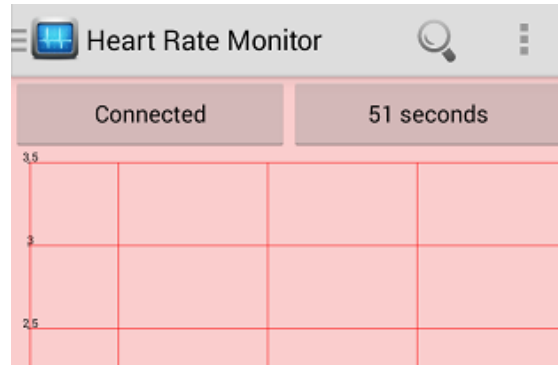


**Figure 16 ECG Test Selection**

### **3.5.1 Quick Test**

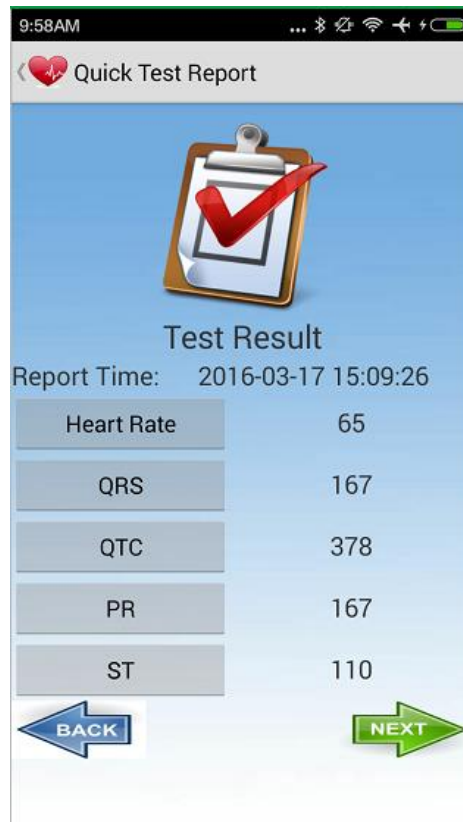
As introduced, quick test is for a user to quickly self check. It will record ECG data for one minutes and generate a report. The quick test report is calculated by the signal processing algorithm in the APP which contains 5 important ECG parameters: heart rate; QRS; the corrected QT interval (QTC), the PR interval (PR), ST segment.

After the quick test starts, a 60 seconds countdown will appear on the origin 'Start Test' button. After 60 seconds, the APP will start to analyse this one-minute ECG data.



**Figure 17 Quick Test Count Down**

Then after analysis, the APP will show that the data can not be analyzed if the data is too noisy. Otherwise, the APP will generate a quick test report. The Figure 18 below is an example of a quick test report. All five parameters and report time are listed and we can view the previous and next quick test report by clicking back and next button. The storage of history report will be described in Section 3.6.



**Figure 18 A Example of Quick Test Report**

### 3.5.2 Long Term Monitor

The other important feature for our ECG monitor system is long term ECG monitoring. As most anomalies of ECG signals happen randomly, these abnormal episodes are hard to catch during regular short ECG tests in hospitals. Thus, long term ECG monitoring can be very helpful for patients and doctors.

When a user chooses long term monitor, the APP will start to store the ECG data. As the long term monitoring has no time limits and it could be even as long as 24 hours or more, it is not convenient to read if storing them in one single file. Thus, as introduced in the web service part, we separate one ECG test in to several records and each test is up to 10 minutes long.

All the ECG records are saved in binary bin files in the “Saved for upload” folder in the mobile phone.

Besides the ECG data itself, there are some extra information we need to store including the record’s start time and the whole test’s start time. Also, we need to know which ECG test this ECG record belongs to. We take advantage of the file name to store such information.

filename = test time + record time + if first record in the test

One example of the file name can be “2016032415193320160324150933 0.bin” which means this ECG record start at 15:19:33 24th March 2016 and the whole test time start at 15:09:33 24th March 2016. The last digit 0 means this is not the start of the whole test.

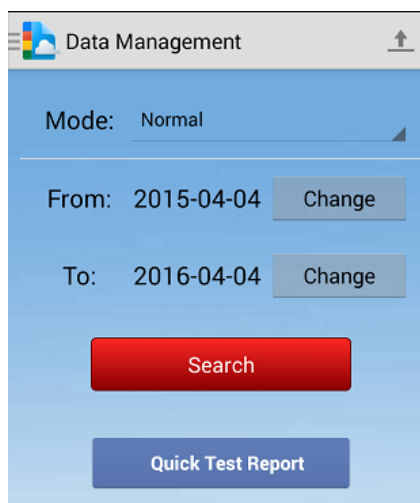
We only use WiFi to upload the ECG data, considering the ECG data can be overwhelming in size if using cellular data. However, when needed, it could be easily setup to use the cellular data communication to upload the ECG data in the Android code. Once the bin file is saved, if the WiFi is connected, the APP will upload the file to the server. If the WiFi is not available, the APP will upload every bin file in the “Saved for upload” next time when the WiFi is connected. Before sending the file, the APP firstly compresses the bin file into zip file to reduce the size of transmitting package.

On the server side, by knowing the user’s identification, the test’s start time and the record start time, the server can insert the related information in the table ecgdatainfo and table ecgtestinfo and the file will be saved in the server storage.

After the transmission is finished, as the data has already been stored on the cloud, the file in the “save for upload” will be deleted to save the memory on the client side.

### 3.6 History ECG Data Management

The ECG data management is in the ‘Data Management’ option in the main menu, as shown in Figure 19. To search for a long term test’s record, users can choose a time interval and click red ‘Search’ button. The blue button is used for view history quick test report.



**Figure 19 Data Management Option**

#### 3.6.1 Quick Test Report History Management

Each time after quick test, we want to store the quick test report so that the user can have a continuous monitor of how his/her ECG condition changes.

To store the quick report locally, we choose to build a local database on the Android phone – SQLite as saving data to a database is ideal for repeating or structured data. SQLite is a lite SQL database on Android phone and Google has provided with API for Android developers to access the SQLite [8]. Just like files that be saved on the device's internal storage, Android system stores database in private disk space that is

associated with an application. The data is secure, because by default this area is not accessible to other applications.

The code below defines the structure of the quick test report table:

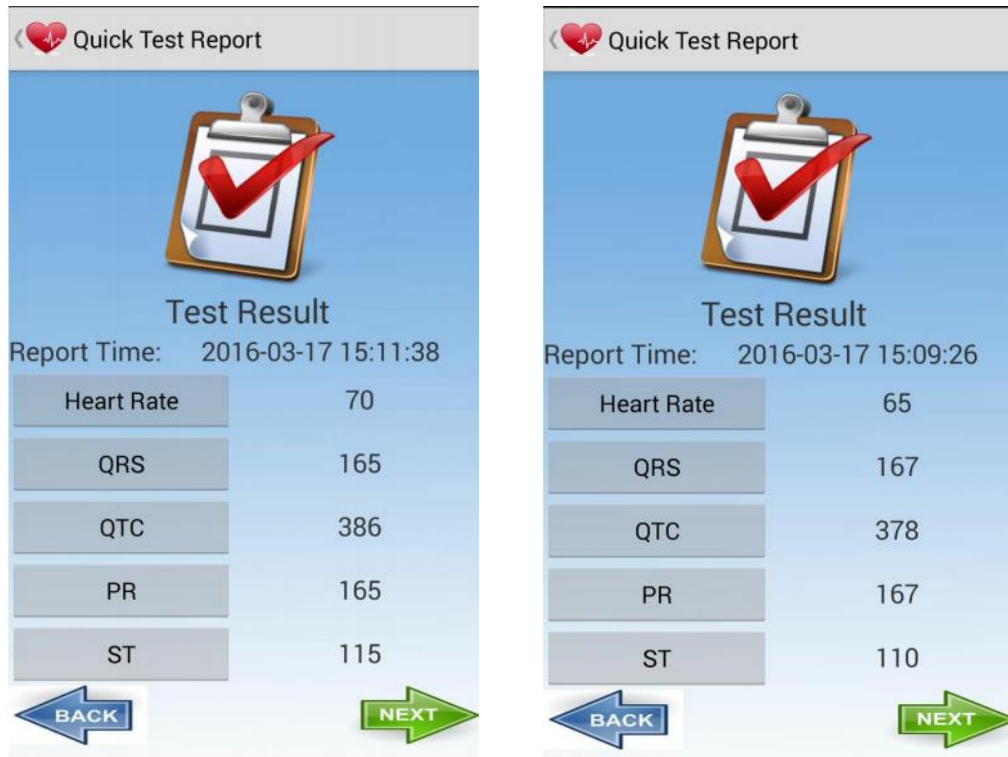
```
public static abstract class QuickTestReport implements BaseColumns {
    public static final String TABLE_NAME = "quick_results";
    public static final String COLUMN_NAME_UserID = "UserId";
    public static final String COLUMN_NAME_TestTime = "TestTime";
    public static final String COLUMN_NAME_HR = "HR";
    public static final String COLUMN_NAME_QRS = "QRS";
    public static final String COLUMN_NAME_QTC = "QTC";
    public static final String COLUMN_NAME_PR = "PR";
    public static final String COLUMN_NAME_ST = "ST";
    public static final String COLUMN_NAME_data_name = "DataName";
}
```

Once the structure database has been defined, a useful set of APIs is available in the SQLiteOpenHelper class. When using this class to obtain references to database, the system performs the potentially long-running operations of creating and updating the database only when needed and not during app startup. All we need to do is call `getWritableDatabase()` or `getReadableDatabase()`. To use SQLiteOpenHelper, create a subclass that overrides the `onCreate()`, `onUpgrade()` and `onOpen()` callback methods.

The code below shows how to create the database with the structure we defined above.

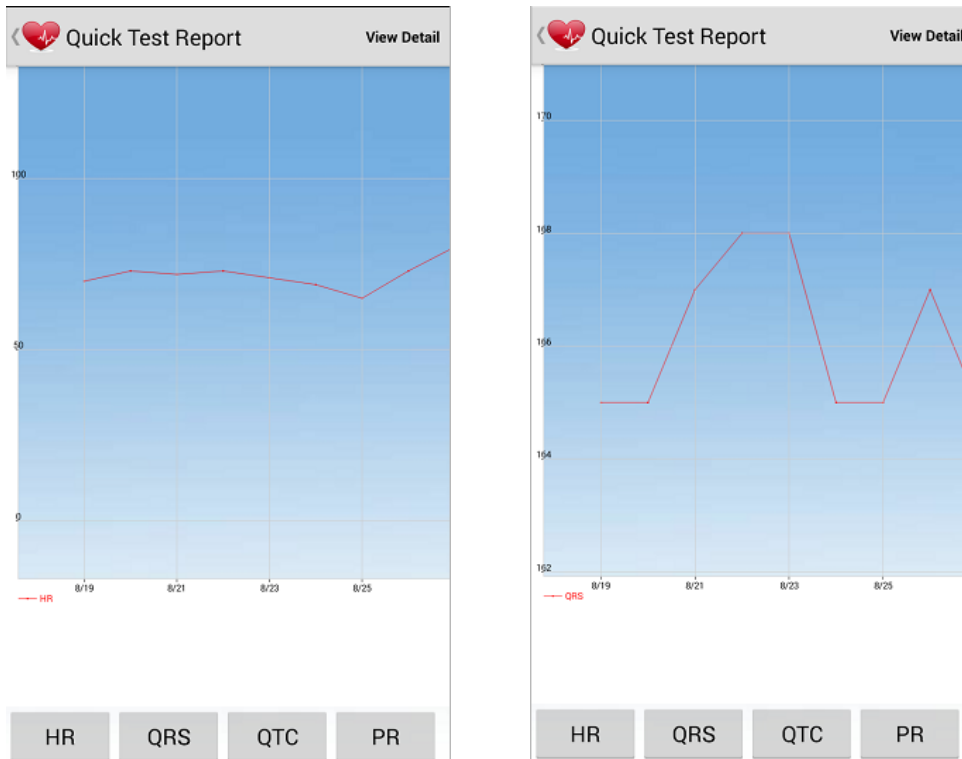
```
public void onCreate(SQLiteDatabase db) {
    db.execSQL(SQL_CREATE_ENTRIES);
}
```

Each time a new quick test report is generated, a row will be inserted into the table which contains the information of the quick test. The Figure 20 shows a few ECG quick test reports. Patients can view other history reports by clicking “BACK” and “NEXT”.



**Figure 20 Examples of Quick Test Report**

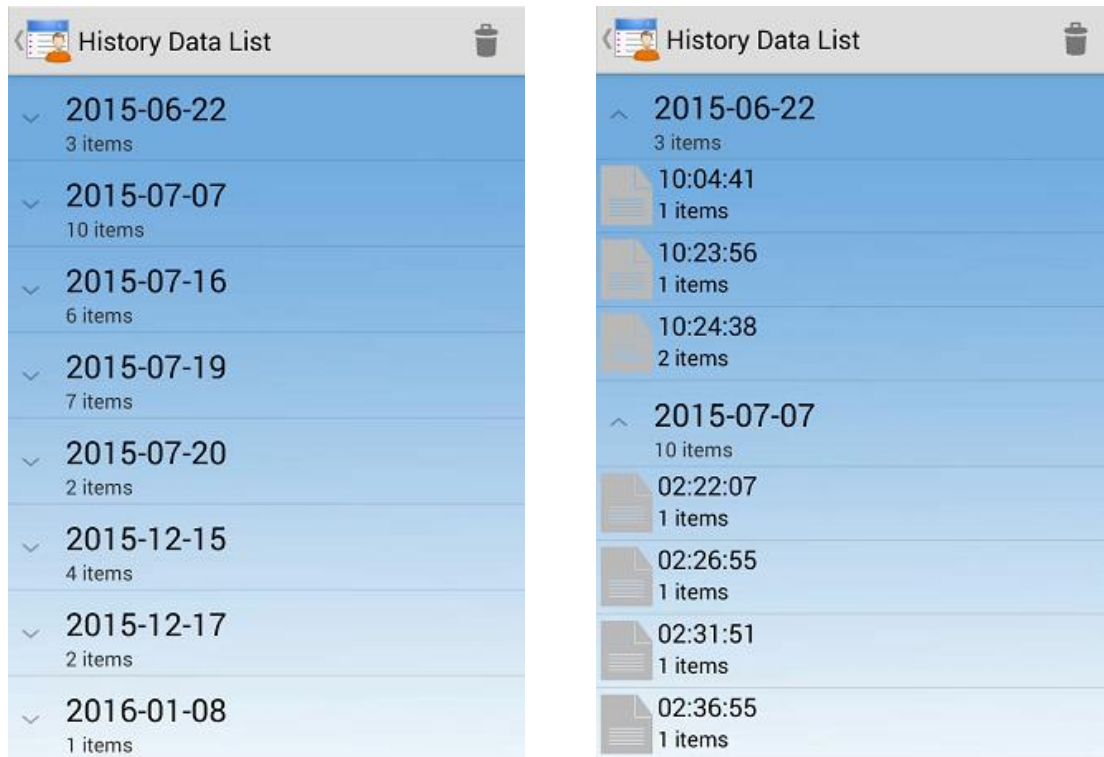
Also, to help a patient to have a clearer understanding of his/her heart condition, we provide a trend line chart showing how heart rate, QRS, QTC, PR change in the passed time. The Figure 21 is an example of a trend line of heart rate's and QRS's change in a few days. Users can click on different button in the bottoms to view other parameters' chart.



**Figure 21 ECG Parameter Trend Line Chart**

### 3.6.2 Long Term ECG Data Management

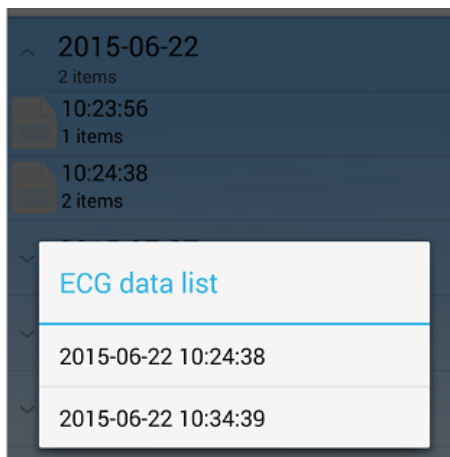
After we select a time interval and click the 'Search' button, the APP will send request to web service to retrieve the list of history ECG tests within the given time frame. Then the history data list is displayed, shown in the left of Figure 22. Each cell of the list represents one day and the number of tests in a specific day shows under the date.



**Figure 22 Long Term ECG Data Management**

As discussed above, because the long term ECG monitor does not have time limit and too long ECG data is not convenient to read for users, we separate one single ECG test into several ECG records. Each ECG record can be up to 10 minutes.

If the user click into a day on the list, then a list of all the ECG tests performed on the specific day will show up. Each cell in the expanded list represents one ECG test. If we click into one ECG test, a dialog will show up which list all the ECG records within the specific ECG test, shown in Figure 23.



**Figure 23 ECG Record List**

When the user chooses a specific ECG record, the APP will firstly search on the local storage to see if the data is already downloaded. If not, the APP will send a request to the server to download the ECG data from the server. An example of history ECG record is given in Figure 24.



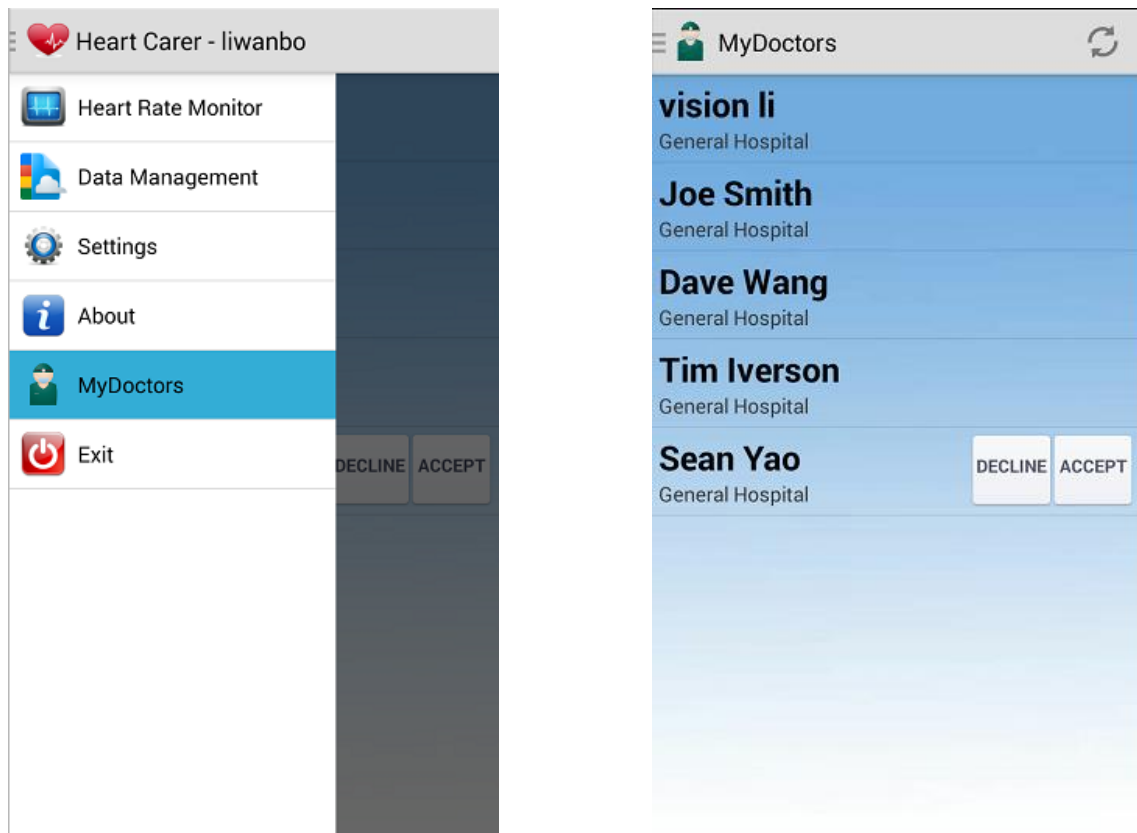
**Figure 24 History ECG Data Plot**

### 3.7 Doctor / Doctor's Comment Management

As explained in Chapter 2, our ECG monitor system is doctor involved. Since the APP is only designed for patients, the doctor management and doctor comments management functions are supported in APP.

#### 3.7.1 Doctor Management

Each patient can have multiple doctors and all the data of this patient can be accessed by his/her connected doctors. The data includes the patient's own profile, patient's note and patient's ECG test data. In the mobile APP, there is a 'MyDoctors' option in the main menu, as shown in the left of Figure 25. After clicking on the 'MyDoctors' option, there is a doctor list.

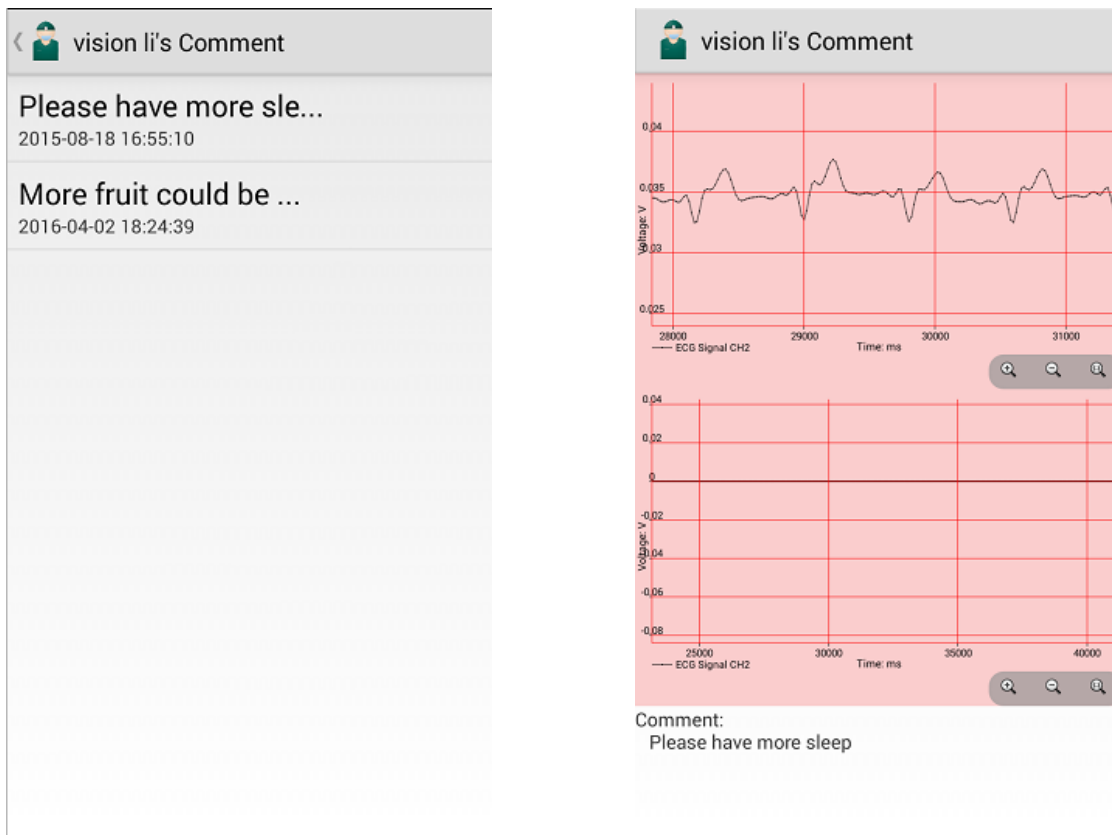


### **Figure 25 Doctor Management**

Each cell of the doctor list shows the doctor's first name, last name and hospital. There are two types of doctors, the connected doctors and the doctors who have sent connect request. As shown in the right of Figure 25, the first four doctors are connected doctors while the last doctor, Sean Yao, has sent the request for connections. The patient can choose accept the request or decline. After acceptance, this doctor would be in this list and have access to the patient's data. If declined, this doctor will be removed from the doctor list.

#### **3.7.2 Doctor Comment Management**

As described before, in our ECG system, doctors can leave comments to some of patient's ECG records. In the mobile APP, patients are able to view doctor's comments. When patients want to view a specific doctor's comments, they can click the doctor's name in the doctorlist. Then all the comments of the specific doctor will be listed. By clicking the specific comment, we can get the detail of the comment including the whole message as well as the ECG record which the comment is associated with.



**Figure 26 Doctor Comment**

The left of Figure 26 shows the comment list of doctor 'vision li' and the right figure shows the details of the first comment. The ECG data above is related to this doctor's comment.

## Chapter 4 Conclusion and Future Work

### 4.1 Conclusion

This project has focus on the development of Android APP and web service for wireless ECG system.

The Android APP and the web service have been verified in real time tests. The Android APP can scan and connect to the ECG sensor quickly. The single/double ECG data are displayed, stored and uploaded to server correctly. The management of doctors and doctor comments have been verified to be correct.

The web service provides API for mobile APP and web site. Both mobile APP and web site can correctly contact with web service. The APIs includes uploading and downloading ECG data, patient/doctor's log in/log out, patient/doctor's relation management, doctor comment reading and writing, patient note reading and writing.

### 4.2 Future Work

Currently, the system does not support the real time ECG data display on web site which could be useful for doctor's remote diagnosis. To realize this function, there are two potential methods, polling and pushing on web service. If we use polling, on the client side, the APP should send ECG data package every time received from the ECG monitor to the server. On the server side, the web site should send a request to the server in a fixed time period, 20 seconds for example, to see if there are new ECG data package on the server. If we use pushing, which means we have to use TCP/IP directly instead of http. The server and the mobile APP should hold the connection and the mobile APP

sends the ECG data package to the server every time a new package received. Also, the server should hold the connection with the web site and push the ECG data to the web client every time a new ECG data package is received. Both methods are doable.

However, the first method can be expensive for the server as too much Http requests can increase the server's load while the second methods' logic is more complex, especially when lots of users are using the real time ECG displaying at the same time.

Furthermore, if the system has more users, a single server may not be enough to handle too many requests. A properly designed distributed server architecture can be implemented in order to make the capacity of the server expandable.

Also, as our mobile APP is currently built on Android platform, it will be useful if an IOS version can also be built. Just like the Android platform, the IOS platform can support the Bluetooth Low Energy and all the other APIs used in this ECG mobile APP.

## Bibliography

- [1] Wikipedia – the free encyclopedia, “Bluetooth Low Energy (BLE)”. Available online:  
[https://en.wikipedia.org/wiki/Web\\_service](https://en.wikipedia.org/wiki/Web_service)
- [2] The Java EE 6 Official Tutorial from Oracle. Available online:  
<http://docs.oracle.com/javae/6/tutorial/doc/gijqy.html>
- [3] Jersey Restful Web Service official introduction. Available online: <https://jersey.java.net/>
- [4] JSON Official Introduction. Available online: <http://www.json.org/>
- [5] Wikipedia – the free encyclopedia, “Bluetooth Low Energy (BLE)”, Available online:  
[http://en.wikipedia.org/wiki/Bluetooth\\_low\\_energy](http://en.wikipedia.org/wiki/Bluetooth_low_energy)
- [6] Kevin Townsend, Carles Cufi, Akiba and Robert Davison: Getting Started with Bluetooth Low Energy, published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2014.
- [7] Android official development document, blue tooth low energy part. Available online:  
<http://developer.android.com/guide/topics/connectivity/bluetooth-le.html>
- [8] Android official development document, data base part. Available online:  
<http://developer.android.com/training/basics/data-storage/databases.html>

## Appendix A: Structure of Database Tables

### Table patient\_info

This table would store the information of patients

Field	Type	Description
userid	int(10) unsigned	Every user have a unique userid
username	varchar(32)	Patient's username
email	varchar(40)	Patient's email
firstname	varchar(20)	Patient's first name
lastname	varchar(20)	Patient's last name
phone	varchar(20)	Patient's phone number
emergencynum	varchar(20)	Patients' emergencynum
emergencymes	varchar(100)	Patient's emergency messages
ifsendsms	tinyint(1)	Whether Patient is allowed to send message in emergency situation
access_request	int(5)	The number of access request from doctors
doctor_num	int(10)	The number of doctors
dynamic_id	char(32)	This id is used for authentication on mobile APP
birthday	date	The birthday info of the user
address	varchar(100)	The home address
gender	int(1)	The gender information
medical_plan_number	varchar(11)	The medical plan number information
last_test_time	datetime	The user's last login time
session_id	varchar(32)	The user's session id

total_tests	int(11)	The number of user's ECG tests
total_comments	int(11)	Number of the comments given to this user
total_notes	int(11)	Number of notes this user made
created_time	datetime	The creation time of the user

### Table doctor\_info

This table would store all the information of a doctor

Field	Type	Description
doctor_id	int	The doctor's unique user id
password	varchar(20)	The doctor's password for login
docot_username	varchar(40)	The doctor's username
email	varchar(40)	The doctor's email address
firstname	varchar(20)	The doctor's first name
lastname	varchar(20)	The doctor's last name
phone	varchar(20)	The doctor's phone number
mobile	varchar(20)	The doctor's mobile phone number
lasttime	datetime	The last time this doctor login
hospital	varchar(50)	The hospital name of the doctor
patients_num	int	The doctor's connected patient numbers
gender	tinyint(1)	The gender of the doctor
address	varchar(100)	The work address of the doctor
birthday	date	The birthday of the doctor
session_id	varchar(32)	This is used for the identity authentication

**Table relation**

This table would store the connected relation between user and doctor

Field	Type	Description
doctor_id	int	The doctor's user id
user_id	int	The patient's user id
connected_date	date	When the relation was built

**Table access\_request**

This table would store the doctor's connection request to a patient

Field	Type	Description
doctor_id	int	The doctor's user id
user_id	int	The patient's user id
viewed	tinyint(1)	Whether patient has viewed this request
connected_date	date	When the request was sent

**Table doctor\_comment**

This table is used to store the information of a doctor comment.

Field	Type	Description
comment_id	Int	The unique id of each comment
doctor_id	Int	The id of the doctor who made the comment
created_time	datetime	The time when the comment was created
userid	int	Patient's userid
content	mediumtext	The content of the comment
viewed	tinyint	Whether this comment has been viewed

**Table comment\_relation**

This table is used to store the relation between the comment and note or ECG record.

Field	Type	Description
comment_id	int	The unique id of the comment
note_id	int	Related note's unique id
record_id	int	Related record's unique id

**Table mynote**

This table stores the information of the patient note

Field	Type	Description
userid	int	The patient's userid
note_time	datetime	The time when the note was created
content	mediumtext	Related record's unique id
noteid	Int	The note's unique id
event_start	datetime	Store the related time with the note

**Table note\_relation**

This table stores the relation between patient note and ECG test / record

Field	Type	Description
noteid	int	The note's id
testid	int	The related ECG test's id
recordid	int	The related ECG record's id

**Table ecgtestinfo**

This table stores the information of the ECG long term monitoring tests

<b>Field</b>	<b>Type</b>	<b>Description</b>
testid	int	The test's unique id
userid	int	The id of the user who made this test
testtime	datetime	The start time of this test
data_num	int	The number of the ECG data with this tes
viewed	tinyint(1)	Whether this test has been viewed
total_notes	int	Total patient note made on this test
total_comments	int	Total doctor comment made on this test

### **Table ecgdatainfo**

This table stores the information of each ECG data

<b>Field</b>	<b>Type</b>	<b>Description</b>
dataid	int	The data's unique id
starttime	int	The start time of the ECG data
length	datetime	The length of the ECG data in seconds
ifdone	int	Whether this data has finished uploading
userid	tinyint(1)	The id of the user who made this ECG data
testid	int	The id of the test this ECG data belongs to
testtime	int	Test time of the test this ECG data belongs to
HR	int	The heard rate of this ECG data
QRS_duratoin	int	The QRS duration of this ECG data
QT_interval	int	The QT interval of this ECG data
PR_interval	int	The PR interval of this ECG data
QTC	int	The QTC value of this ECG data



## Appendix B: Server Configuration

Steps to configure ECG Web Server after Linux OS reinstalled:

### Step One—Update and install SSH

Using apt-get as an installer for all the server programs to download all the latest patched software:

```
sudo apt-get update
sudo apt-get install open-ssh
```

After ssh has been installed, we can access Linux server through SSH using SecureCRT or any other similar tools.

### Step Two—Install Nginx

Our Linux Server uses Nginx as a front-end proxy server, all of the requests will be redirected to port:8080 (tomcat) when receiving a URI `http://ecg.ece.uvic.ca/`.

```
sudo apt-get install nginx
```

Configuration file named default under the directory `/etc/nginx/sites-available/` needs to be modified. (See file default attached together for more detail.)

To get Nginx running, type

```
sudo service nginx start
```

### Step Three—Install tomcat7

```
sudo apt-get install tomcat7
```

To get tomcat7 running, type

```
sudo service tomcat7 start
```

```
sudo adduser ecg tomcat7
```

To conform tomcat7 running properly, type `//localhost:8080`, we will see tomcat7 welcome page.

### Step Four—Install MySQL

MySQL is a powerful database management system used for organizing and retrieving data.

```
sudo apt-get install mysql-server
```

During the installation, MySQL will ask you to set a root password (ecg1234). If you miss the chance to set the password while the program is installing, we need to set the password later from within the MySQL shell.

After the installation, using `mysql -uroot -pecg1234` to conform that we have installed MySQL successfully.

### Step Five—Import ecgdatabase

1) access to MySQL

```
mysql -uroot -pecg1234
```

2) create ecgdatabase

```
create database ecgdatabase;
```

Ctrl+C to exit

3 ) Copy the backup database file ecgdatabase.sql to a directory using xftp or any other sftp tool, go to the directory under the Linux terminal, import the data.

```
mysql -u root -p -h localhost ecgdatabase<ecgdatanase.sql
```

(when using this command, we don't need to access to MySQL)

### Step Six—Import our ECG WebService servlet

Copy the whole project file EcgWebService to /usr/share/tomcat7/webapps

Modify Tomcat7 configuration files server.xml and context.xml under the directory /etc/tomcat7/ (see attached file server.xml and context.xml for more detail).

Restart tomcat7: sudo service tomcat7 restart

Type //localhost/rest in the browser to see if our ECG server runs correctly.

Notes:

Please backup everything list below before reinstalling system:

1) Executable file: backup whole root directory

```
/usr/share/tomcat7/webapps/EcgWebService
```

2) Eclipse workspace

```
/home/ecg/workspace
```

3) Tomcat7 configuration file

```
/etc/tomcat7/context.xml
```

```
/etc/tomcat7/server.xml
```

```
sudo passwd root //set Linux root password after system reinstalled.
```

```
chmod 777 directoryname //change the privilege of the directory
```

```
chmod 777 * //change all the files' privilege under the current directory
```

It's very helpful to check the tomcat log file catalina.out under the directory if any error occurs.