Kirirom Weather Station

Version 0.1

DEVELOPED BY

DUK PANHAVAD (BATCH3)

HOUERN LYHOV (BATCH3)

Kirirom Weather Station V0.1

JUN 13, 2019

TEAM - Capsule
WEEK04 - Projects

Members:
Duk panhavad
Heourn Lyhov

This is written to make understand of how this project created.

On Jun 10, 2019 all **KIT-Bootcamp** students are asking to form and team and come up with their own idea of any python project they would like to do. So we form a team of 3 which have **Mr.Kevin Sabbe** as the team coordinator. After a short discussion in team we have come up with 2 projects and **Kirirom Weather Station** is the first.

During the initial creation of the **Kirirom Weather Station V0.1** we are targeting a specific goal to achieve in this version such as:

- Collecting almost accurate data of Temperature, Humidity and Rain in vKirirom region
- Utilize all the resources of hardware and software we currently have without any purchase proposal
- Provide a usable API that could receive live data from board and provide data through API to our client (public)
- Use almost 100% of python for the project

Finally, after all the hard work, effort and massive research (~35hours). On Jun 13, 2019 we have successfully complete the **Kirirom Weather Station V0.1** which satisfy all the goals mentioned above. And we realize that this might be a good project to bring to official in **Kirirom Institute of Technology** on **Kirirom Weather Station V1.1.** which will be better than the previous version. And the goal of **Kirirom Weather Station V1.1** does not decide yet but will be listed at the end of the document.

Yours Truly,

Duk Panhavad

Table of Contents

1. Project Overview

Overview information of the whole project to give an idea what is the project about.

All source code is available here: https://github.com/panhavad/kwsv0.1

A. Requirement for Version 0.1

a. Collecting almost accurate data of Temperature, Humidity and Rain in vKirirom region
b. Utilize all the resources of hardware and software we currently have without any purchase proposal
c. Provide a usable API that could receive live data from board and provide data through API to our client (public)
d. Use almost 100% of python for the project
e. Create a weather data station of Kirirom that anyone can query

B. Main Tasks & It's Goal

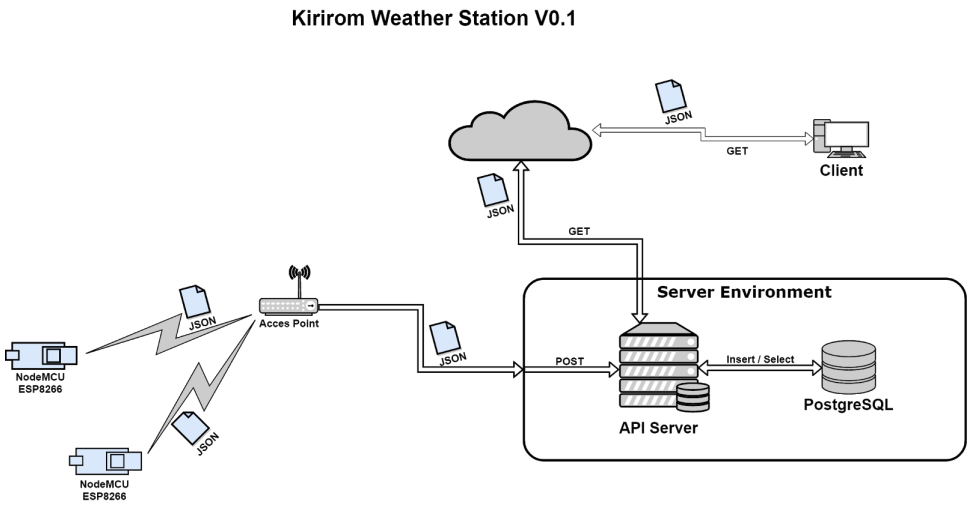
a. Programming board and hardware feasibility study
    i. Goal: All component connected and working with code uploaded and placed in the desired location
b. Data allocation
    i. Goal: Successfully store all the data push by board to PostgreSQL
c. Data storing technique
    i. Goal: Store all the existing data in a proper form that might be easily for API query and make sure all the data in almost accurate
d. API Creation:
    i. Goal: Can push and fetch data to database freely without any technical skills required

C. Project Obstacles

a. Lack of hardware components
b. Python programming on Microcontroller and API creation is the team first time try

1.1. How its work?

A. Overall System Diagram



a. POST Data

i. Board (NodeMCU) Connected to the 2 sensor that will provide enough information about weather we need for this version

ii. Board (NodeMCU) also connected to network through WIFI module and obtain its own IP for each board

iii. Every one second board collected data from sensor and push to server using HTTP protocol with POST methods, pass inform of JSON file

iv. API Server will handle the JSON file extract everything by executing ETL Pipeline every one minute to store the final information to database

b. GET Data

i. Any client can request for Kirirom weather information using HTTP protocol GET methods

ii. API Server will handle the request and do the query on PostgreSQL to get the related result that client need

iii. Pass the query information to client in form of JSON using HTTP response protocol

2. Requirement for “Weather Station V0.1”

This is the list of skills, software, hardware that will be needed to complete this **Kirirom Weather Station V0.1** project.

A. Skills Set

a. Basic electronics knowledge
b. Researching
c. Data manipulation knowledge
d. Python (Importance)
e. MicroPython (Importance)
f. Shell Scripting (Optional)

B. Software Components

a. Non-Server

i. **uPyCraft** - is an IDE for programing microcontroller like NodeMCU ESP8266 in MicroPython language (Importance)
    1. Download: https://github.com/DFRobot/uPyCraft
ii. **Postman** - is used to test the API (Optional)
iii. **Visual Paradigm** - for easy schema design and generate table directly from schema to database (Optional)
iv. **Sublime Text 3** - as a debug tool when we had indent problems (Optional)
v. **pgAdmin3** - used as a database browser to see the data (Optional)
vi. **Bitvise** - easy access to the server file and command line interface (Optional)

b. Server

i. **Ubuntu 16.04 Server** - Main operating system for API and Database server
ii. **Flask** - API server backend (Server - Importance)
iii. **PostgreSQL10** - Relational database to store all collected data (Server - Importance)
iv. **Python2.7** - Main language using for API backend and Board (Server - Importance)
v. **Gunicorn, Supervisor, Ngnix** - Handle the API server processing (Server - Importance)

C. **Hardware Components**

a. **NodeMCU ESP8266** - Main board that going to interacting will all sensor and API server
   i. Documentation: https://www.handsontec.com/pdf_learn/esp8266-V10.pdf
b. **DHT-11** - Temperature and Humidity sensor for collecting the related data
   i. Documentation: https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
c. **Water Sensor Module** - Detect and give the value of value that touch on the sensor surface
   i. Documentation: https://drive.google.com/open?id=1Op3B4car_tkfgaLv0c49mQRkshPOENr3
d. **Adaptor MicroUSB 5v 2A** - Power supply for NodeMCU ESP8266 in order to work properly
e. **Jumper wire** - Need around 6 to 10 Jumper wire

3. Implementation

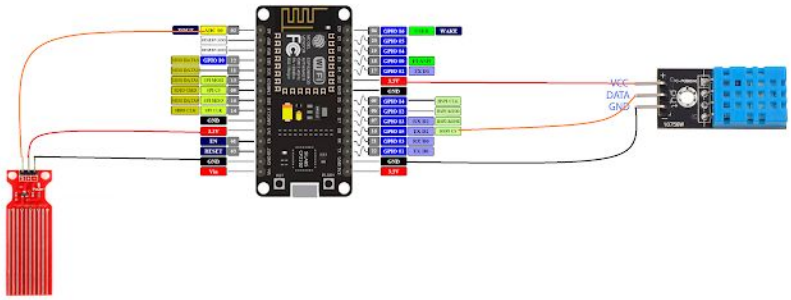
This is the implementation of the hold project step by step both hardware and software.

A. **Board Setting Up**

a. **Circuit Connection**

   i. **High Sensitivity Water Sensor** - 3 pins connection:
      1. **Ground (-)** --> connect to **Pin G** on Board
      2. **Voltage (+)** --> connect to **Pin 3.3v** on Board
      3. **Signal (s)** --> connect to **Pin A0** on Board
   ii. **DHT-11** (Temperature & Humidity) - 3 pins connection:
      1. **Ground (-)** --> connect to **Pin G** on Board
      2. **Voltage (+)** --> connect to **Pin 3.3v** on Board
      3. **Signal (s)** --> connect to **Pin D7(GPIO13)** on Board

b. **Circuit Connection Diagram** (Figure1.0)



c. **Board Setup**

i. **Connect** (NodeMCU 8266) to computer
ii. Open **uPyCraft IDE**
iii. Select **COM Port** correspond to board
iv. If **first time** for board to program with uPyCraft, Flash - **Flash** new image from uPyCraft by select **ESP8266** and Choose **Burn**
v. Open the existing program and change code in program in statement that have comment **#Change here.**
vi. **Upload file** (file below) to board and set that program file to **default** run so it will start that program after the board start
vii. Board are **ready** to use

d. **Program for Board** (MicroPython)

This is the program in board that we use to get data from sensor and post data to API server.

Source: https://github.com/panhavad/kwsv0.1/blob/master/kws-board.py

```
from machine import Pin, ADC
from dht import DHT11
import utime, ujson, network, urequests

ANALOG_PIN = 0 #A0
BULDIN_LED_PIN = 2
DHT11_PIN = 13 #D7
LOCATION = "Borey R - J37" #Change here
```

```
def post_data(payload):
  """ Post the data to API server"""

  url = 'http://10.10.11.137/posts/weather/v0.1' #Change here
  headers = {'content-type': 'application/json'} #Allow to pass JSON data with HTTPS protocol

  try:
    respond = urequests.post(url, data = payload, headers = headers)
    respond.close()

    print("Data posted.")

    return True
  except Exception as e:
    print("Data posted Error!!")

    return False

def ap_mode(mode):
  """" Turn off the AP mode on ESP8266"""

  wifiap = network.WLAN(network.AP_IF)
  wifiap.active(mode)
  if wifiap.active() == False:
    print("AP-Mode Diable Successfully :)")

def connect_wifi(ssid, password):
  """ Collecting to the network with pre-define ssid and password"""

  station = network.WLAN(network.STA_IF)

  if station.isconnected() == True:
      print("Network connected - IP:", station.ifconfig()[0])
      return

  station.active(True)
  station.connect(ssid, password)

  while station.isconnected() == False:
      pass

  print("Connection successful")
  print(station.ifconfig())

def temp_humi_reader():
    """ Give the temperature in Celcius and Humidity in % """

    temp_humi_sensor = DHT11(Pin(DHT11_PIN))
    temp_humi_sensor.measure()

    return {
        "temp" : temp_humi_sensor.temperature(),
        "humi" : temp_humi_sensor.humidity()
    }

def water_reader():
```

```
    """ Give the status of rain (True or False) """

    digital_input = ADC(0)
    water_level = digital_input.read()
    status = 0

    if water_level <= 100:
      status = 0
    else:
      status = 1

    return {
      "status" : status,
      "level" : water_level
    }

def led_working(status):
    """ LED feedback of the working status blink 0.5sec """

    led = Pin(BULDIN_LED_PIN, Pin.OUT)

    if status:
      led.value(1)#turn off
      utime.sleep(0.5)
      led.value(0)#turn on
      utime.sleep(0.5)
    else:
      for i in range(3):
        led.value(1)#turn off
        utime.sleep(0.1)
        led.value(0)#turn on
        utime.sleep(0.1)

def collect_data():
  """Collecting temp, humi, and rain status and return as JSON"""

  data_package = dict()
  if temp_humi_reader() and water_reader():
    data_package["dht11"] = temp_humi_reader()
    data_package["rain"] = water_reader()
    data_package["location"] = LOCATION

    return ujson.dumps(data_package)
  else:
    print("!Error!")

    return False

#START HERE
if __name__ == "__main__":
  """ Program start from here"""

  #diable the ap mode on ESP
  ap_mode(False)

  #connect to the network
```

```
ssid = "Borey R-J37" #Change here
password = "" #Change here
connect_wifi(ssid, password)
running = True

while running:
  data = collect_data()
  status = post_data(data)
  led_working(status)
```

B. API Creation

a. Config File

This is the Config file Template for **api.py**

```
{
        "DBHOST" : "DBHOST",
        "DBPORT" : "5432",
        "DBUSER" : "postgres",
        "DBPASS" : "admin",
        "DBNAME" : "DB_Name"

}
```

b. API Python program

This is the code of API server that use for POST and GET the data between postgreSQL and client:
Source:
https://github.com/panhavad/kwsv0.1/blob/master/api/kws_api.py

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow
from datetime import *
import os, psycopg2, json


#Init app
app = Flask(__name__)

with open('config.json') as config_file:
        config = json.load(config_file)

def db_connect():
        try:
           connection = psycopg2.connect(user = config.get('DBUSER'),
                                password = config.get('DBPASS'),
                                host = config.get('DBHOST'),
```

```
                                    port = config.get('DBPORT'),
                                    database = config.get('DBNAME'))
            cursor = connection.cursor()
            # print ( connection.get_dsn_parameters(),"\n")
            return connection, cursor
        except (Exception, psycopg2.Error) as error :
            return False

def db_close(connection, cursor):
        if(connection):
                cursor.close()
                connection.close()
                print("PostgreSQL connection is closed")

@app.route('/gets/weather/v0.1/historical', methods = ['GET'])
def get_historical():
        """ Give the historical weather data in between the given date """
        if len(request.args) > 0:#check if the request come with params or not
                res_json = dict()

                try:#exception occur when wrong params was given
                        from_date = request.args['from_date']
                        to_date = request.args['to_date']

                        select_query = "select * from weather_station where \"datetime\"
>= '" + from_date + "' and \"datetime\" <= '" + to_date + "'"
                        cursor.execute(select_query)
                        res = cursor.fetchall()

                        for row in res:
                                datetime, temperature, humidity, rstatus, rlevel, location
= str(row[1]), row[2], row[3], row[4], row[5], row[6]
                                res_json[datetime] = { "temperature" : temperature,
                                                       "humidity" : humidity,
                                                       "rstatus" : rstatus,
                                                       "rlevel" : rlevel,
                                                       "location" : location }
                        return jsonify(res_json)
                except Exception as e:
                        print(e)
                        return jsonify({ "code" : 400})

        return jsonify({ "code" : 200})

@app.route('/gets/weather/v0.1/lastest', methods = ['GET'])
def get_lastest():
        """ Give the lastes weather data limited with the given amount """
        if len(request.args) > 0:
                res_json = dict()
                try:
                        count = request.args['count']

                        select_query = "select * from weather_station order by id desc limit
" + count
                        cursor.execute(select_query)
                        res = cursor.fetchall()
```

```
                        for row in res:
                            datetime, temperature, humidity, rstatus, rlevel, location
= str(row[1]), row[2], row[3], row[4], row[5], row[6]
                            res_json[datetime] = { "temperature" : temperature,
                                                   "humidity" : humidity,
                                                   "rstatus" : rstatus,
                                                   "rlevel" : rlevel,
                                                   "location" : location }
                        return jsonify(res_json)
                    except Exception as e:
                        print(e)
                        return jsonify({ "code" : 400})

        return jsonify({ "code" : 200})


@app.route('/posts/weather/v0.1', methods = ['POST'])
def main():
        """ Process the recieved data from board """
        post_storing_datetime = storing_datetime()
        post_actual_datetime = current_datetime()
        post_temp = request.get_json().get('dht11').get('temp')
        post_humi = request.get_json().get('dht11').get('humi')
        post_rstatus = request.get_json().get('rain').get('status')
        post_rlevel = request.get_json().get('rain').get('level')
        post_location = request.get_json().get('location')

        posted_data = [ (post_storing_datetime, post_actual_datetime, post_temp, post_humi,
post_rstatus, post_rlevel, post_location) ]
        insert_query = """ INSERT INTO weather_all(storing_datetime, actual_datetime,
temperature, humidity, rain_status, rain_level, location)
                VALUES (%s, %s, %s, %s, %s, %s, %s) """
        # executemany() to insert multiple rows rows
        result = cursor.executemany(insert_query, posted_data)
        connection.commit()

        return jsonify({"code" : 200})


def current_datetime():
        current_datetime = datetime.now()
        str_current_datetimestamp = current_datetime.strftime("%Y-%m-%d %H:%M:%S")
        return str_current_datetimestamp

def storing_datetime():
        current_datetime = datetime.now()
        str_current_datetimestamp = current_datetime.strftime("%Y-%m-%d %H:%M:00")
        return str_current_datetimestamp

connection, cursor = db_connect()

#Run server
if __name__ == '__main__':
        app.debug = True
        connection, cursor = db_connect()
```

```
host = os.environ.get('IP', '0.0.0.0')
port = int(os.environ.get('PORT', 8019))
app.run(host=host, port=port)
```

B. Server setting up

a. Install **Ubuntu 16.04 Server** onto the server instance
b. Install **Python2.7**
c. Install **PostgreSQL10** (Remember the Username and Password because we are going to authenticate in API python file)
d. Clone **API project** to the home directory
e. Inside API Project folder, create a **virtual environment** for the project
f. **Activate** the **virtual environment** that just created
g. Run this to do pip install all **dependencies**:

```
pip install flask flask-sqlalchemy flask-marshmallow
marshmallow-sqlalchemy
```

```
python -m pip install --trusted-host pypi.org --trusted-host
files.pythonhosted.org --trusted-host pypi.python.org psycopg2
```

h. Install **nginx, gunicorn** to handle all the start and stop process of our API:

```
sudo apt install nginx
pip install gunicorn
```

i. Remove the **default** file in nginx

```
sudo rm /etc/nginx/sites-enabled/default
```

j. Create our own file called **kws file**

```
sudo nano /etc/nginx/sites-enabled/kws
```

Place this following content in **kws file**:

```
server {
        listen 80;
        server_name 10.10.11.137;

        location / {
                proxy_pass http://localhost:8000;
                include /etc/nginx/proxy_params;
                proxy_redirect off;
        }
}
```

k. Make sure the **firewall rule** is **disable** to prevent any problem from accessing outside

l. Create a config file called **kws.config**

```
sudo nano /etc/supervisor/conf.d/kws.conf
```

Place the following content in **kws.config**

```
[program: Kirirom_Weather_Station]
directory=/home/nextcloud/kws_api
command=/home/nextcloud/kws_api/venv/bin/gunicorn -w 9
run:app
user=nextcloud
autostart=true
autorestart=true
stopasgroup=true
killasgroup=true
stderr_logfile=/var/log/kws/kws_api.err.log
stdout_logfile=/var/log/kws/kws_api.out.log
```

m. Finally run this 3 command to make sure our **log files** is created

```
sudo mkdir -p /var/log/kws
sudo mkdir -p /var/log/kws/kws_api.err.log
sudo mkdir -p /var/log/kws/kws_api.out.log
```

4. Technical Issues

These are all the most difficult and tricky technical issue we are facing during testing and deployment.

A. Issue01 - when run program on the board it **POST** data not more than 5 requests and then it display **LimacBlk:1** after that board cannot connect to wifi until we reset board.

a. ***Solution*** - Because of request that we used for **POST** data using HTTP Request is not close() so it can't handle more than 5. We just create variable **response** of request and say **response.close()** to close the request object.

5. Final Product and Output

A. Product Usage

a. API GET Method

i. Historical Data

API for request **historical data** between two datetime. Accept **2** datetime valid parameters. Return the weather record in between those dates.

1. *from_date → example - 2019-06-12 16:13:00*
2. *to_date → example - 2019-06-12 16:52:00*

```
http://10.10.11.137/gets/weather/v0.1/historical?from_date=2019-06-12 16:13:00&to_date=2019-06-12 16:52:00
```

ii. Latest Data

API for request **latest data** correspond to the number given. Accept **1** parameters as integer. Return the lastes 100 record available

1. *count → example - 100*

```
http://10.10.11.137/gets/weather/v0.1/lastest?count=100
```

iii. Result JSON Format

This is format data that we return as json format.

```
{
   "2019-06-14 00:24:00": {
      "humidity": 13.1923,
      "location": "Kirirom",
      "rlevel": 4,
      "rstatus": 0,
      "temperature": 29.4231
   }
}
```

NOTE: rlevel - is rain level, rstatus - is rain or not if 1 mean rain and 0 not rain

B. Product deployment location

a. Borey V R10
b. Borey R J-37
c. KIT Architecture Class

7. Plan for Kirirom Weather Station V1.0

For the next version of this project is to make the more improvement on the previous version and fill in the missing parameter that the previous project can not complete.

A. Requirement

a. Ensure there are no loss of data even no electricity or network connection
b. Rain meter might be included in this version to understand more about the rain behavior in Kirirom which is the most valuable information of weather
c. Wind meter might be included to monitor the speed and behavior of the wing on mountain, this might be beneficial for other project research in future
d. Make the proper Box/Container and design as real product that will be capable of stay under sun light, rain and strong wind

B. Goals

a. Board capable to monitor and behavior of rain wind temperature and humidity
b. Good quality product and design

C. How to achieve this?

a. Solar and Battery module for Arduino (Buy)
b. High value *capacitor* (Buy)
c. *Rain meter* for arduino (Buy)
d. Board box/container 3D printing
e. API Server with good spec that can handle concurrence request

For **Kirirom Weather Station V1.0** estimate to be **complete** in **2 weeks** after all component is available.

8. Milestones and Reporting

Total hours spent on Kirirom Weather Station v0.1: 45hours

Milestone	Tasks	Hrs	Date
1 - Programming board and hardware feasibility study			
1.1	Research and feasibility study on hardware	1h	2019-06-10
1.2	Choosing hardware components and software to be used	1h	2019-06-10
1.3	Board preparation for Python programming	2h	2019-06-10
1.4	Researching on MicroPython language and its library	5h	2019-06-10
1.5	Start programming the board and debugging	5h	2019-06-10
1.6	Design circuit connection and implementation on breadboard	1h	2019-06-10
1.7	Start coding and testing collecting data from each sensor	3h	2019-06-10
2 - Data allocation			
2.1	Prepare data for sending to server	3h	2019-06-11
2.2	Create API for Post data to server	2h	2019-06-11
2.3	Create Database	2h	2019-06-11
2.4	Build function to store data	3h	2019-06-11
3 - Data Storing Technique			
3.1	Create an ETL pipeline with will allow to manage data much more better	2h	2019-06-11
3.2	Scheduling the ETL job and testing final board implementation	2h	2019-06-11
4 - API Creation			
4.1	Create API Push data to server	3h	2019-06-12
4.2	Create API Fetching data	3h	2019-06-12