Project Report

> Group Number: 3
>
> Name: Probuddho Chakraborty / Zipeng Lin、
>
> GitHub: https://github.com/hrit1995/HeartBeatSensorUsingPPG

1. Introduction

Understanding of our heartrate pattern and some abnormal heartrate behavioral is one of the key factor to monitor one's body and mental health. Therefore, almost all of the trending smart watch, or other fitness device has a heartbeat sensor built in. Among which, the most popular sensor is the PPG sensor which use light to observe the change of our vessel to calculate heartrate changes.

So in this project, we implemented a cloud-based real time heartrate monitoring and anxiety detection system with an Android App using the SparkFun PPG sensor and SparkFun ESP8266 chip.

2. Challenges

For this project, there are three major challenges:

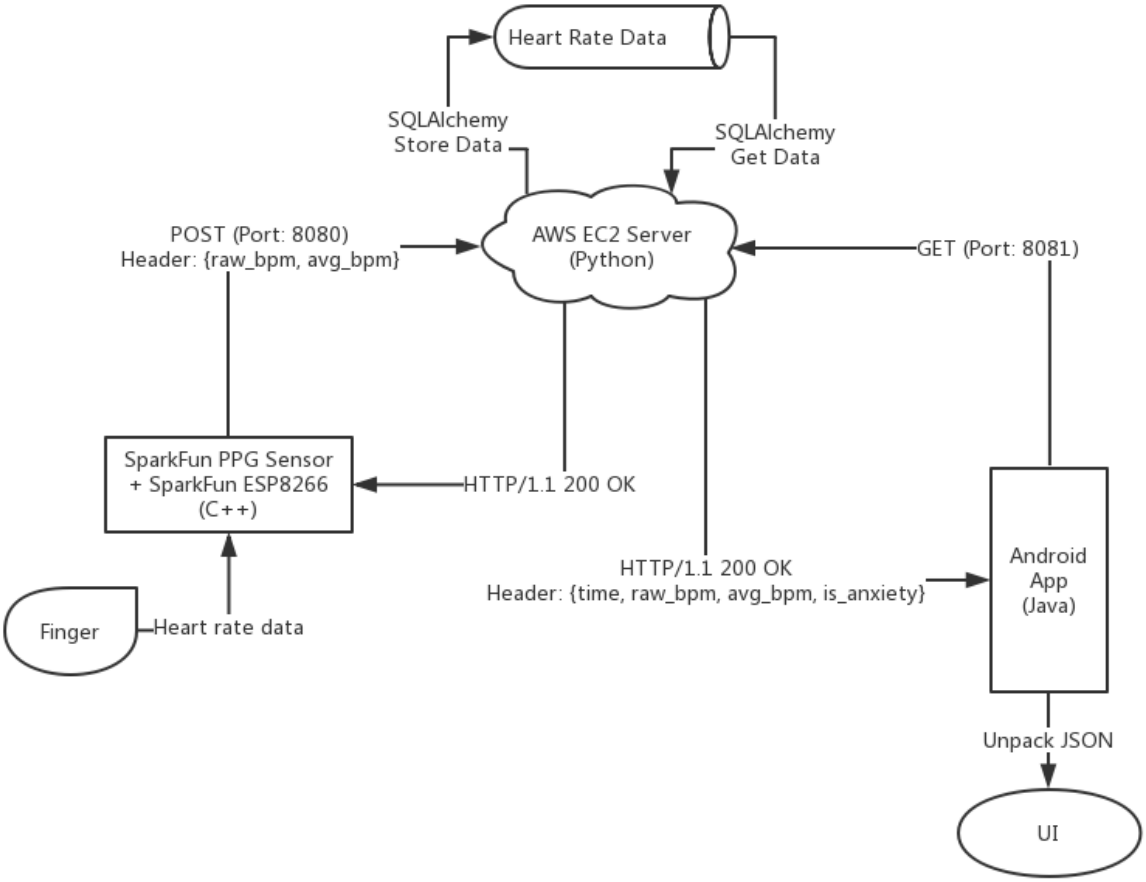
- How to turn raw data from the sensor into real heartbeat data and send to cloud.
- How to manage the data on the cloud and to design the APIs for data transfer.
- How to design and code the Android App and the way to display the data

3. Solution

The solution will consists of three parts:

- Hardware Design (Sparkfun C++)
- Connectivity Design (Python, AWS EC2)
- Android App Design (Java)

And the whole design pattern is:



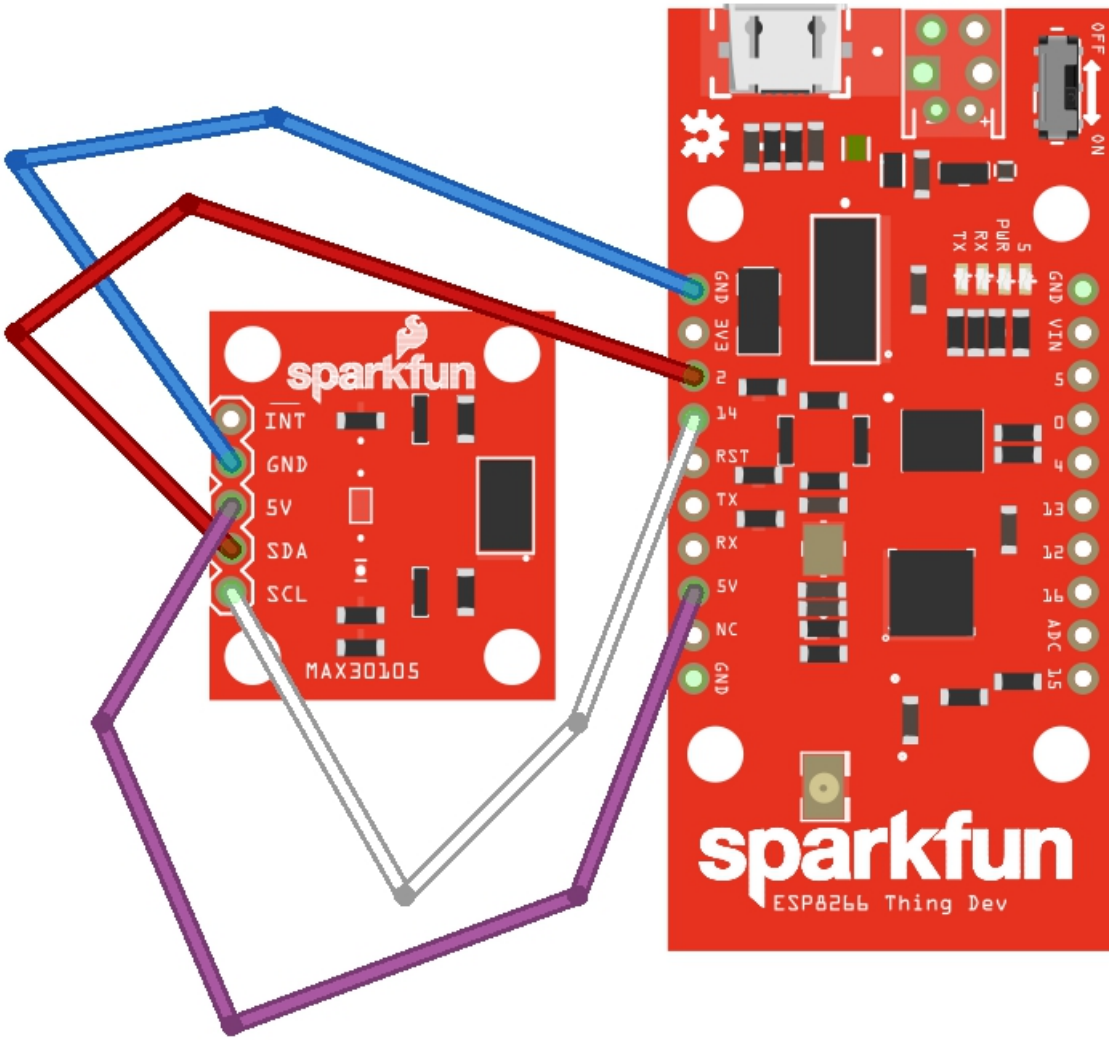
3.1 Hardware Design

In the hardware design, we use SparkFun PPG Sensor for getting the heartrate data and SparkFun ESP8266 WiFi chip for connectivity.

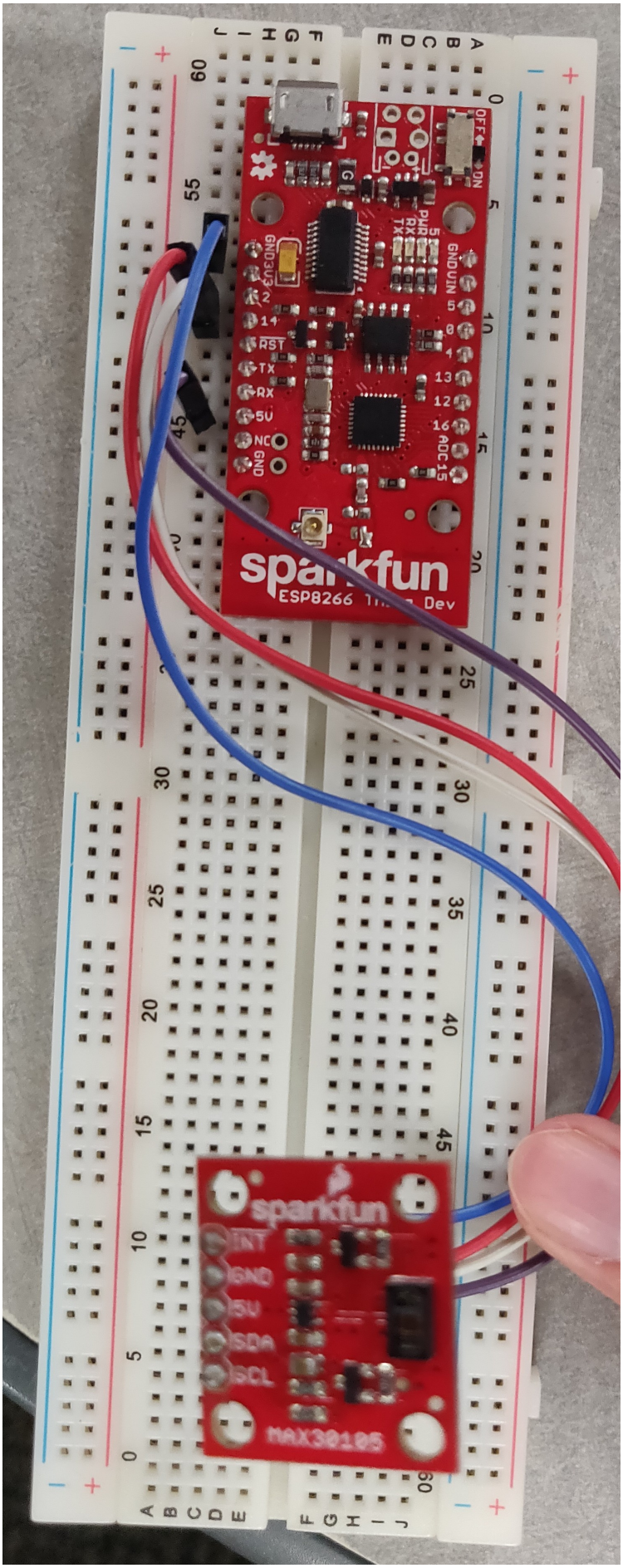
Pin Assignment

Pin	Assignment	Device
2	SDA In	SparkFun ESP8266
14	SCL Out	SparkFun ESP8266
SDA	SDA Out	SparkFun PPG Sensor
SCL	SCL In	SparkFun PPG Sensor
GND	GND	Both PPG & ESP8266
5V	5V	Both PPG & ESP8266

Wiring


sparkfun
INT
GND
5V
SDA
SCL
MAX30105
OFF
ON
PWR
RX
TX
GND
3V3
2
14
RST
TX
RX
5V
NC
GND
GND
VIN
5
0
4
13
12
16
ADC
15
sparkfun
ESP8266 Thing Dev

fritzing



The SparkFun ESP8266 Thing Dev board will read analog data from the PPG sensor and use the algorithm below to generate the heartbeats data.

Performed Algorithm

The algorithm will convert the input raw data into heartbeats data.

The BPM will be:

$$BPM = \frac{60 \times 1000.0}{\Delta IR}$$

And the average BPM will be evaluate every `RATE_SIZE` (Currently we set it to 4) input BPM data:

$$Average\ BPM = \frac{60 \times 1000.0}{RATE\_SIZE \times \Delta IR}$$

```
if (checkForBeat(irValue) == true)
{
    //We sensed a beat!
    long delta = millis() - lastBeat;
    lastBeat = millis();

    beatsPerMinute = 60 / (delta / 1000.0);

    if (beatsPerMinute < 255 && beatsPerMinute > 20)
    {
        rates[rateSpot++] = (byte)beatsPerMinute; //Store this reading in the
array
        rateSpot %= RATE_SIZE; //Wrap variable

        //Take average of readings
        beatAvg = 0;
        for (byte x = 0 ; x < RATE_SIZE ; x++)
            beatAvg += rates[x];
        beatAvg /= RATE_SIZE;
    }
}
```

Connectivity Design

For the connectivity design, I use HTTPS POST method to send data to the server on the cloud. The data will be sent in form `String(bpm + "," + avgbpm)`

```
HTTPClient http;    //Declare object of class HTTPClient

http.begin("https://[my url]:8080");      //Specify request destination
http.addHeader("Content-Type", "text/plain");  //Specify content-type header
http.setReuse(true);

String bpm = String(beatsPerMinute);
String avgbpm = String(beatAvg);
String sender = String(bpm + "," + avgbpm);

int httpCode = http.POST(sender);   //Send the request
```

3.2 Server Design

On the server part, we will have the connectivity part as well as the MySQL database to store the data. The server is written by Python with a RESTful design on the APIs, so as long as the sender and receiver POST or GET to the right address and socket with the right data structure their operation will be a success. The design of the socket is listed in the table.

> Database Design

The database on the cloud is based on MySQL, and we use SQLAlchemy to manage data in MySQL.

SQLAlchemy is a Python package that works as an ORM(Object-relational Mapping) from Python to MySQL so that the structure design as well as input or read data can use only Python without using SQL codes.

Every connection to a database will have an engine as well as a session object to do all the operations.

- Engine & Session

```
engine = create_engine("mysql+pymysql://root:
[password]@localhost:3306/iot_proj", echo=True)
Session = sessionmaker(bind=engine)
session = Session()
```

- Design of Database

```
class heartRate(base):
    __tablename__ = "heartrate"
    time = Column(Integer, primary_key=True, autoincrement=True)
    rdata = Column(Integer)
    avgbpm = Column(Integer)
    is_anxiety = Column(Boolean)
```

Index	Data Type	Is_Primary_Key	Auto Increment	Function
time	Integer	True	True	Set time stamp for data
rdata	Integer	False	False	Raw BPM data
avgbpm	Integer	False	False	Average BPM data
is_anxiety	Boolean	False	False	A flag for anxiety detection

- Insert & Select Operation

```
# Insertion
session.add(heartRate(rdata=res[0], avgbpm=res[1], is_anxiety=False))
# Select
for data in session.query(heartRate).order_by(heartRate.time.desc()).limit(20):
```

> Socket Design

socket	function
8080	Receive POST request for receiving data
8081	Receive GET request for returning data

- Port 8080

```
HOST, PORT = '', 8080

listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
listen_socket.bind((HOST, PORT))
listen_socket.listen(1)
print('Serving HTTP on port {} ...'.format(PORT))
while True:
    client_connection, client_address = listen_socket.accept()
    request = client_connection.recv(1024)
    res = request.decode().split("\r\n")[-1].split(',')
    print(request)
    print(res)
    if len(res) == 2:
        session.add(heartRate(rdata=res[0], avgbpm=res[1], is_anxiety=False))
        session.commit()
    http_response = """\
                        HTTP/1.1 200 OK
                        \nEOF
                    """
    client_connection.sendall(http_response.encode())
    client_connection.close()
```

In here the server will keep listening port 8080, once it receive the data, it will put them into the database.

- Port 8081

```
HOST, PORT = '', 8081

listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
listen_socket.bind((HOST, PORT))
listen_socket.listen(1)
print('Serving HTTP on port {} ...'.format(PORT))
tmp = 0
while True:
    client_connection, client_address = listen_socket.accept()
    request = client_connection.recv(1024)
    print(request.decode())
    http_response = "{"
    tmp += 1
    Session = sessionmaker(bind=engine)
    session = Session()
    for data in
session.query(heartRate).order_by(heartRate.time.desc()).limit(20):
        if tmp % 20 == 0:
            http_response += "}"
```

```
            break
        else:
            tmp += 1
            if tmp != 1:
                http_response += ","
        http_response += "\"{}\":".format(data.time) + "[" + "\"{}\", \"{}\", \"
{}\"".format(data.rdata, data.avgbpm, data.is_anxiety) + "]"
    tmp = tmp % 100
    client_connection.send(http_response.encode())
    client_connection.close()
```

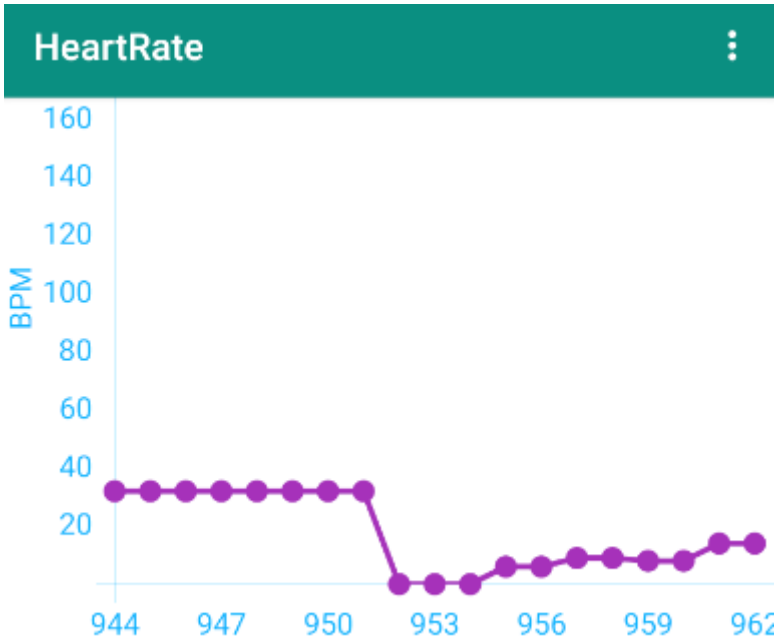
The port 8081 will listen to any GET request and return the latest 20 data on the database and wrap them into JSON form and send back to the host that send the request.

3.3 Android App Design

The design of the Android App leverages the multithread design to make sure that the result will be updated in real time.

> Design of the Data Display Line chart

In this part, I use `hellocharts` to draw the line chart. The final result will be like this.



Since the `chartView` is a thread, so to update the chart for multiple times, we must use a handler to update the chart. At the same time, I use another thread to do the connectivity part.

> Connectivity

```
private void startNetThread() {
    Thread thread = new Thread() {
        @Override
        public void run() {
            try {
                Socket socket = new Socket("[server ip]", 8081);
                OutputStream outputStream = socket.getOutputStream();
                outputStream.write(("test").getBytes());
                outputStream.flush();
```

```
                socket.shutdownOutput();
                InputStream is = socket.getInputStream();
                byte[] bytes = new byte[1024];
                int n = is.read(bytes);
                str = new String(bytes, 0, n);
                //updateTextView(str);
                is.close();
                socket.close();
            } catch (Exception e) {
            }
        }
    };
    thread.start();
    try {
        thread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

So the handler will call `startNetThread` in a cycle to keep updating the data on the chart.

4. Result

The result will be split into four parts:

- On Sparkfun
- Receiver side on the server
- Sender side on the server
- Android App Demonstration

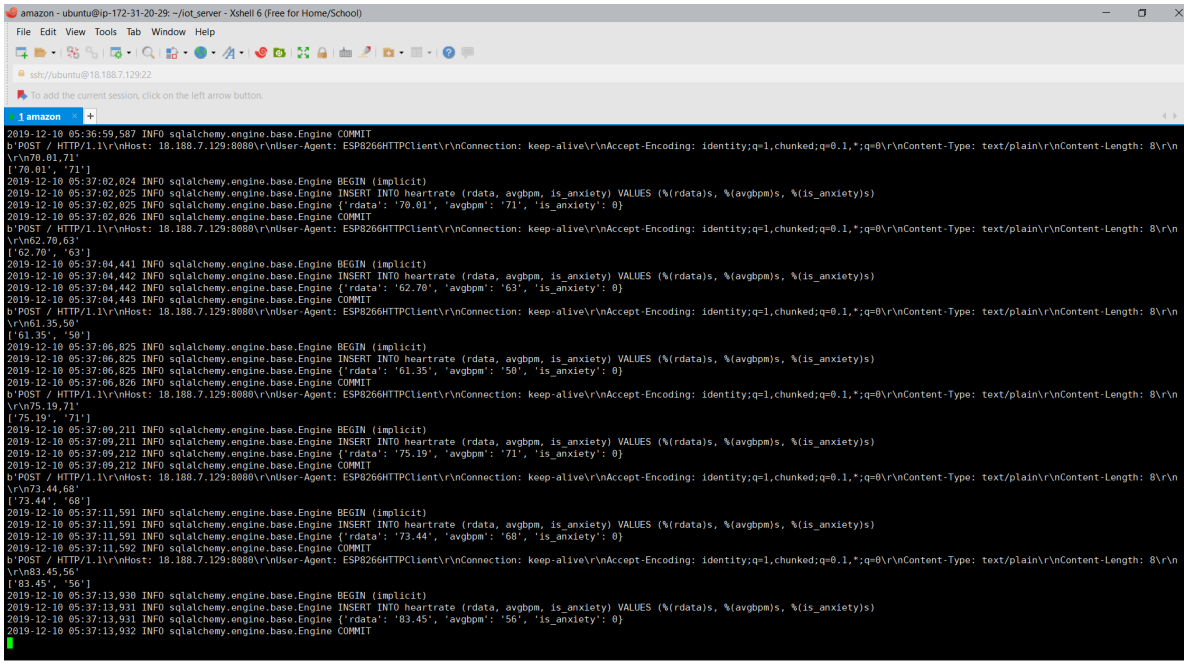
On Sparkfun

Send

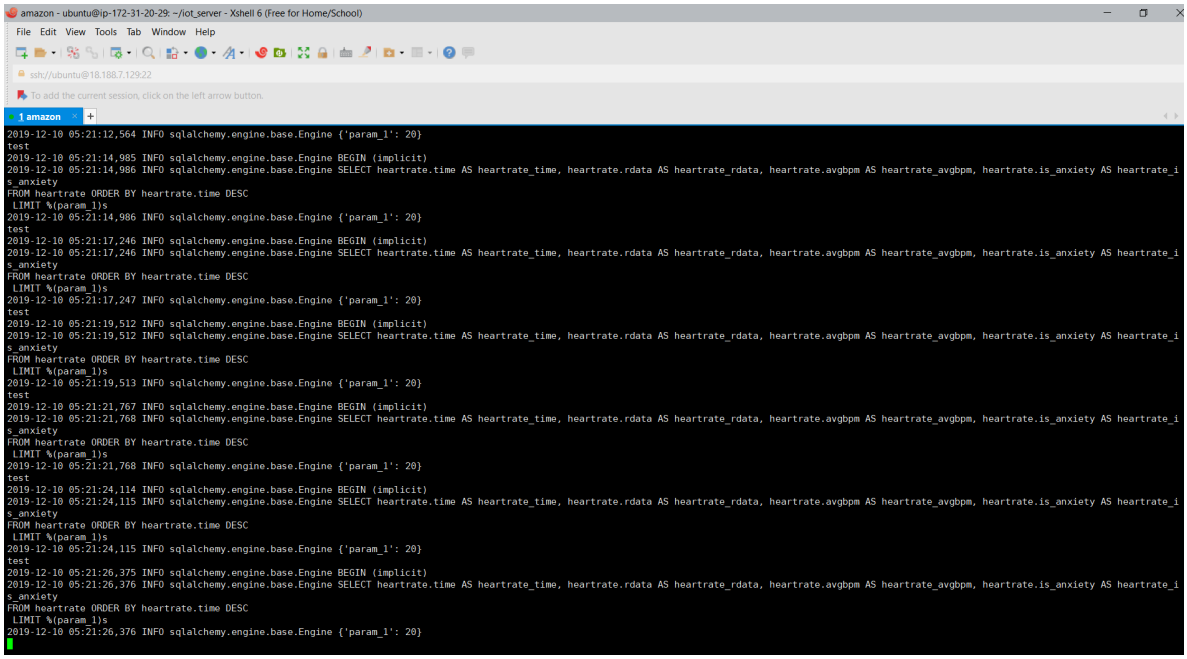
```
IR=116880, BPM=47.02, Avg BPM=45
IR=116847, BPM=47.02, Avg BPM=45
IR=116934, BPM=47.02, Avg BPM=45
IR=116901, BPM=47.02, Avg BPM=45
IR=116891, BPM=47.02, Avg BPM=45
IR=116923, BPM=47.02, Avg BPM=45
IR=116844, BPM=47.02, Avg BPM=45
IR=116893, BPM=47.02, Avg BPM=45
IR=116892, BPM=47.02, Avg BPM=45
IR=116919, BPM=47.02, Avg BPM=45
IR=116960, BPM=47.02, Avg BPM=45
IR=116893, BPM=47.02, Avg BPM=45
IR=116959, BPM=47.02, Avg BPM=45
IR=116927, BPM=47.02, Avg BPM=45
IR=116901, BPM=47.02, Avg BPM=45
IR=116972, BPM=47.02, Avg BPM=45
IR=116938, BPM=47.02, Avg BPM=45
IR=116983, BPM=47.02, Avg BPM=45
IR=116979, BPM=47.02, Avg BPM=45
IR=116956, BPM=47.02, Avg BPM=45
IR=116988, BPM=47.02, Avg BPM=45
IR=116964, BPM=47.02, Avg BPM=45
IR=116981, BPM=47.02, Avg BPM=45
IR=117027, BPM=47.02, Avg BPM=45
IR=116991, BPM=47.02, Avg BPM=45
IR=117063, BPM=47.02, Avg BPM=45
IR=117040, BPM=47.02, Avg BPM=45
IR=117065, BPM=47.02, Avg BPM=45
IR=117067, BPM=47.02, Avg BPM=45
IR=117025, BPM=47.02, Avg BPM=45
IR=117088, BPM=47.02, Avg BPM=45
IR=117029, BPM=47.02, Avg BPM=45
```

Autoscroll Show timestamp No line ending 115200 baud Clear output

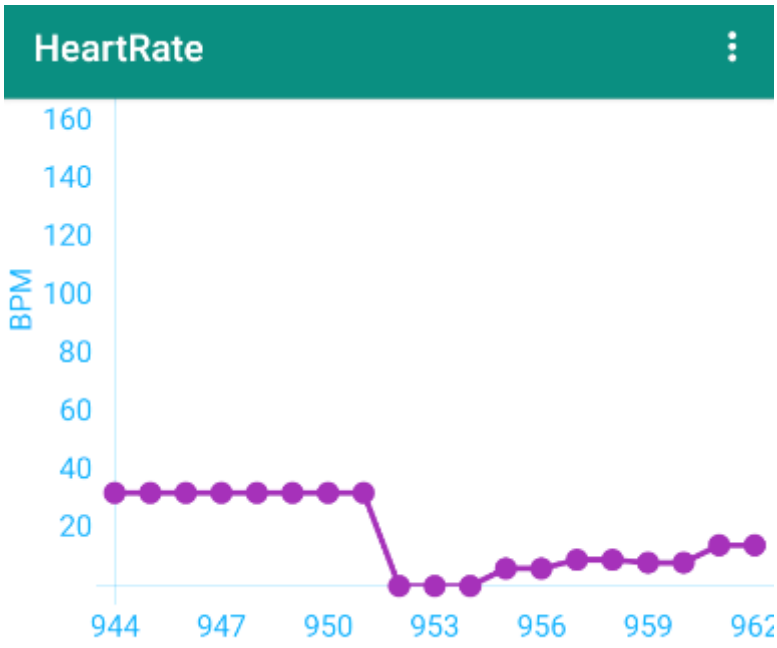
Receiver side on the server



Sender side on the server



Android App Demonstration



The video demo is in the project GitHub directory please open Demo video of app.mp4

Future Work

- Display the numeric heart rate value below the line chart.
- Irregular heart rate pattern pop out notification on Android App.
    - If the user is working out, he/she will get an option to close the irregularity notification from the app.
    - If it is indeed a dire situation, the user can choose to acknowledge the notification and take necessary steps.

Reference

[1]. MAX30105 Particle and Pulse Ox Sensor Hookup Guide. https://learn.sparkfun.com/tutorials/max30105-particle-and-pulse-ox-sensor-hookup-guide/all#hardware-hookup

[2]. SparkFun ESP8266 Thing - Dev Board. https://www.sparkfun.com/products/13711

[3]. Android Line Chart – How to Draw Line Chart in Android. https://www.codingdemos.com/draw-android-line-chart/

[4]. Handler Documentation. https://developer.android.com/reference/android/os/Handler

[5]. SQLAlchemy: The Python SQL Toolkit and Object Relational Mapper. https://www.sqlalchemy.org/