ENSTA BRETAGNE

Rapport Final

Stage 2A

Ang LI

10/10/2016

Introduction

My internship project achieved an embedded system about Raspberry Pi and Sense Hat for measuring the weather. The project includes the system configurations of Raspbian which is based on Debian, the establishment of the server based on Python, web development by HTML and JavaScript and REST telecommunication. The system can be accessed remotely via a browser and transmit measured data to users.

This weather measurement system can be used for static and dynamic measurement in the extreme environments, such as Hypoxic environment, toxic environment and so on. This system could appear as a part of detection robot because of the integrated sensors in Sense Hat.

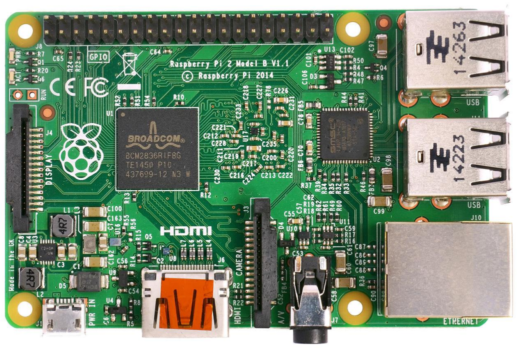
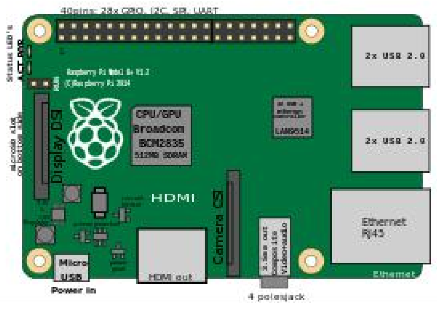
Contents

1.Raspberry Pi and Sense Hat

1. 1 Presentation

1.1.1 Raspberry Pi

The Raspberry Pi is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science. Several generations of Raspberry Pis have been released. I use Raspberry 2B for my project.

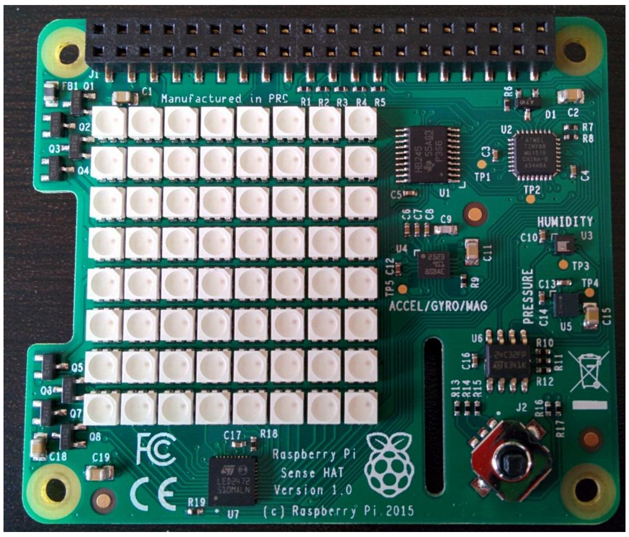
As a card-sized single-board computer, Raspberry Pi works well for a convenient development. There are enough interfaces for different functions. The following table shows the parameters of Raspberry Pi I used.

	Raspberry Pi 2 model B
Architecture	ARMv7 (32-bit)
SoC	Broadcom BCM2836
CPU	900 MHz 32-bit quad-core ARM Cortex-A7
GPU	Broadcom VideoCore IV @ 250 MHz,OpenGL ES 2.0 MPEG-2 and VC-1, 1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder
Memory (SDRAM)	1 GB
USB 2.0 ports	4
Video input	15-pin MIPI camera interface (CSI) connector, used with the Raspberry Pi camera or Raspberry Pi NoIR camera
Video outputs	HDMI
On-board network	10/100 Mbit/s Ethernet (8P8C) USB adapter on the USB hub
On-board storage	MicroSDHC slot
Low-level peripherals	17× GPIO plus the same specific functions, and HAT ID bus
Power ratings	800 mA
Size	85.60 mm × 56.5 mm

Weight	45 g

In my project, i just used a 16G SD card for larger storage space, 2 USB for keyboard and mouse, the Ethernet for network, HDMI interface for a visible development and the power supply interface. For the rest interfaces, such as video input which needs an external camera, i didn’t consider them.

1.1.2 Sense Hat



The Sense HAT is an add-on board for Raspberry Pi, made especially for the Astro Pi mission. It launched to the International Space Station in December 2015 and is now available to buy. The Sense HAT has an 8×8 RGB LED matrix, a five-button joystick and includes the following sensors: Gyroscope, Accelerometer, Magnetometer, Temperature, Barometric pressure, Humidity. There is a Python library providing easy access to everything on the board.

Thanks for sophisticated design, Sense Hat can works with other external devices, such as an external camera at the same time.

1. 2 Configurations of Raspberry Pi

1.2.1 System

In the official website of Raspberry Pi, there are some operating system used for Raspberry Pi. But NOOBS and Raspbian are two recommended system because of stability. Raspbian is the Foundation’s official supported operating system based on Debian. NOOBS is a easy installer for Raspbian and more. It has a good novice guidance for the beginners. NOOBS is a little more simple than Raspbian, it can not support some modules, such as the Sense Hat. So in my project, i chose Raspbian.

In order to install the system in Raspberry Pi, first, downloading the system with another computer, and burn the system to the SD card. Secondly, inserting the SD card into the Raspberry pi, then following the steps and the system can be installed. It's not difficult.

It is worth mentioning that we can use two different ways to visually access the Raspberry Pi not just for installing the system but also for later development. One is Secure Shell(SSH) and the other is direct connection like a desktop PC with external keyboard, mouse and monitor.

Comparing the two methods, SSH requires fewer accessories, no keyboard and no mouse but a twisted pair for directly connecting notebook and a power cable. It It is very convenient for programming and testing. However, there is a problem that without wifi card, it couldn't connect the internet and update system components from sources. So i just used SSH for later web development and test.

1.2.2 Network

For my web development, i need to give my server a static IP address. So The network configuration of Raspberry Pi is necessary.

The best way is as follows:

① Open the file from the console:

```
sudo nano /etc/dhcpcd.conf
```

② Add content in the file:

```
interface eth0
static ip_address=172.20.88.88/24
static routers=172.20.88.1
static domain_name_servers=192.168.1.13 192.168.1.17
```

③ And then reboot Raspberry Pi.

The IP address is permanently changed. This way is more suitable for my project. There is another way for a temporary IP change, that is to say, IP will become the default value after reboot.

The way is as follows:

① Open the file from the console:

```
sudo nano /etc/network/interfaces
```

② Change content in the file:

```
iface eth0 inet static
address 172.20.88.88
netmask 255.255.0.0
gateway 172.20.88.1
```

③ And then reboot Raspberry Pi.

I stress this approach because I spent a lot of time on this issue of static network configuration, most of the tutorials in internet use the second one. I must say that the first is the best one. I don't elaborate dynamic network configuration and wifi configuration because my project does not involve them.

Because i used the school's network, i had to configure the proxy for Raspberry Pi.

The way is as follows:

① Open the file from the console:

```
sudo nano ~/.bashrc
```

② Add content in end of the file:

```
http_proxy=http://pi:raspberry@192.168.1.17:8080
export http_proxy
```

③ And then reboot Raspberry Pi.

"pi" is user name and "raspberry" is password of the Raspberry Pi. "192.168.1.17" is proxy of school. After finishing all the network configurations, Raspberry Pi can connect to the internet. Downloading software and visiting the website are possible.

1.2.3 Necessary modules

After connecting to the internet, Raspberry Pi can update and install many different types of modules. In my project, i need to install Flask, node.js, sense-hat, chart.js and so on.

Firstly, the software default source is usually good but it could be changed for a higher download speed:

```
sudo nano /etc/apt/sources.list
```

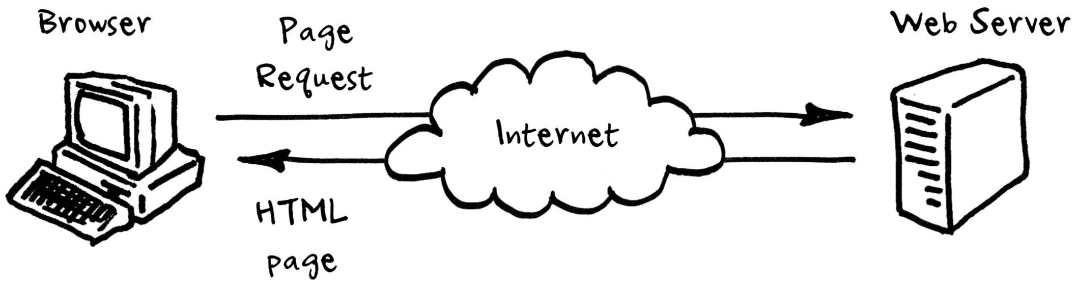
In the file, the source could be changed to another source from this official mirror list: http://www.raspbian.org/RaspbianMirrors.

Secondly, "apt-get" is a very useful command for installing new software. The commands are as follows:

```
sudo apt-get install nodejs
sudo apt-get install npm
sudo apt-get install sense-hat
npm install chart.js --save

sudo apt-get install python-pip
sudo pip install flask
```

Finally, "sudo apt-get update" and "sudo apt-get upgrade" could be used for updating system software and updating installed packages.

2. REST



Representational state transfer (REST) or RESTful web services are one way of providing interoperability between computer systems on the internet. REST-compliant web services allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations. In a REST web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or some other defined format. Using HTTP, as is most common, the kind of operations available include those predefined by the HTTP verbs GET, POST, PUT, DELETE and so on.

REST architectural style The most important architectural constraints are six:

Client-Server: Communication can only be initiated by the client unilaterally, manifested in the form of request-response.

Stateless: The session state of the communication should be maintained by the client.

Cache: Response content can be cached somewhere in the communication chain to improve network efficiency.

Uniform Interface: Components of the communication chain communicate with each other through a unified interface to improve the visibility of the interaction.

Layered System: The architecture is decomposed into layers of several levels by limiting the behavior of the components.

Code-On-Demand (optional): Support for extending the functionality of the client by downloading and executing some code (such as Java Applet, Flash, or JavaScript).

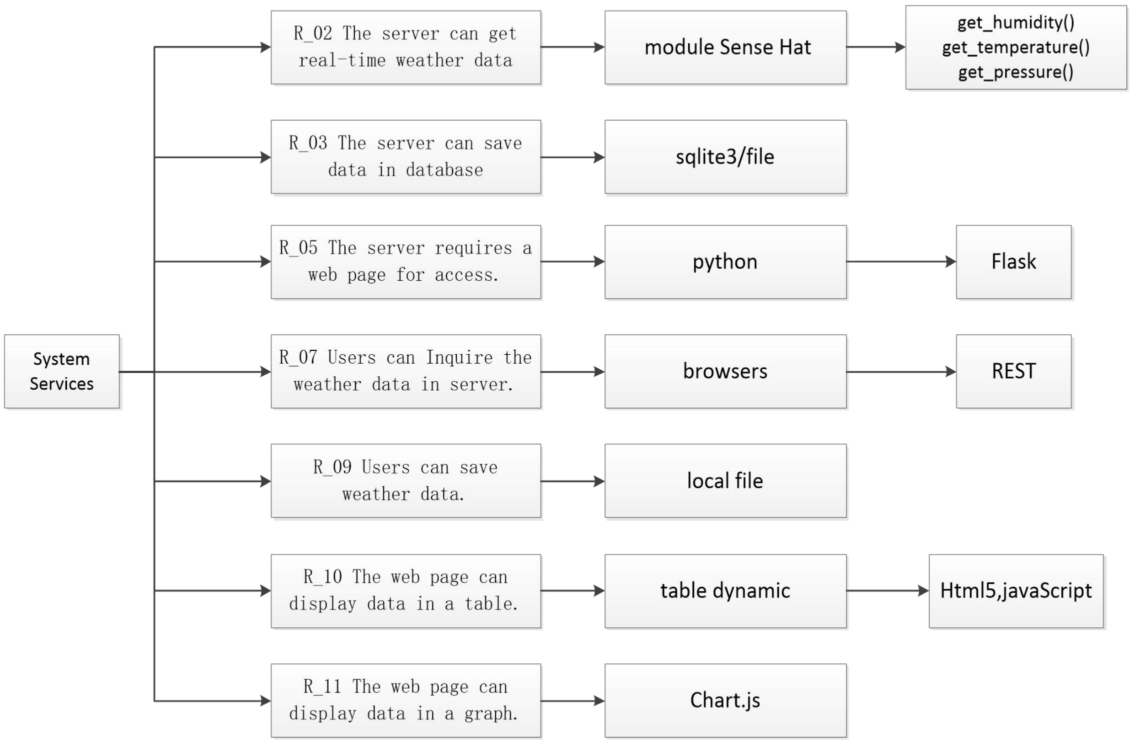
3.System architecture

3.1 Requirements analysis

ID	Type	Description of Requirement
R_01	Constraint	The server works in Raspberry Pi.
R_02	Service	The server can get real-time weather data from the captures of Sense Hat.
R_03	Service	The server can save data in database.
R_04	Constraint	The server should be built by Python.
R_05	Service	The server requires a web page for access.
R_06	Constraint	Raspberry Pi and Sense Hat can work for 24 hours
R_07	Service	Users can Inquire the weather data in server by browser.
R_08	Constraint	Users can not add, delete or modify any data in server.
R_09	Service	Users can save weather data as a local file.
R_10	Service	The web page can display data in a table dynamic.
R_11	Service	The web page can display data in a graph.
R_12	Constraint	The web page can be compatible with multiple browsers.

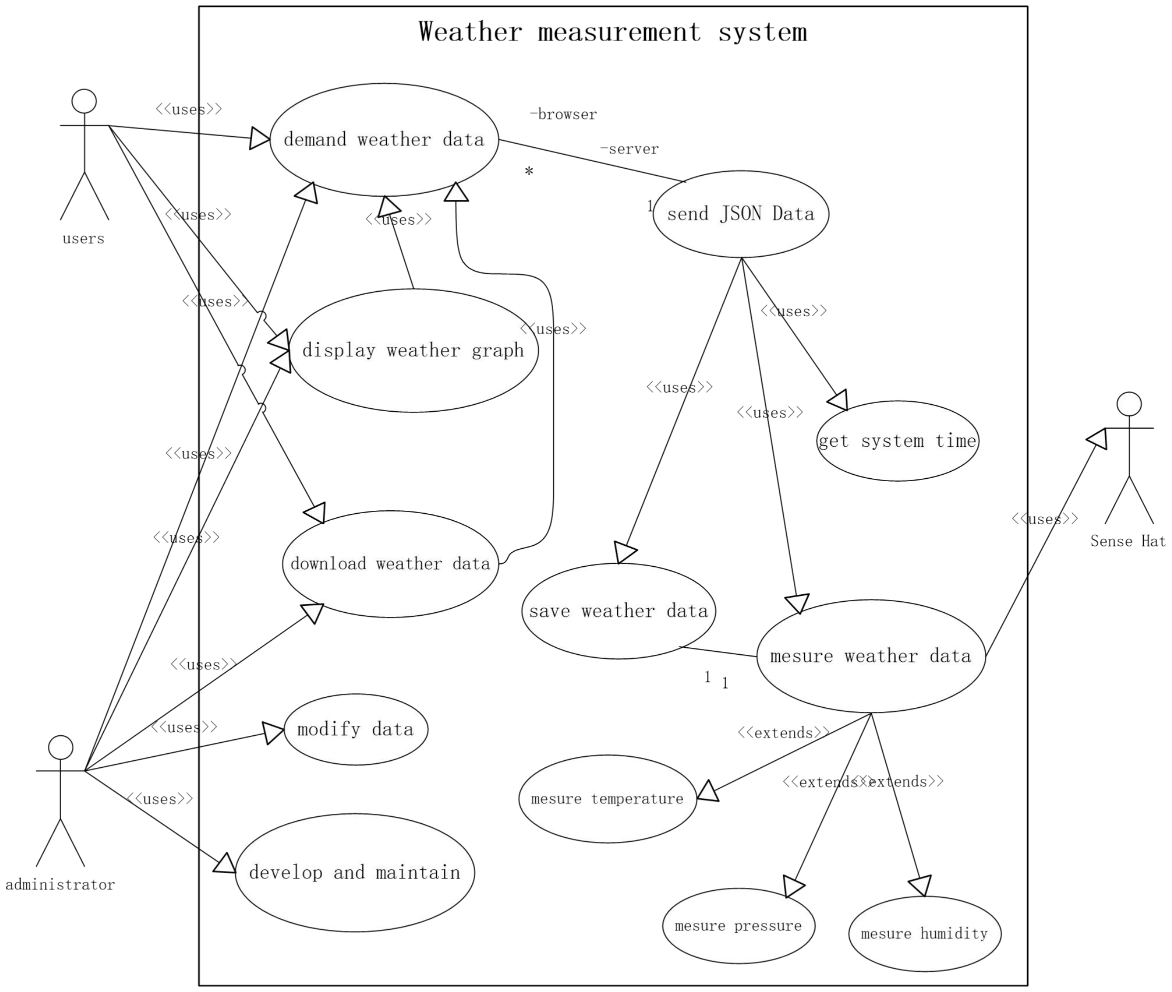
As shown in the table above, the requirements of my project is clear. In general, The server works in Raspberry Pi and it get weather data from Sense Hat. Users can demand these weather data by browser.

3.2 FAST diagram


System Services
R_02 The server can get real-time weather data
module Sense Hat
get_humidity()
get_temperature()
get_pressure()
R_03 The server can save data in database
sqlite3/file
R_05 The server requires a web page for access.
python
Flask
R_07 Users can Inquire the weather data in server.
browsers
REST
R_09 Users can save weather data.
local file
R_10 The web page can display data in a table.
table dynamic
Html5,javaScript
R_11 The web page can display data in a graph.
Chart.js

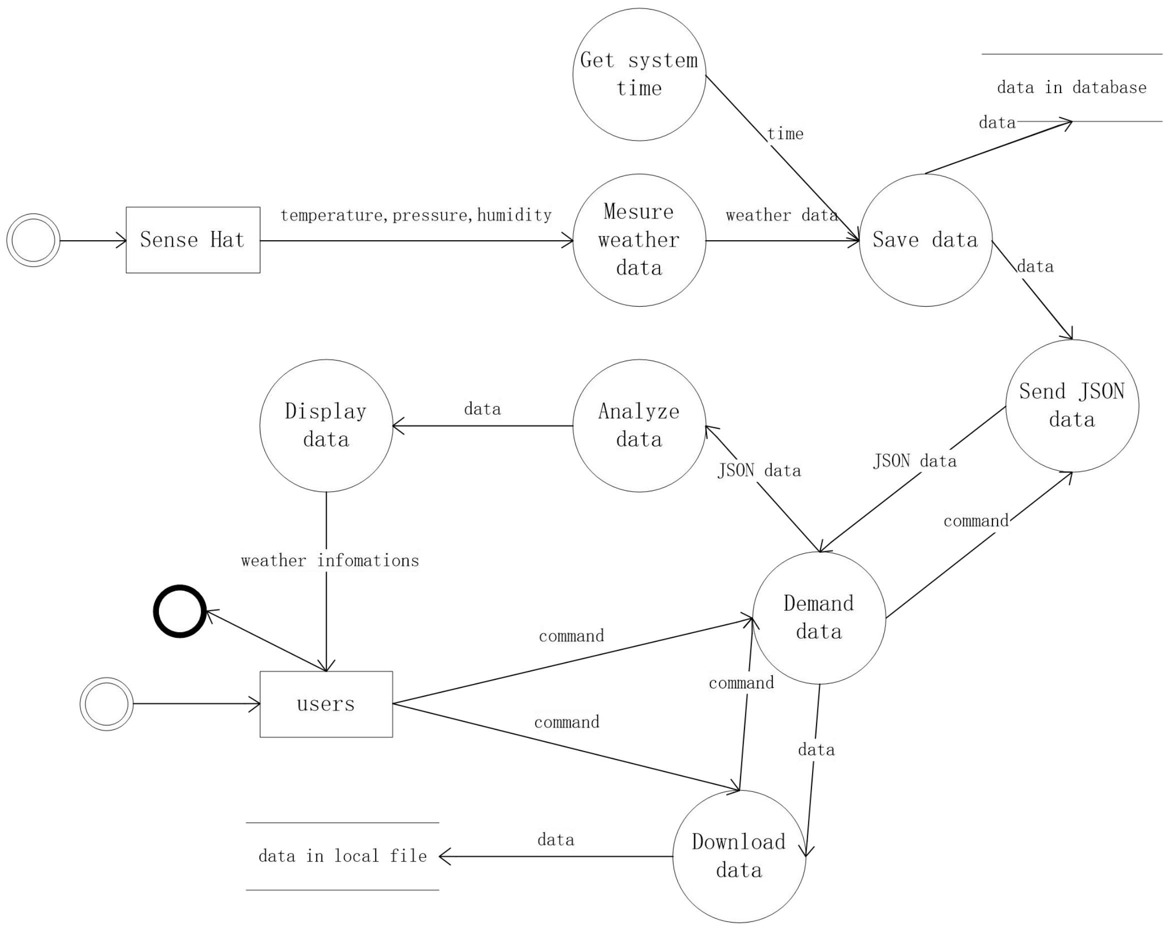
As shown in the FAST diagram above, my project requires some different knowledge about Framework Flask, REST, HTML5, JavaScript, JSON data and open source library Chart.js. Each of them was a new one for me.

3.3 Use case diagram



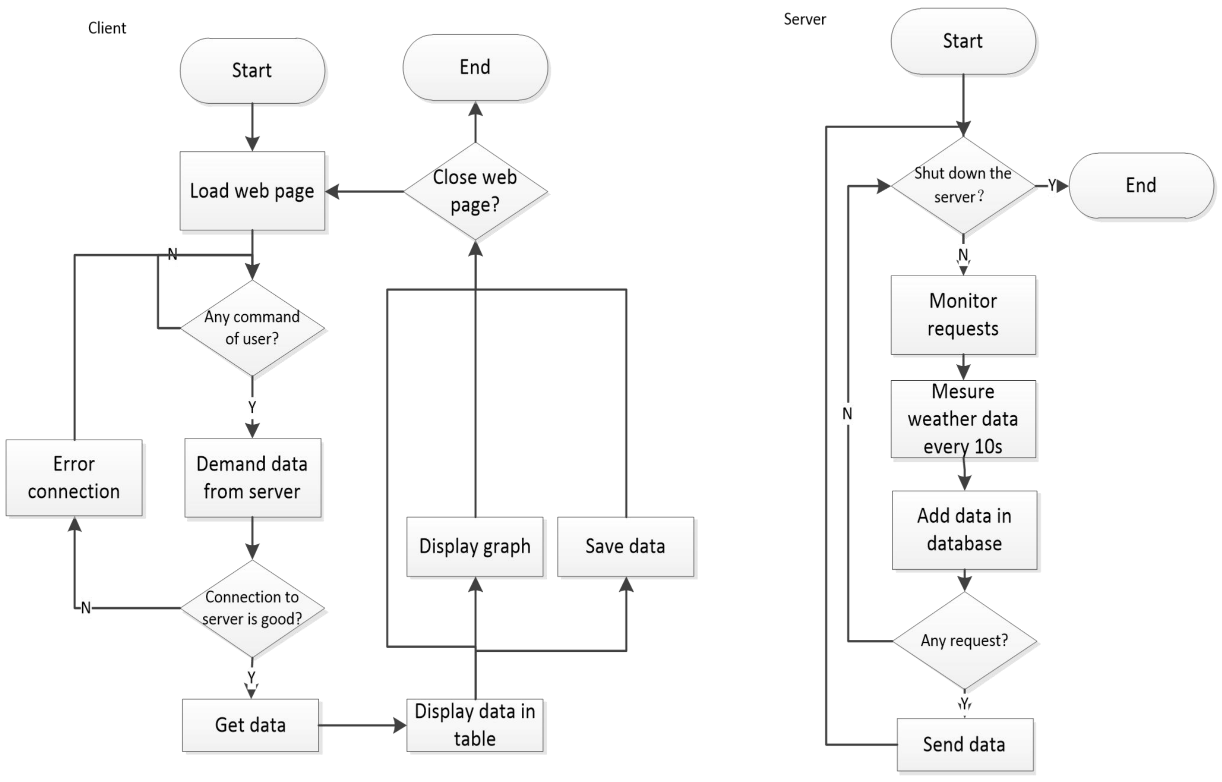
As shown in the use case diagram above, there are three actors in weather measurement system, users, administrator and captures of Sense Hat. The administrator has more authorities than users. The administrator can modify data and monitor the server. The users only have authorities to get weather information by browser. Sense Hat is out of the system because it is an entity of many captures. These real weather data could be changed to the simulated data that does not affect the use of the system.

3.4 Data flow diagram



The system measures initial data from Sense Hat and combine data with system time. The new data was identified by system time. Every ten second, a new data will be saved in database or file. The server will monitor users’ requests and send right data as JSON format. In client side, the system will execute different functions according to different commands.

3.5 Procedure flow diagram



As shown in the procedure flow diagram above, the server constantly monitors requests and adds new data every ten seconds. And the client constantly waits for users' commands, send requests to server and receive data from server.

4.Description developments

4.1 Server

The Web server is the entity that makes the connection between users and system. The server is an important part of weather measure system.

Because of the good support to python, i built the server with python for Raspberry Pi. After a comparison with different framework, such as Django, socket... I find Flask is the best one for me and my project.

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed. Flask is called a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

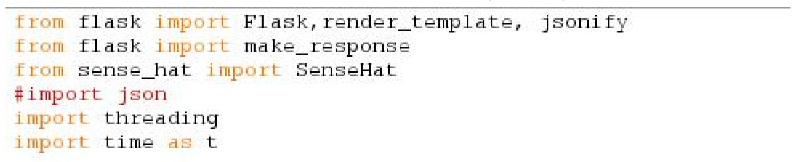
As a micro web framework, Flask is simple to study and use, it's small and it's suitable for small-scale development in a card-sized computer, like Raspberry Pi.

The most basic server structures are as follows:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello, World!"

if __name__ == '__main__':
    app.run(debug=True)
```

This is the simplest server, and it's easy to understand and develop.

For my own server, i need to use Sense Hat to get data from captures, and send the data to users as JSON format. So firstly, i imported some necessary modules:



```
from flask import Flask,render_template, jsonify
from flask import make_response
from sense_hat import SenseHat
#import json
import threading
import time as t
```

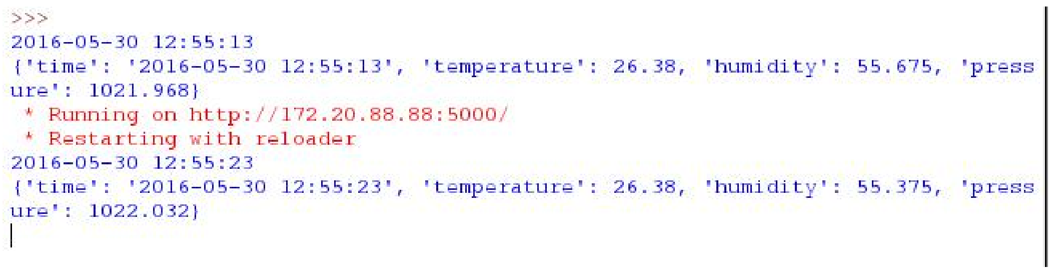
"render_template" is used for displaying a static web page and transferring some parameter, it could be used as follow:

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/infos')
def get_infos():
    return render_template('infos.html',datatest=data1,dataStr=data,
                           data=jsonify({'data':data}))
```

“jsonify” is used for package JSON data. It’s carried by Flask. ”time” is used for getting system time which will identify different data. In my project, i defined a list “data” for saving data, it has two initial values for testing:

```
data=[
    {
        'time': u'2017-08-08 15:30:00',
        'temperature': u'100',
        'pressure': u'101.325',
        'humidity': u'1'
    },
    {
        'time': u'2017-08-08 16:30:00',
        'temperature': u'-100',
        'pressure': u'103.555',
        'humidity': u'99'
    }
]
```

And when the server is running, it can measure the real data from Sense Hat and insert them into this list. As follow, the server can get a new value every ten seconds:



```
>>>
2016-05-30 12:55:13
{'time': '2016-05-30 12:55:13', 'temperature': 26.38, 'humidity': 55.675, 'press
ure': 1021.968}
 * Running on http://172.20.88.88:5000/
 * Restarting with reloader
2016-05-30 12:55:23
{'time': '2016-05-30 12:55:23', 'temperature': 26.38, 'humidity': 55.375, 'press
ure': 1022.032}
```

There are a lot of functions about Sense Hat and its captures, but in my project, i used get_temperature(), get_humidity() and get_pressure() for the values what i need.

”SenseHat” is the necessary module for using captures in Sense Hat. It requires download and install before using. It could be used as follow:

```
def insert_data():
    sense = SenseHat()
    ISOTIMEFORMAT='%Y-%m-%d %X'
    temperature = round(sense.get_temperature(),3)
    humidity = round(sense.get_humidity(),3)
    pressure = round(sense.get_pressure(),3)
    time = t.strftime(ISOTIMEFORMAT)
    sensedata={'time':time,'temperature': temperature,
            'pressure': pressure,
            'humidity': humidity}
    data.append(sensedata)
    print(time)
    print(sensedata)

    count = threading.Timer(10.0,insert_data)
```

I used “round()” for a beautiful display. The module “threading” is used for automatic execution of data measurement.

For the data interaction between server and browser, i used “GET” because this system don’t allow users to modify data in server for system security. The code is as follow:

```
@app.route('/data',methods=['GET'])
def get_data():
    return jsonify({'data':data})
```

When the server receives the “GET” request from browsers, it would package and send weather data.

4.2 Client (Browser)

4.2.1 Connection with server

On the client side, i used browser. The default browser of Raspberry Pi can run HTML5 and JavaScript, moreover, other advanced browsers, such as FireFox, can also work in Raspberry Pi.

The first step is to connect to the server. I used the class “XMLHttpRequest” to establish connection.

```
<script type="text/javascript">
  var jsondata;
  //var jsontext;
  function get_data(){
    var xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function(){
      if (xmlhttp.readyState==4 && xmlhttp.status==200){
        jsondata=JSON.parse(xmlhttp.responseText);
          ......
      }
    }
    xmlhttp.open("GET","/data",true);
    xmlhttp.send();
  }
</script>
```

As the code above, i created an object “xmlhttp”, connected it to the server, and used “GET” to obtain “responseText”(weather data as JSON format) . After that, i used JSON data analysis function ”JSON.parse” to get real weather data.

4.2.2 Dynamic table

After connecting to server, i got JSON data about the weather information. And then, i created a table dynamic to put the data. This table is a uncertain size one which can modify its rows according to the weather data from server.

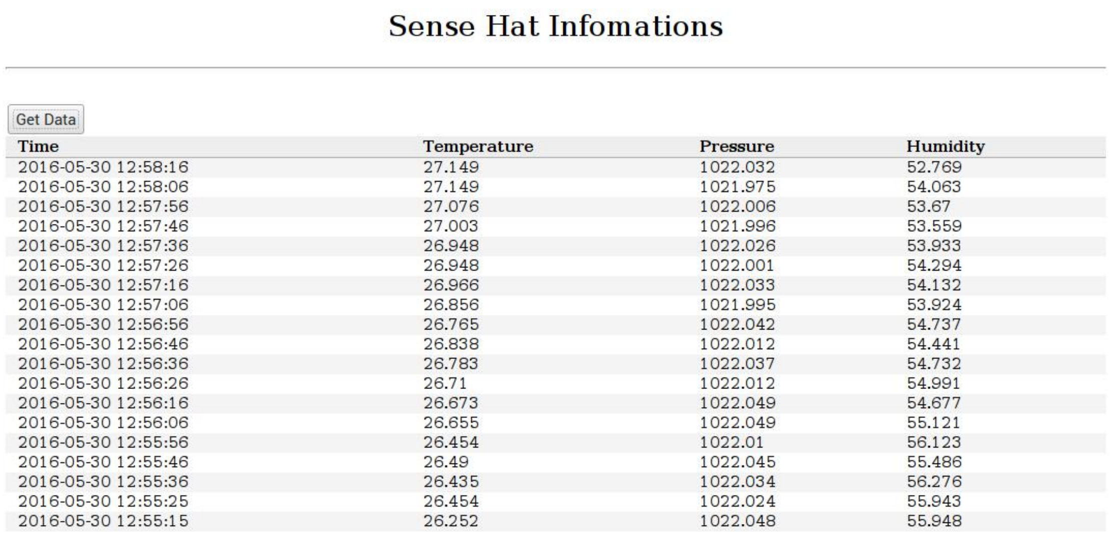
To achieve this goal, i inserted following code to script:

```
    var table = document.getElementById("MainTable");
    var rowcount = table.rows.length;
    for (var j=1;j<rowcount;j++){ table.deleteRow(1);}
    //---add data in table---//
    for(var i=0;i<jsondata.data.length;i++){
        //document.getElementById("Time1").innerHTML=jsondata.data[i].time;
        var u = jsondata.data;
        var tab = document.getElementById("MainTable");
        var rows= tab.rows;
        var row1=tab.insertRow(1);
        row1.insertCell(0).innerHTML="&nbsp;"+u[i].time;
        row1.insertCell(1).innerHTML="&nbsp;"+u[i].temperature;
        row1.insertCell(2).innerHTML="&nbsp;"+u[i].pressure;
        row1.insertCell(3).innerHTML="&nbsp;"+u[i].humidity;
```

This method firstly deletes all the table content before getting new data. And then it requests new data and it inserts new data in the first row. In addition to the part of JavaScript, i added some HTML code for achieving a dynamic table :

```
<div>
<button onclick="get_data()">Get Data</button>
<table id="MainTable" width="100%" border="0" cellspacing="0" cellpadding="0" class="table0">
    <tr>
      <th>&nbsp;<b>Time</b></th>
      <th>&nbsp;<b>Temperature</b></th>
      <th>&nbsp;<b>Pressure</b></th>
      <th>&nbsp;<b>Humidity</b></th>
    </tr>
</table>
</div>
```

I used button click to get data from server. Now i can get the weather information from server. The result as follow:


Sense Hat Infomations

Get Data

Time	Temperature	Pressure	Humidity
2016-05-30 12:58:16	27.149	1022.032	52.769
2016-05-30 12:58:06	27.149	1021.975	54.063
2016-05-30 12:57:56	27.076	1022.006	53.67
2016-05-30 12:57:46	27.003	1021.996	53.559
2016-05-30 12:57:36	26.948	1022.026	53.933
2016-05-30 12:57:26	26.948	1022.001	54.294
2016-05-30 12:57:16	26.966	1022.033	54.132
2016-05-30 12:57:06	26.856	1021.995	53.924
2016-05-30 12:56:56	26.765	1022.042	54.737
2016-05-30 12:56:46	26.838	1022.012	54.441
2016-05-30 12:56:36	26.783	1022.037	54.732
2016-05-30 12:56:26	26.71	1022.012	54.991
2016-05-30 12:56:16	26.673	1022.049	54.677
2016-05-30 12:56:06	26.655	1022.049	55.121
2016-05-30 12:55:56	26.454	1022.01	56.123
2016-05-30 12:55:46	26.49	1022.045	55.486
2016-05-30 12:55:36	26.435	1022.034	56.276
2016-05-30 12:55:25	26.454	1022.024	55.943
2016-05-30 12:55:15	26.252	1022.048	55.948

The newest data is always in the first row.

4.2.3 Graphical representation of data

The data presented in the form of charts, looks more intuitive. The weather changes can also be clearly seen with a chart. So i tried to use Chart.js to achieve graphical representation of data.

The first method is using Chart.js. Chart.js is a open source JavaScript Chart Library. There are six types charts based on HTML5.

Chart.js provides two different builds that are available for your use. The Chart.js and Chart.min.js files include Chart.js and the accompanying color parsing library. If this version is used and it requires the use of the time axis, Moment.js will need to be included before Chart.js. The Chart.bundle.js and Chart.bundle.min.js builds include Moment.js in a single file. This version should be used if it requires time axes and want a single file to include, select this version. Do not use this build if the application already includes Moment.js. If do, Moment.js will be included twice, increasing the page load time and potentially introducing version issues.

Chart.js requires to be imported in a script tag before using:

```
<script src="Chart.js"></script>
<script>
    var myChart = new Chart({...})
</script>
```

Not only local Chart.js file path, but also the internet path could be used as source. For example, in my project, i used the source as follow:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/1.0.2/Chart.min.js">
</script>
```

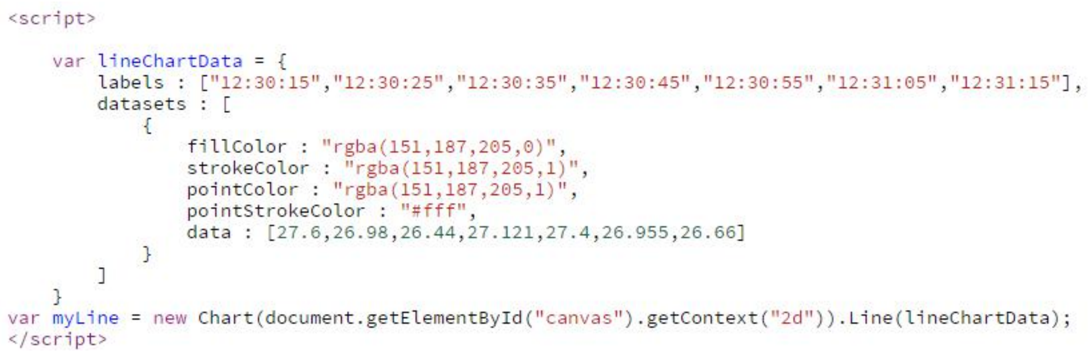
And it worked well when my Raspberry Pi could connect to the internet.

To display data, the chart must be passed a data object that contains all of the information needed by the chart. The data object can contain the following parameters.

Name	Type	Description
datasets	Array[object]	Contains data for each dataset. See the documentation for each chart type to determine the valid options that can be attached to the dataset
labels	Array[string]	Optional parameter that is used with the category axis.
xLabels	Array[string]	Optional parameter that is used with the category axis and is used if the axis is horizontal
yLabels	Array[string]	Optional parameter that is used with the category axis and is used if the axis is vertical

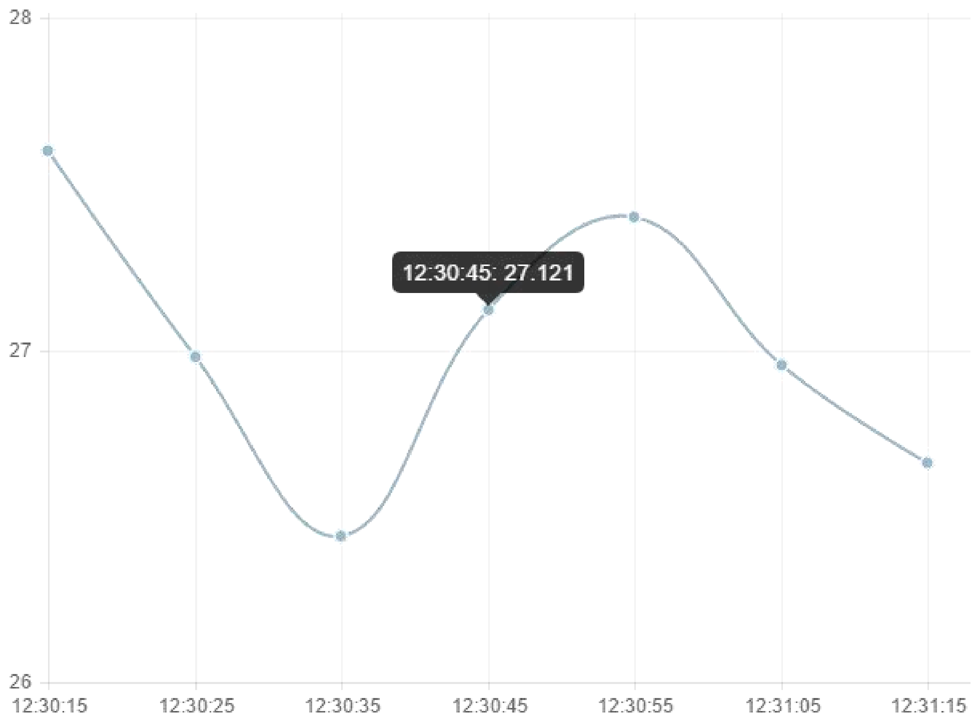
In six types of charts, i chose line chart for my project because line chart could display a clear changes for temperature, pressure and humidity. In line chart, the parameters “labels” and “datasets” are used.

The code of line chart is as follow:



```
<script>

    var lineChartData = {
        labels : ["12:30:15","12:30:25","12:30:35","12:30:45","12:30:55","12:31:05","12:31:15"],
        datasets : [
            {
                fillColor : "rgba(151,187,205,0)",
                strokeColor : "rgba(151,187,205,1)",
                pointColor : "rgba(151,187,205,1)",
                pointStrokeColor : "#fff",
                data : [27.6,26.98,26.44,27.121,27.4,26.955,26.66]
            }
        ]
    }
var myLine = new Chart(document.getElementById("canvas").getContext("2d")).Line(lineChartData);
</script>
```

This is my test example to explain Chart.js. In my project, i changed the values of "labels" and "data" with real measuring data. And the result of execution is a beautiful line chart：



Users can click a button in the web page to get the line chart. The function of creating line chart calls the function of requesting data from the server. So the line chart would change every ten seconds like the dynamic table.

5.Test

In this part, i will present some details in test.

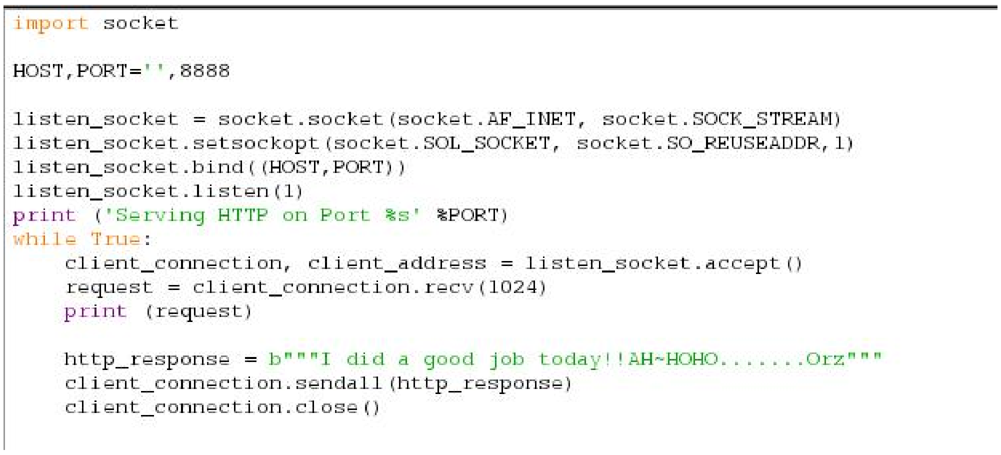
5.1 Hardware test

The first test is about the affect of current capacity to Raspberry Pi. When I used the power cable connected to a laptop, all the functions of Raspberry worked well. However, when there are a lot of external devices in Raspberry Pi, such as Sense Hat, keyboard and mouse, some functions of Raspberry Pi would stop. Usually, there would be some problems about network connection. So it's better to choose energy-efficient peripherals or use SSH to access Raspberry Pi.

The second test is about Sense Hat. I followed a Sense Hat tutorial to develop some interesting examples about Joystick. Unfortunately, i can't import Joystick module in my python program, even if i have installed it.

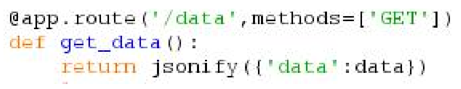
5.2 Software test

The first software test is about the server framework selection. At the beginning, i had no idea about framework Flask, i didn't know Flask could work or not. So i used socket for testing network when i developed Flask server.



```
import socket

HOST,PORT='',8888

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 %s' %PORT)
while True:
    client_connection, client_address = listen_socket.accept()
    request = client_connection.recv(1024)
    print (request)

    http_response = b"""I did a good job today!!AH~HOHO.......Orz"""
    client_connection.sendall(http_response)
    client_connection.close()
```

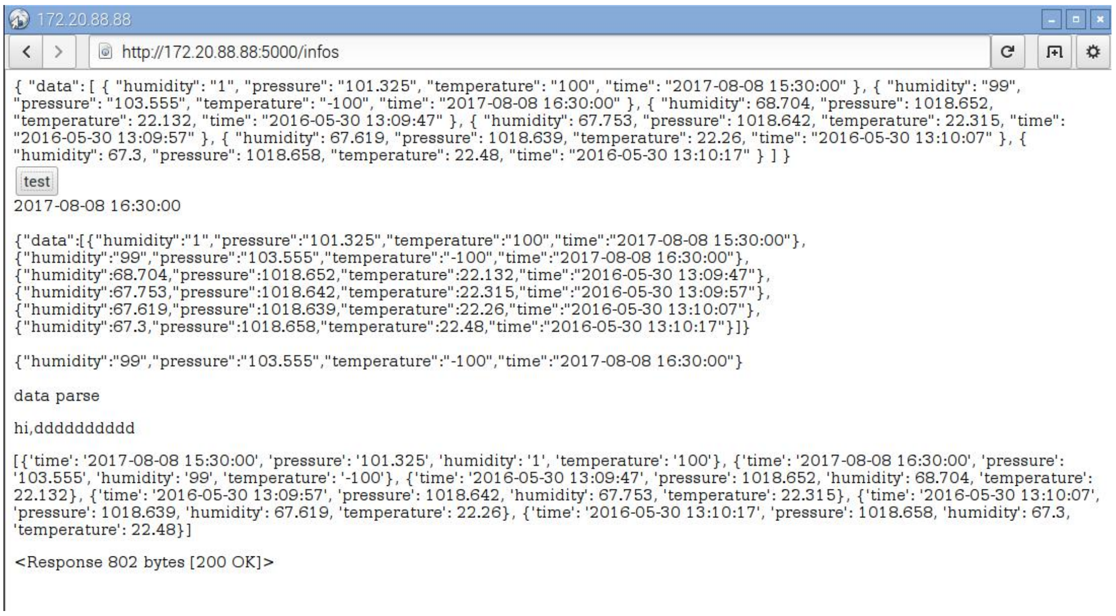
Another important test is about JSON data and communication between server and browser. After deciding to use Flask framework, the methods of sending JSON data is not many on server side. I can send a static web page, send a static web page with some parameters or send a JSON object.



```
@app.route('/data',methods=['GET'])
def get_data():
    return jsonify({'data':data})
```

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/infos')
def get_infos():
    return render_template('infos.html',datatest=data1,dataStr=data,
                           data=jsonify({'data':data}))
```

However, on client side, there are a lot of ways to receive JSON data. The problem is that the selected method must match the server. So i used a test web page to test content, format, use of all data and test data receive mode.



```
{ "data": [ { "humidity": "1", "pressure": "101.325", "temperature": "100", "time": "2017-08-08 15:30:00" }, { "humidity": "99",
"pressure": "103.555", "temperature": "-100", "time": "2017-08-08 16:30:00" }, { "humidity": 68.704, "pressure": 1018.652,
"temperature": 22.132, "time": "2016-05-30 13:09:47" }, { "humidity": 67.753, "pressure": 1018.642, "temperature": 22.315, "time":
"2016-05-30 13:09:57" }, { "humidity": 67.619, "pressure": 1018.639, "temperature": 22.26, "time": "2016-05-30 13:10:07" }, {
"humidity": 67.3, "pressure": 1018.658, "temperature": 22.48, "time": "2016-05-30 13:10:17" } ] }

2017-08-08 16:30:00

{"data":[{"humidity":"1","pressure":"101.325","temperature":"100","time":"2017-08-08 15:30:00"},
{"humidity":"99","pressure":"103.555","temperature":"-100","time":"2017-08-08 16:30:00"},
{"humidity":68.704,"pressure":1018.652,"temperature":22.132,"time":"2016-05-30 13:09:47"},
{"humidity":67.753,"pressure":1018.642,"temperature":22.315,"time":"2016-05-30 13:09:57"},
{"humidity":67.619,"pressure":1018.639,"temperature":22.26,"time":"2016-05-30 13:10:07"},
{"humidity":67.3,"pressure":1018.658,"temperature":22.48,"time":"2016-05-30 13:10:17"}]}

{"humidity":"99","pressure":"103.555","temperature":"-100","time":"2017-08-08 16:30:00"}

data parse

hi,ddddddddd

[{'time': '2017-08-08 15:30:00', 'pressure': '101.325', 'humidity': '1', 'temperature': '100'}, {'time': '2017-08-08 16:30:00', 'pressure':
'103.555', 'humidity': '99', 'temperature': '-100'}, {'time': '2016-05-30 13:09:47', 'pressure': 1018.652, 'humidity': 68.704, 'temperature':
22.132}, {'time': '2016-05-30 13:09:57', 'pressure': 1018.642, 'humidity': 67.753, 'temperature': 22.315}, {'time': '2016-05-30 13:10:07',
'pressure': 1018.639, 'humidity': 67.619, 'temperature': 22.26}, {'time': '2016-05-30 13:10:17', 'pressure': 1018.658, 'humidity': 67.3,
'temperature': 22.48}]

<Response 802 bytes [200 OK]>
```

There are eval() and JSON.parse() to parse JSON data. but eval() can not work in my project because its parse method doesn't match the data format from the server, i can not get any useful data.

The response text of the object of XMLHttpRequest can match the server data format and match the function parse() of JSON module.

```
var xmlhttp=new XMLHttpRequest();
xmlhttp.onreadystatechange=function(){
  if (xmlhttp.readyState==4 && xmlhttp.status==200){
    jsondata=JSON.parse(xmlhttp.responseText);
```

For testing line chart, i used a web site, JS Bin (jsbin.com) to test HTML and JavaScript code. JS Bin is a online web page editing platform. All the code of line chart works well on JS Bin. But if i use these code in Raspberry Pi, it doesn't work, i can not get any chart in web page. I think there is not a good support of graphics processing capabilities in Raspberry Pi, maybe it requires installing some other modules to support these functions.

Conclusion

This project is a good professional discovery for me. I began to try to develop embedded system in this project. As a beginner in the field of embedded system, i met a lot of challenges. Fortunately, with the help of my professors, i solved most problems and achieved a weather measure system which has some basic functions. In fact, the system is a little simple. There are not enough functions for users. But this system achieves REST communication between server and browser which is the main purpose of the project. These problems and experiences of development will help me learn and work in future. If possible, i will continue to improve this system.

References

https://www.raspberrypi.org/
https://pythonhosted.org/sense-hat/api/
http://wiki.jikexueyuan.com/project/raspberry-pi/
http://www.geekfan.net/4552/
http://www.cnblogs.com/
https://sites.google.com/site/debianpackageshare/Home/linux-tips/page-47
http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xvii-deployment-o
n-linux-even-on-the-raspberry-pi
https://docs.python.org/3/
https://jsbin.com/
http://www.alloyteam.com/2014/01/use-js-file-download/
http://blog.csdn.net/huaweidong2011/article/details/17271067
http://www.chartjs.org/docs/
http://caibaojian.com/json-length.html
http://www.w3school.com.cn/
https://astro-pi.org/get-involved/program-the-sense-hat/python/
http://simplejson.readthedocs.io/en/latest/