An Air Quality Monitoring System

Sakkarin Pumprajum 56090047

Bachelor of Engineering in Software Engineering
International College
King Mongkut's Institute of Technology Ladkrabang
Academic Year 2020

Thesis – Academic Year 2020

Bachelor of Engineering in Software Engineering
International College
King Mongkut's Institute of Technology Ladkrabang

Title: An Air Quality Monitoring System

Authors

Sakkarin Pumprajum Student ID: 56090047

Approved for submission

.....

(Project Advisor Name)
Advisor

Date ...15.../...01.../...2021...

Abstract

Air quality in big cities is getting deteriorated in recent years due to rapid industrial development and the lack of environment protection. This situation puts people living in those polluted cities at risk of health problems. Thus, an air quality monitoring system is necessary for people to prepare and help themselves from the terrible air quality conditions. In this software project, we aim to develop an air quality station system that monitors and reports the air quality information to suit people's need.

In this project, we aim to develop a weather monitoring system based on LoRa and MQTT technology. We develop a weather monitoring station using LoPy microcontroller equipped with sensors to measure weather properties such as temperature, relative humidity, PM_{10} concentration, and $PM_{2.5}$ concentration. Besides of these capabilities, the weather monitoring station includes extra features like tracking the location of the weather station, LoRa communication between weather station and gateway, and analyze weather data to predict future weather using Pandas. By creating our own local LoRa network, we use LoPy microcontroller as the main device both end node and gateway of LoRa system. At the gateway, the received data is sent to a cloud server via MQTT messaging protocol. Then, MQTT cloud server will further send the data to our web application running on a server to log the data on our database server. We represent the data through a web-based application that we develop with Django framework and use Grafana which is visualization tool to perform advanced data visualization and transmission in our front-end website. Finally, We analyze the weather log data using Pandas in order to predict the future weather data and alert to the user.

Our Air Monitoring System is developed using Python applications running on the embedded system part and server part. While the embedded system part consists of two types of hardware devices working together in order to communication with the server part. We also enhance the system with the analyzed function that summarize the statistic of the previous weather data according to our log data which is precisely stored in term of time series data.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

At the present time, there has been great concern among the public over weather and air quality due to pollution and the occurrence of $PM_{2.5}$ covering big cities in many countries such as India [1], China [2] and Thailand [3], which smog is a common phenomenon affecting people lifestyle. Based on this health issue faced by the people in those cities causing the demand for independent information increases significantly. Thus, weather monitor is becoming crucial for to everyone in those polluted cities so that they can prepare and help themselves from the impacts of the terrible air quality conditions these days. Our software project is expected to respond those demands by develop an air quality monitoring system to collect air quality data with IoT devices and technologies and create our own web application to present air quality indicators.

1.2 Objectives

1. To develop the wireless weather monitoring station(s) for obtaining data of temperature, relative humidity, PM_{10} concentration, $PM_{2.5}$ concentration and GPS location of the station.

2. To develop a LoRa gateway which capable of receiving the data from all stations and send the data to the cloud server.

3. To develop a web application which capable of logging and storing the received data from the cloud server to our database server. In order that the web application can present the overview data through the frontend.

4. To enhance capability of our web application by adding its ability to analyze the weather data. We aim to analyze our collection of weather data which is time-series data by using Pandas.

1.3 Problem Description

Nowadays, there are several projects of weather station and there are quite a number of ways to develop a system. We believe that the current best way to develop wireless sensor network is to use LoRa technology for its long-range communication. We use LoRa communication for passing sensor value within our local wireless network. However, the sensor devices and the gateway devices have quite a number of choices to choose with different pros and cons. The gateway is responsible for collecting data from all weather stations in the network and passing the data to the cloud server through the internet. In this part, we use MQTT messaging technic protocol which is one of the best ways to communicate with IoT related things in order to transmit all parameters data from the gateway to the cloud server through the internet then the cloud server will transmit the data to our database server. As a software part, the web application with python language is quite powerful and flexible, so we aim to use Django Framework to build our web application with python language. A combination of above technology technics we used, it will be efficient and simpler way in creating an air quality monitoring system. From user experiences, using only website may not satisfy the need of usage, so we also develop mobile application to make our application compatible with mobile using Cordova. Lastly, the capability of weather prediction of our system will be developed using Pandas library for analyzing time-series data in our database, so we need the knowledge of machine learning to construct the efficient model to predict the most correct value with our weather data.

1.4 Scope of Work

The scope of this project can be listed as follows:

- To design the architecture of the weather station system from the gathered requirements.

- To build weather station(s) according to the design for reading the air quality data and GPS location of the station.

- To build gateway(s) according to the design for getting the data from the weather station and passing them to the server.

- To develop web application for storing, logging, and presenting the data from all weather stations using Django web framework and Grafana.

- To analyze the weather data on the web application for summarize the history weather data using Pandas library.

1.5 Overview Process of Project

In this topic, we will show the overview steps of process in our project.

1. Our weather stations get all the weather data and location from sensor devices.

2. Our weather stations send the data to our gateway via LoRa communication.

3. Our gateway receives the data from the weather stations and sends the data to the MQTT cloud server via Wi-Fi connection.

4. Our web server receives the data from The cloud server via MQTT protocol and collects the data to our PostgreSQL database.

5. In our application, the system shows overall air quality information of the nearest weather station and the location of the weather station on our dashboard with our own combination design of Grafana visualization.

6. The user can select a weather station to observe the weather information of that selected weather station.

7. The system will automatically update the data in real time.

1.6 Thesis Structure

In this topic, we will show the overview steps of process in our project.

- Chapter 1 Introduction – refers to the motivation, objectives, problem description, scope of work, overview process of project, and thesis structure of this thesis.

- Chapter 2 Related work – explain the literature review that relates to this project and compare with our project.

- Chapter 3 Background knowledge – explains the necessary knowledge and technology for the reader to understand the thesis.

- Chapter 4 System Architecture and Design – presents the requirement of the system, the use case diagram, relevant system architecture diagram and communication design.

- Chapter 5 Development – explains embedded hardware, software and techniques that are used in developing the project.

- Chapter 6 Results – refers to the results of hardware and software demonstration, which consists of the weather station, the gateway, and the user interfaces of the web application.

Chapter 2

Related Works

This chapter primarily focuses on evaluation existing projects, which are related to weather monitoring system or time-series data forecasting, and compare with our project.

2.1 Wireless Portable Microcontroller based Weather Monitoring Station

In this following work, we study a work of the research "Wireless Portable Microcontroller based Weather Monitoring Station" author by J.T. Devaraju, K.R. Suhas, H.K. Mohana and Vijaykumar A. Patil [4]. This research was published online in 2015.

2.1.1 System Design

This paper aimed to develop the Wireless Portable Weather Monitoring Station using PIC16F887 microcontroller equipped with several sensors including humidity, temperature, pressure, rainfall, solar radiation, wind speed, and wind direction. Further, Modbus communication protocols has been used to acquire data from the weather station to server over Zigbee wireless. At the server station, the received data from the weather station is stored online on a MYSQL server.

The PIC is related to Harvard architecture microcontrollers from Microchip Technology. PIC microcontrollers are popular because of their vast availability, extensive source of application documents.

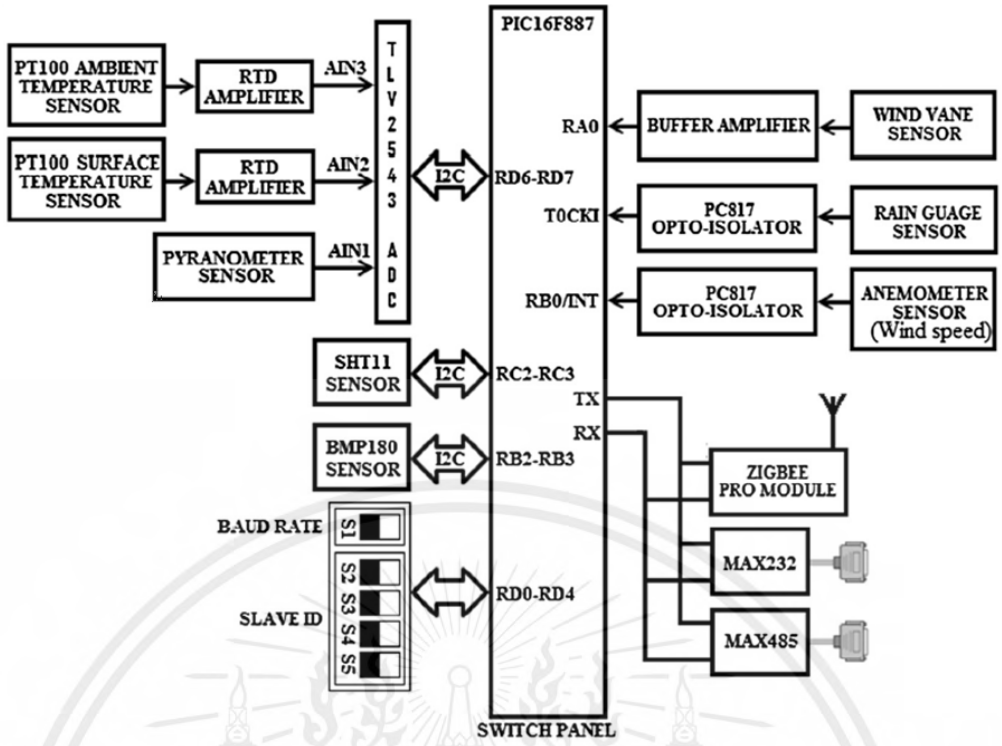

Figure 1: Block diagram of Wireless Portable Microcontroller

2.1.2 Communication Design

In this design, this research provides for both wired and wireless communication, but we only focus on wireless communication which is similar to our project communication. Wireless communication between the weather station and server station is provided by using XBee-Pro module with Zigbee protocol. The Xbee-Pro Module [5] is provided 256 kbps bandwidth over 2.4 Ghz frequency bands and has transmission range of 1 km as specified by manufacturer.

Figure 2: XBee-PRO module.

2.1.3 Software Architecture

The software consists of firmware and application software. The firmware is implemented with embedded C language. The Modbus slave protocol is implemented within firmware of the weather station to get data and communicate serially through Xbee Pro radio module. Modbus ActiveX Controller within Microsoft Excel is used for getting data from the weather station. If the station server is successful receiving the data, then it is displayed (Fig. 3). The logged data from Excel is uploaded to online MySQL server (Fig. 4) hosted by db4free.net using Visual Basic form application (Fig. 5).

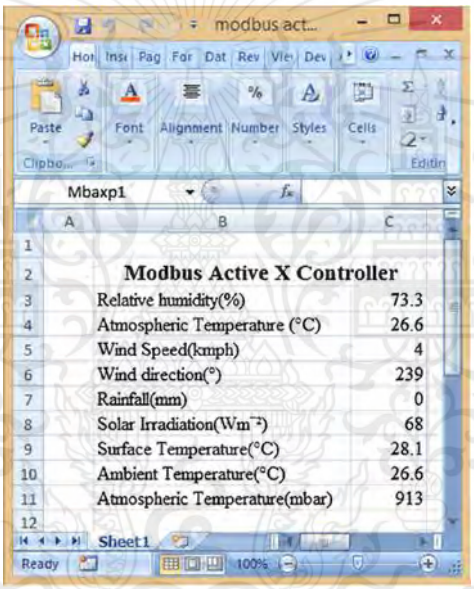

Figure 3: Modbus Active X controller displaying acquiring data

Figure 4: Online MYSQL Data server storing weather parameters.

2.1.4 Comparison between our project with this research

Table 1: Our Project compare with first research

Subject	Wireless Portable Microcontroller based Weather Monitoring Station	Our Project
Communication Protocol	Zigbee is used for communication to base station. It is the wireless protocol based on IEEE802.15.4 and has transmission range of 300 meters. It has main pros as connecting a number of devices in short-range.	LoRa is used for wireless communication between station and gateway. LoRa has pros as Zigbee in most ways except the range of transmission, it has transmission range up to 10 km which is far better than Zigbee. LoRa focuses on wide-area networks.
Processing Unit	PIC16F887 microcontroller is used to interfaces with Zigbee Pro module and all other sensors. PIC family is a microcontroller from Microchip which supports SPI, I2C and UART communication protocols. Programmed by embedded C language	LoPy 1.0 board with Espressif ESP32. It is MicroPython microcontroller with LoRa, Wi-Fi and Bluetooth. LoPy can set to be either end device or gateway of LoRa network. We use this board to set as end device and gateway in LoRa network.
Parameters Monitored	Humidity, Pressure, Rainfall, Temperature, Solar radiation, Wind speed, Wind direction	Temperature, Humidity, PM_{10}, $PM_{2.5}$, and GPS

Database	Online MySQL server	Local PostgreSQL
Monitor Interface	Online MySQL website (db4free.net)	Our own Web application and Mobile application
Analyze Data	None	Provided

2.2 Weather Station Design Using IoT Platform Based On Arduino Mega

In this work, we study a work of the research "Weather Station Design Using IoT Platform Based On Arduino Mega" author by Medilla Kusriyanto. Agusti Anggara Putra [6]. This research was published online in 2018.

In this research, it is mentioned that they try to make weather station that will be monitored through IoT platform website by sending the results to the IoT platform website via WIFI connection and also develop weather prediction by using the algorithm that uses the barometric sensor measurement or air pressure to estimate the weather.

2.2.1 Weather System

The weather system in this research is shown in Fig 4 . This system consists of Arduino Mega 2560 microcontroller as processing unit, DHT-22 as temperature and humidity sensor, BMP-180 as pressure sensor, FC-37 as rain sensor, ESP-8266 as WIFI module, DS-3231 as Real Time Clock module, and touch screen LCD

As it used WIFI module in this system, the architecture of this system is quite simple. The Arduino Mega 2560 will collect all the result data from all sensors and collect results to SD data storage card then sending the result to IoT platform website through WIFI connection directly. The website they used in this research is

‘Dweet.io’ as their website monitor and ‘Freeboard.io’ as their presenting dashboard visualization tool.

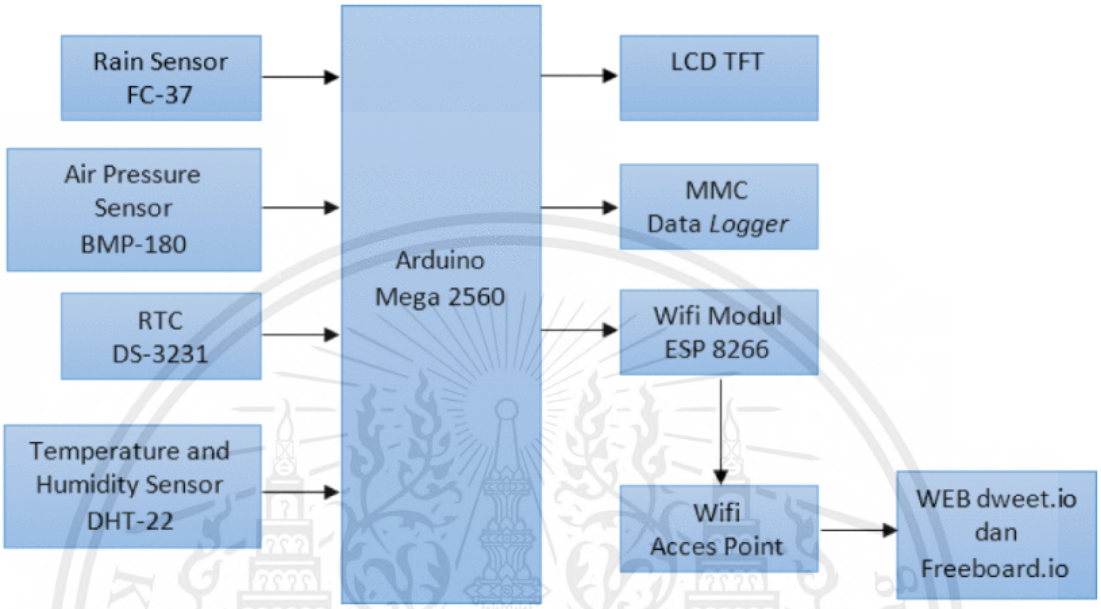

Figure 5: Weather System Block Diagram

2.2.2 Weather Forecast Algorithm

This research uses the algorithm from the research “Modern Altimeter and Barometer System using the MPL115A system” written by John B. Young. This algorithm uses the barometric measurement pressure to predict the weather. The approach of this algorithm is to see an increase or decrease in pressure. In simple, the increase in pressure over time is the inclination of the weather will be shiny. While the pressure decrease refers the weather will cloudy or rain.

Table 2: Pressure and Predicted weather

Analysis	Output
dP/dt > 0.25 kPa/h	Increased pressure fast, unstable.
0.05 kPa/h < dP/dt < 0.25 kPa/h	Increased pressure slowly, stable or good weather
-0.05 kPa/h < dP/dt < 0.05 kPa/h	Stable weather conditions
-0.25 kPa/h < dP/dt < -0.05 kPa/h	Pressure drop slowly, light rain / stable
dP/dt < -0.25 kPa/h	Fast pressure drop, storm or unstable

2.2.3 Comparison between our project with this research

Table 3: Our Project compare with second research

Subject	Wireless Portable Microcontroller based Weather Monitoring Station	Our Project
Communication Protocol	WIFI is used to communicate from weather station to the website monitoring. This way is easy to set up only one station for monitor the weather, but it is not for many stations.	LoRa is used for wireless communication between station and gateway. Our gateway use WIFI module to connect to the internet. Our way can easily add more station within the LoRa Range with one gateway that connect to the internet via WIFI

Processing Unit	Arduino Mega 2560 microcontroller as processing unit. Programmed by C language.	LoPy 1.0 board with Espressif ESP32 with LoRa, Wi-Fi and Bluetooth. We use this board to be both end node device and gateway station in LoRa network. Make it is very easy to develop with Python language.
Parameters Monitored	Temperature, Humidity, Pressure, and Rain	Temperature, Humidity, PM10, PM2.5, GPS.
Database	SD Card Storage	Local PostgreSQL
Monitor Interface	Dweet.io online website	Our own Web application and Mobile application
Weather Forecast	Calculate by barometric pressure algorithm.	Analyze time-series data statistic using Pandas library

Chapter 3
Background Knowledge

In this chapter, relevant background knowledge is required in order to get a better understanding techniques and technologies we are using in the project.

3.1 MicroPython

MicroPython is a tiny open source Python programming language interpreter that runs on very tiny embedded development boards. MicroPython allows you to write clean and simple Python code for controlling hardware instead of using complex low-level languages like C or C++. This simplicity of the Python programming language makes MicroPython a great choice for beginners who are new to programming and hardware [7]. The MicroPython language can implement small subset of the Python standard library, however, it is optimised to run on microcontrollers and in constrained environments. Python is known for having an extensive standard library, but trying to squeeze such a big library onto tiny boards with just kilobytes of memory is not possible. MicroPython instead implements smaller versions of some Python standard libraries to give you a great development experience.

MicroPython has some features that distinguish it from other embedded systems:

- Interactive REPL, or read-evaluate-print loop. This allows you to connect to a board and have it executed code without any need for compiling or uploading

- Extensive software library. MicroPython have batteries included and libraries built in to support many tasks.

- Extensibility. For the advanced developer, MicroPython is extensible with low-level C/C++ functions so you can mix high-level MicroPython code with faster low-level code.

3.1.1 Comparison of MicroPython and Arduino

Recently according to the IoT field, you will see a lot of products with Arduino more than MicroPython. What is the difference between them? There are two main differences between MicroPython and Arduino. The first is that Arduino is an ecosystem with the Arduino IDE, the Arduino programming language is based on C/C++ language. MicroPython is only a programming language interpreter and does not include an editor. The second difference is that the MicroPython language is interpreted instead of being compiled into code the CPU can run directly like with the Arduino programming language, means that when the MicroPython code runs it has to do a little more step to convert from the MicroPython code to the CPU understandable instructions.

An advantage of interpreted code is that it can be much simpler, and cleaner compared to languages that compile directly to CPU instructions. The developer can write and run interpreted MicroPython code outright on a board with Interactive REPL that we mentioned earlier. By the way, there somehow is a disadvantage of the interpreted code of the MicroPython languages that it performs less performance and sometimes it uses more memory when interpreting code which giving the developer some hard time in managing the memory. A function written in Arduino will run as fast as possible on a board's CPU, whereas similar code in MicroPython will be a little slower because it has to interpret every instruction and convert it to CPU code. Nevertheless, MicroPython will steadily grow and become more popular in the IoT field in the future. That is why we focus this project on the firmware that supports this language. [7]

3.2 LoRa

LoRa is a spread spectrum modulation technique derived from chirp spread spectrum (CSS) technology. Technically, it is a radio modulation scheme. A way of manipulating a radio wave to encode information using a chirped, multi-symbol format. LoRa also refers to the systems that support the modulation, including LoRa chips and gateways. [8]

Also, LoRa refers to the communication network for wireless LAN networks in the category of LPWA (Low Power Wide Area Network) network technologies that is suitable for IoT applications. LoRa, essentially, is a clever way to get very good receiver sensitivity and low bit error rate (BER) from inexpensive chips. That means low-data rate applications can get a much longer range using LoRa rather than using other comparably priced radio technologies.

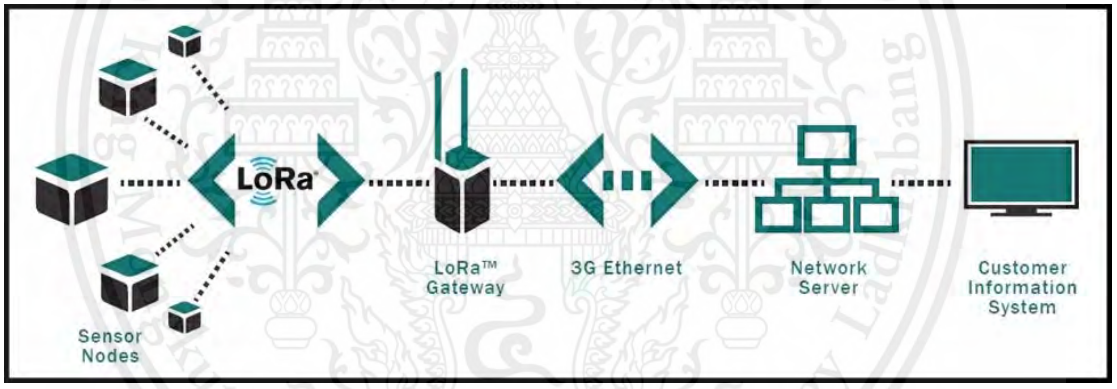

Figure 6: LoRa Network Structure

LoRa is very popular in industrial IoT systems (IIoT), which are suitable for building sensor nodes for industrial plants. Other communication technology options for IoT devices include WIFI, SigFox, BLE, NB-IOT, ZigBee, etc. They have different advantages and disadvantages [9].

The basic features that LoRa has its own advantage

- Long-range, Bidirectional Communication
- Secure, Bidirectional Communication
- Low-power, Long battery life
- Low cost
- Can be deployed to create your own Local Network (SigFox cannot do)

3.2.1 The Difference between LoRa and LoRaWAN

LoRa is a radio remote communication technology using the proprietary spread spectrum technique, which was developed by Semtech Corporation. LoRa is positioned in the Physical Layer of the communication layer which only supports peer-to-peer connections. LoRaWAN is a network communication protocol created using the capabilities of the physical layer of LoRa, which has a group of companies, called LoRa Alliance, sets the LoRaWAN Protocol standards. LoRaWAN can create a base station-based network.

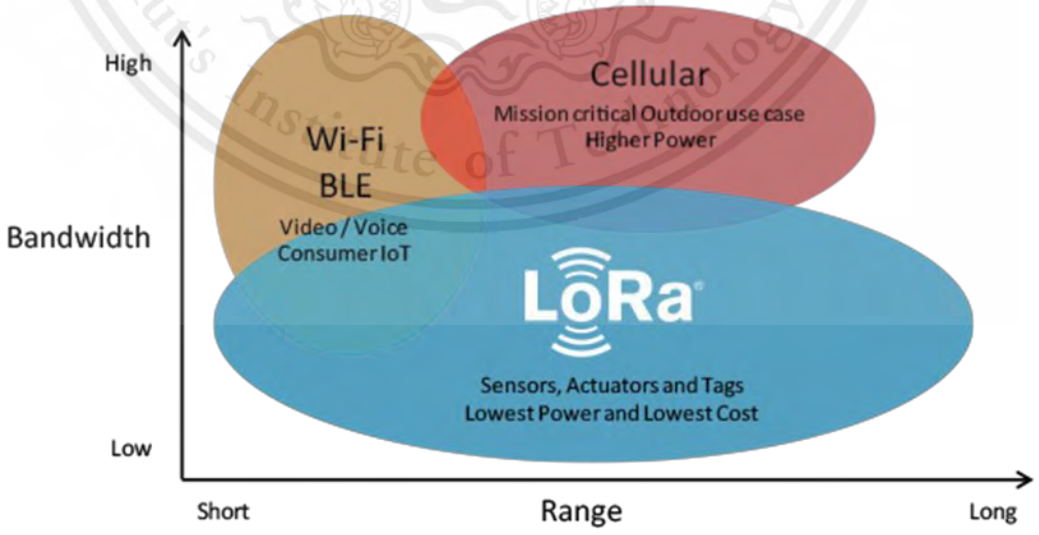

Figure 7: Bandwidth vs Communication Range of Each Technology

3.3 MQTT Protocol

MQTT, stand for **M**essage **Q**ueuing **T**elemetry **T**ransport, is a lightweight public-subscribe based messaging protocol that avoids direct connect between devices by transporting data through a central server called the broker and enables us to establish systems capable of publishing and receiving message. In the MQTT protocol, there are two types of network entities which are a message broker and several clients. The MQTT broker receives all messages from the clients and then transports them to the target clients. In order to perform the message transportation, the clients first need to connect to the broker over a network which is run over TCP/TP. This protocol is really popular in IoT because it uses low bandwidth and lightweight message communication which makes it suit to IoT devices. The lightweight and simple message that allows you to either read, or command the device very easily. Depend on the content of the message you are transferring, it allows you to either read data from the devices, or commands to control the devices. Due to the characteristics of this structure, it is very comfortable to add new devices without involving the existing infrastructure since new devices only have to communicate with the broker, so they do not need to be compatible with the other devices.

There are a few basic concepts about MQTT that clarify to completely understand this protocol which are broker, connection, publish-subscribe, and topics.

3.3.3 MQTT Broker

For MQTT Broker, it is an intermediary server that manages the queue and primarily responsible for receiving all messages in the queue, filtering the messages, transfers data between devices and decide who is subscribed in topics, and then publishing the message to all subscribed clients. For the publish and subscribe system, a device can publish a message on a topic, or it can be subscribed to a topic to receive messages. For example, device 1 publishes on a topic. device 2 is subscribed to the same topic as device 1 is publishing in. So, device 2 receives the message. For messages, they are the information that you want to exchange between your devices. All the information you want to transfer is in this message. It wraps in the format of one single string.

So JSON format may be involved in this part for separating parts in your message whether it is commands or data.

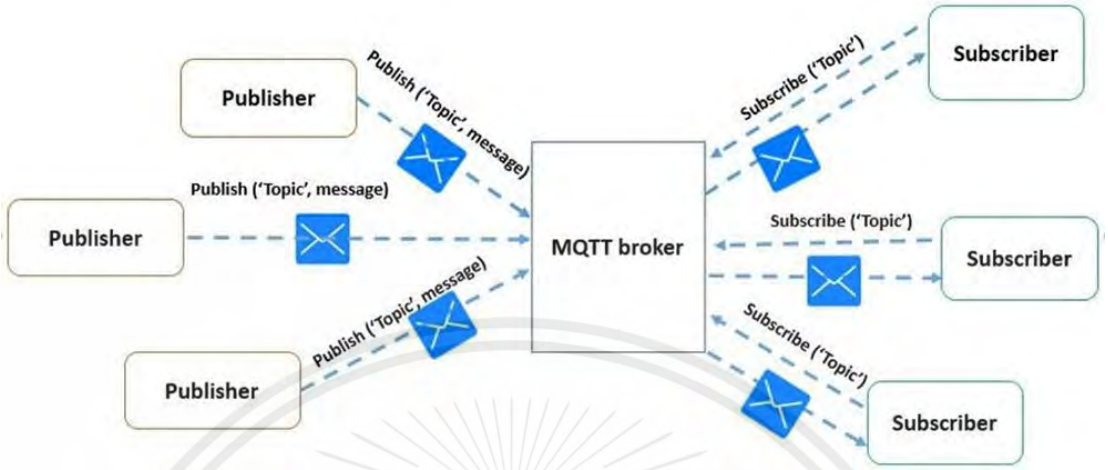

Figure 8: MQTT Broker System Diagram

3.3.1 MQTT Connection

The MQTT protocol runs on an application layer protocol using TCP/IP to connect to the broker. Once the clients connect to the broker over TCP/IP with the specified IP address or domain name and the port of the desired broker, the MQTT client library will establish system by creating socket channel in order to communicate between the clients and the broker server while they are connecting. This connection is normally established by the client so that the client can send message to the broker, as well as receive message from the broker at any time. MQTT client libraries are available for many essential programming languages such as Python, MicroPython, Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, and .NET [10].

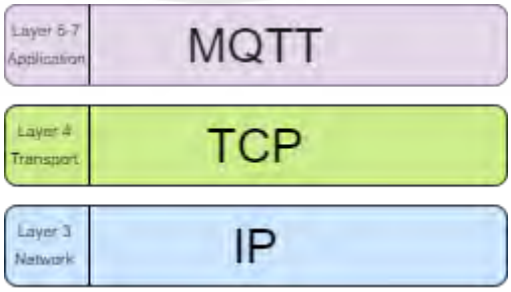

Figure 9: MQTT Network Layer

In this project, we use Python in the development of web application and MicroPython in the development of the microcontroller devices, so both languages have its own support MQTT client library for any developers who want to build the MQTT system for their project. The steps of how to initialize the client connection in both Python language and MicroPython language is being displayed in the following paragraph.

For Python language, we use the Paho library which is the open-source code provides by The Eclipse Foundation. The Eclipse Paho MQTT client library provides a client class which enable applications to connect to the broker. It also gives you some useful functions to make publishing messages to an MQTT server very straightforward [11]. These are example code of how to initialize the client connection to the MQTT broker with Paho library.

1. Import the MQTT client from Paho library.

```
import paho.mqtt.client as mqtt
```

2. Setup the broker configuration

```
DOMAIN_NAME = "www.example.com"
PORT = 1883
KEEP_ALIVE = 60
```

DOMAIN_NAME represents the broker host.

PORT represents the port at which broker is available.

KEEP_ALIVE represents time period in seconds that the broker will check whether the connect is still open and working or not. Set to 0 to make it disabled.

3. Define some necessary methods

```
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc))
    client.subscribe("AQMS/station1")

def on_message(client, userdata, msg):
    message = msg.payload.decode("utf-8")
    print("Received data: " + message)
```

“AQMS/station1” represents the topic that the client subscribes to the broker. We will explain about the topic later.

4. Set the client to listen to the broker channel

```
client = mqtt.Client()
client.connect(DOMAIN_NAME, PORT, KEEP_ALIVE)
client.on_connect = on_connect
client.on_message = on_message
client.loop_forever()
```

For the MicroPython language, we use the uMQTT library which is a simple MQTT client library for microcontroller. This library is free software under the "FreeBSD" license. It provides a set of functions for processing MQTT client packets that are intended basically for use with microcontroller [12]. These are example code of how to initialize the connection with the uMQTT library which is somehow similar to the above example code.

1. Import the MQTT client from uMQTT library.

```
from umqtt import MQTTClient
```

2. Setup the broker configuration

```
ID = "Sakkarin"
DOMAIN_NAME = "www.example.com"
PORT = 1883
```

3. Set the client to listen to the broker channel

```
client = MQTTClient(client_id=ID , server=DOMAIN_NAME, port=PORT)
client.connect()
```

3.3.2 MQTT Publish-Subscribe and Topic

MQTT is a public-subscribe based protocol that provides a traditional client-server model. The publisher is the client that sends the message, and the subscriber is the client that receives the messages. The publishers and subscribers never interact with each other directly and are not knowing the other exists. The connection between them is managed by the broker. When the clients subscribe or publish through the MQTT broker, they communicate with the specified channel which is identified by the name of the topic. So, the topic is the way to identify the name of the channel you

interest to receive incoming messages or where you want to publish the message for those subscribers waiting for messages. Topics are represented with strings separated by a forward slash. Each forward slash indicates a topic level. There is an example of how you would create a topic for our air quality monitoring system.

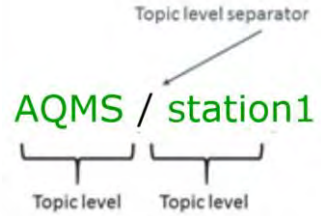

Figure 10: MQTT Topic

The hash symbol represents the multi-level wildcard in the topic, indicates as '#', which means that it covers all the topic levels that after the '#' symbol. For subscribing, the server will acknowledge that the client has subscribed to all the topics that exist after the '#' symbol level. The same for publishing, the client will transport the message to the clients of all the topic channels that exit to all subscribers. For the broker to specify which topics match, the multi-level wildcard must be set as the last character in the topic and preceded by a forward slash.

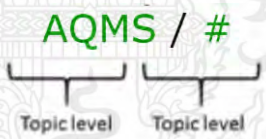

Figure 11: MQTT Topic with Hash Symbol

Since you understand how the idea of the topic works, you can subscribe and publish the broker channel with the name of the topic that you desired. These are the example of how you subscribe and publish to the broker with the Python language. Note that these methods are supposed to be called after you connect to the broker.

```
MQTT_Topic = "AQMS/station1"
Msg = "Message send from station1 client"
client.subscribe(MQTT_Topic)
client.publish(MQTT_Topic, Msg)
```

3.4 Django Framework

Django framework is a high-level Python web framework that good for the rapid development of secure and maintainable websites. A framework is a collection of modules that make development easier. Django comes with many ready-to-use libraries which make it simple to develop a perfect website with ease. The server takes care of much of the inconvenience of the web, so you can focus on developing your website without having to reinvent the tools. The most important is that it is free and open source.

Django framework follows the idea of MVC design pattern, but it has its software design called MVT design pattern which M stand for Model, V stands for View, and T stand for Template. The MVT design pattern of Django typically group the code that manage each steps into different files as the figure shown below.

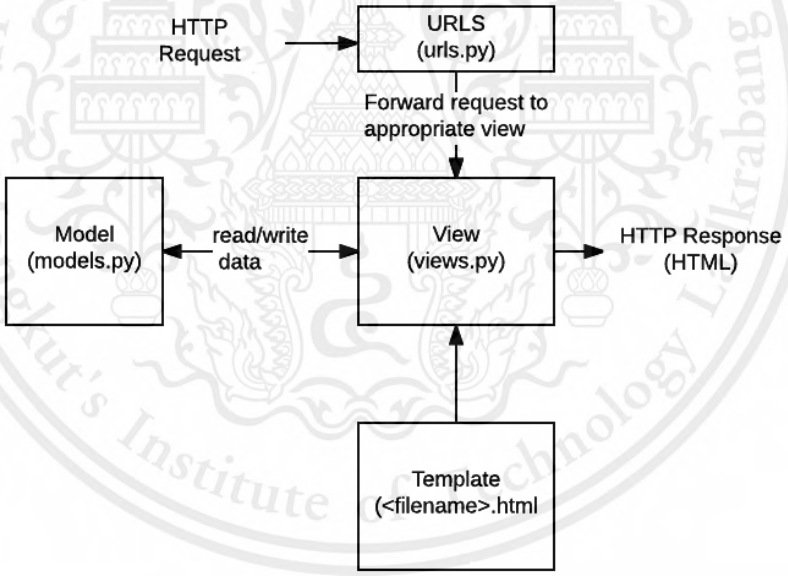

Figure 12: Django MVT Design Pattern

In next sections, we will explain a concept of how these main parts of a Django application work in more detail.

3.4.1 Django URLs

A URL matcher is generally stored in a file named urls.py. The urlpatterns is the list of matching between URL routes. When the server receives an HTTP Request that matches a URL in the urlpatterns, then Django will call the view function and passed the request. In the example below, the urlpatterns defines a list of matchings between routes and the Python view functions.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name='register'),
    path('', include('weatherstation.urls')),
]
```

3.4.2 Model View Template (MVT)

Django framework follows the MVT design pattern (Model, View, Template) as its software design pattern. Model(M) manages everything involving the data in the backend field. For instance, adding data, accessing data, obtaining data, updating data, and deleting data. In short, models are Python objects that define the relationship of the data and query data in the database. View(V) is a request handler function, which receives HTTP requests and returns responses. It responsible of accessing the data needed to satisfy requests via models, and representing the formatting of the response to templates. So, views are the main part responsible for accessing databases, rendering templates, etc. Template(T) is the presentation layer containing text file such as an HTML page. A view can use an HTML template to dynamically create an HTML page, forming it with data from a model. Django allows the developer to put some unique code, called Django Template Language, inside the HTML template to be able to do some logic business that suit the requirement of the website. However, template principal is just to display the data context, so you need to be careful not putting too much code in it. These are examples of how you implement Django in MVT design software patterns.

Models.py holds data structures which can be called the Django data model.

```
from django.db import models

class Stations(models.Model):
    station_id = models.AutoField(primary_key=True)
    description = models.TextField()

    def __str__(self):
        return str(self.station_id)

    class Meta:
        ordering = ['station_id']
        verbose_name = 'Station'
       verbose_name_plural = 'Stations'
```

Views.py holds the Python logic that accesses the data from models.py , processes it, then sends it to the HTML files in the templates directory.

```
from django.shortcuts import render
from .models import Stations, StationLogs
from django.utils import timezone
import random, requests as req, json

def index(request):
    context = {
                'stationlogs': StationLogs.objects.all()
               }

    return render(request, 'weatherstation/index.html', context)


def logs(request):
    context = {
                'stationlogs': StationLogs.objects.all()
               }


    return render(request, 'weatherstation/logs.html', context)
```

Templates are HTML files with the capability to embed Python variables and code which called Django Template Language.

```
{% regroup stationlogs|dictsort:"station_id"
                        by station_id as station_list %}
{% for station in station_list %}
{% if station.grouper == 1 %}
{% for log in station.list %}

<div class="col-lg-3 col-12">
  <div class="info-box">
    <div class="info-box-content">
      <h4><strong>Temperature</strong></h4>
        <span>{{ log.station_temperature|floatformat:1 }} °C</span>
    </div>
  </div>
</div>

{% endfor %}
{% endif %}
{% endfor %}
```

3.5 Pandas

Pandas is a Python software library written for data analysis. It is an open-source Python library under a BSD license, and it was written by Wes McKinney. The name is derived from the term ‘panel data’. Pandas is basically written from Numpy library in order to serve its purpose which is providing data structures and operations for manipulating tables and especially dealing with time series data. Pandas is one of the most popular tool for analyzing data with Python because it can take data like CSV and create Python object called data frame which makes it easier to work with lists, dictionaries, or time-series data and further use it for analyzing data.

Figure 13: Logo of Pandas Library

3.6 Grafana

Grafana is a visualization tool for presenting metrics data from your connected data source. It provides charts, graphs, and alerts for the web with customizable dashboard. It is a very popular visualization tool because it is open source, support several data source, excellent feature to work with time-series data. It is expandable through a plug-in system that developer can install more plug-in to customize more complex presenting dashboard using interactive query builders. Grafana can interface with several popular data sources such as Graphite, Prometheus, Influx DB, ElasticSearch, MySQL, PostgreSQL, etc. Grafana has the tool called time series analytics, it helps us study, analyze, and monitor data over a period of time. It can also track the user behavior, frequency of errors, type of errors by providing relative data. Grafana can be embedded to any website with iframe if you do not desire to get the whole Grafana dashboard on your website.

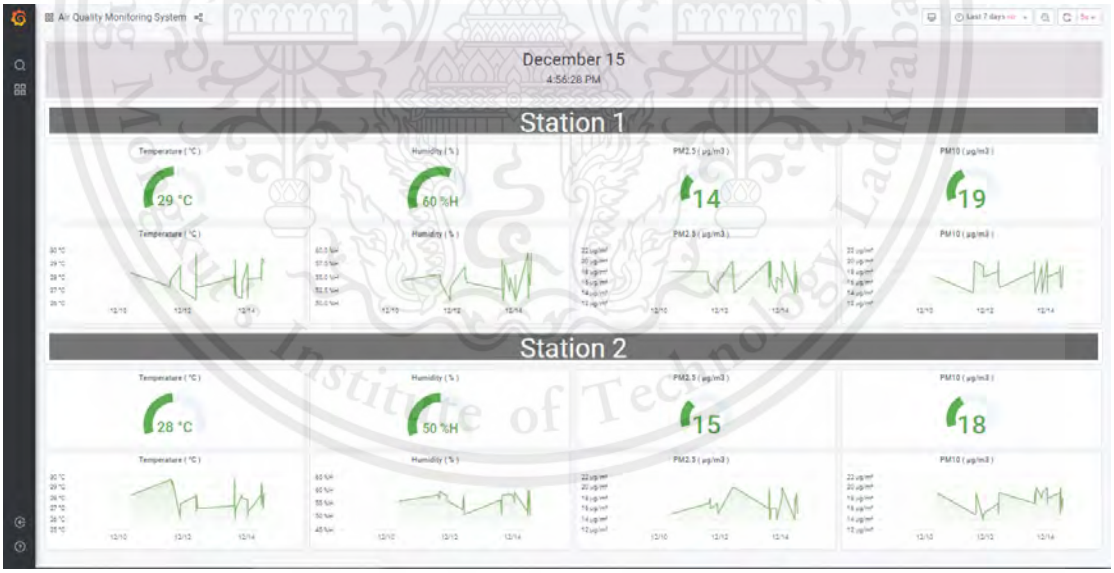

Figure 14: Example of Grafana Dashboard

Chapter 4
Software Architecture and Design

4.1 Requirement Analysis

4.1.1 Functional Requirement

- The system provides users authentication system including login, logout, and registration system.
- The system allows users to edit their account information.
- The system can display the dashboard of all air quality information both on web application and mobile application
- The system allows the user to view the dashboard of air quality information on web application
- The system can display the lasted update time of the data on dashboard.
- The system can display Google Maps interface on the dashboard.
- The system can display all weather stations location in Google Maps interface.
- The system will automatically display the nearest weather station's air quality information on dashboard by default.
- The system allows users to select any weather stations to view that weather station's air quality information on dashboard.

- The system can display air quality information history in graphical charts on dashboard.

- The system can summarize the weather statistic using Pandas analytics.

- Weather station can read temperature, humidity, PM_{10} concentration, $PM_{2.5}$ concentration and GPS location from its sensors.

- Weather station can send all the data to the gateway via LoRa network

- The Gateway can connect to the internet via WIFI module.

- The Gateway can receive data from all weather stations via LoRa network and forward the data to the cloud with MQTT protocol.

- The system can receive the data from the cloud through MQTT protocol

- The system can represent Grafana graph on our dashboard through iframe.

4.1.2 Non-Functional Requirement

- The Graphic User Interface of dashboard is easy to understand and user-friendly.

- The system shall be a web application that run on web browser.

- The system shall be a mobile application that run on any devices.

- A web application has a responsive design for supporting all platform.

- The system shall display the last updated time of the sensor information.

- The system provides near-real time data with an update frequency of 5 seconds

- The system shall allow for users to access weather information of the selected weather station within almost two mouse clicks.

- The system complies to radio frequency system usage laws in Thailand.

- The system uses LoRa technology for wireless data transmission from weather station to the gateway.

- The system uses MQTT messaging protocol technology for transmitting data between gateway, cloud server and web server.

- The system uses PostgreSQL relational database to support a large amount of sensored data.

- The system uses Django web framework to support the Model-View-Template architectural design pattern (MVT) for developing a web application.

- Weather station can be mobilized with ease by our engineer due to our WSN (Wireless Sensor Network) concept.

4.2 Use Case Diagram

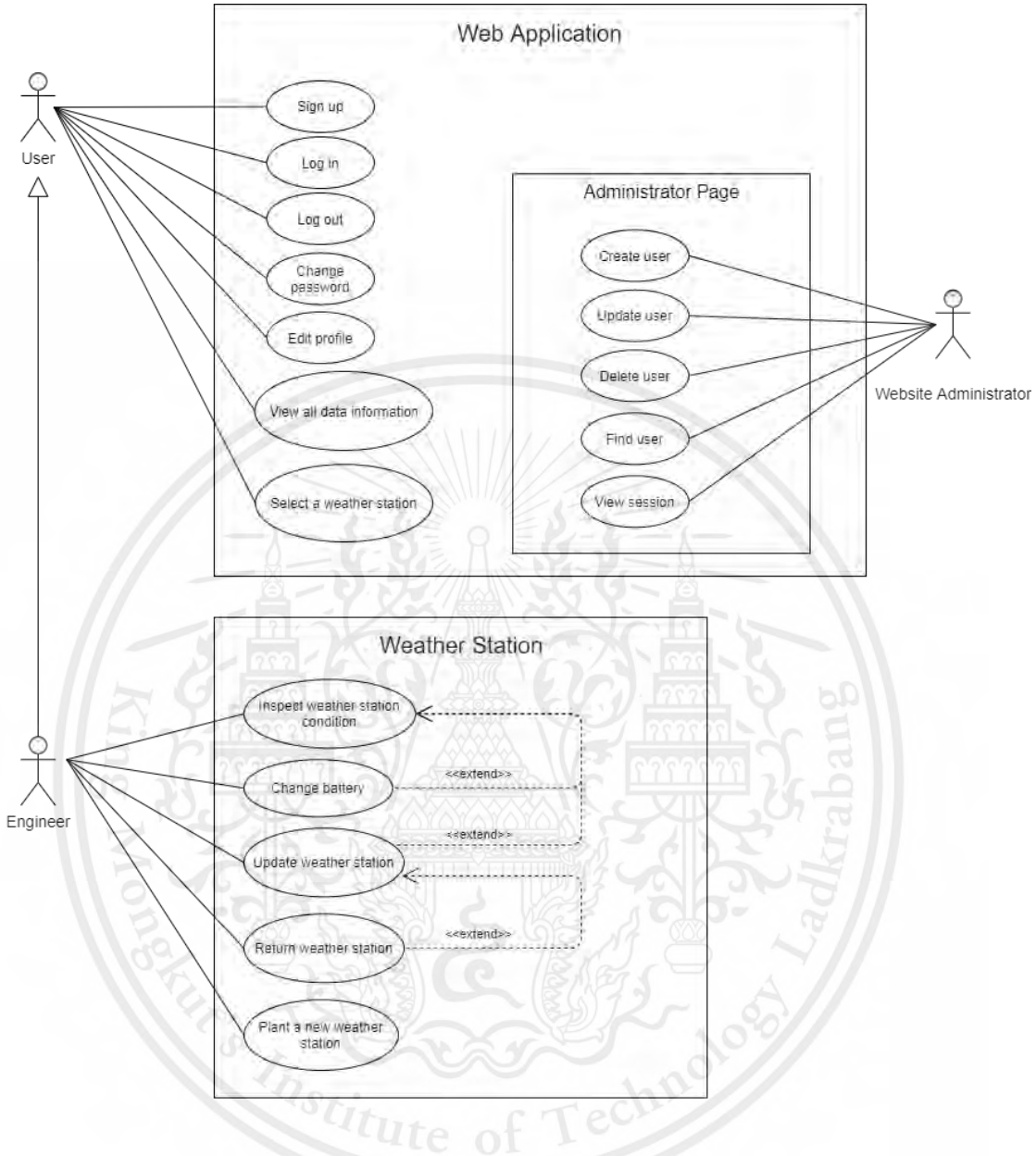

Figure 15: Use Case for Air Quality Monitoring System

For a complete list of all use case description please see Appendix A.

4.3 System Architecture

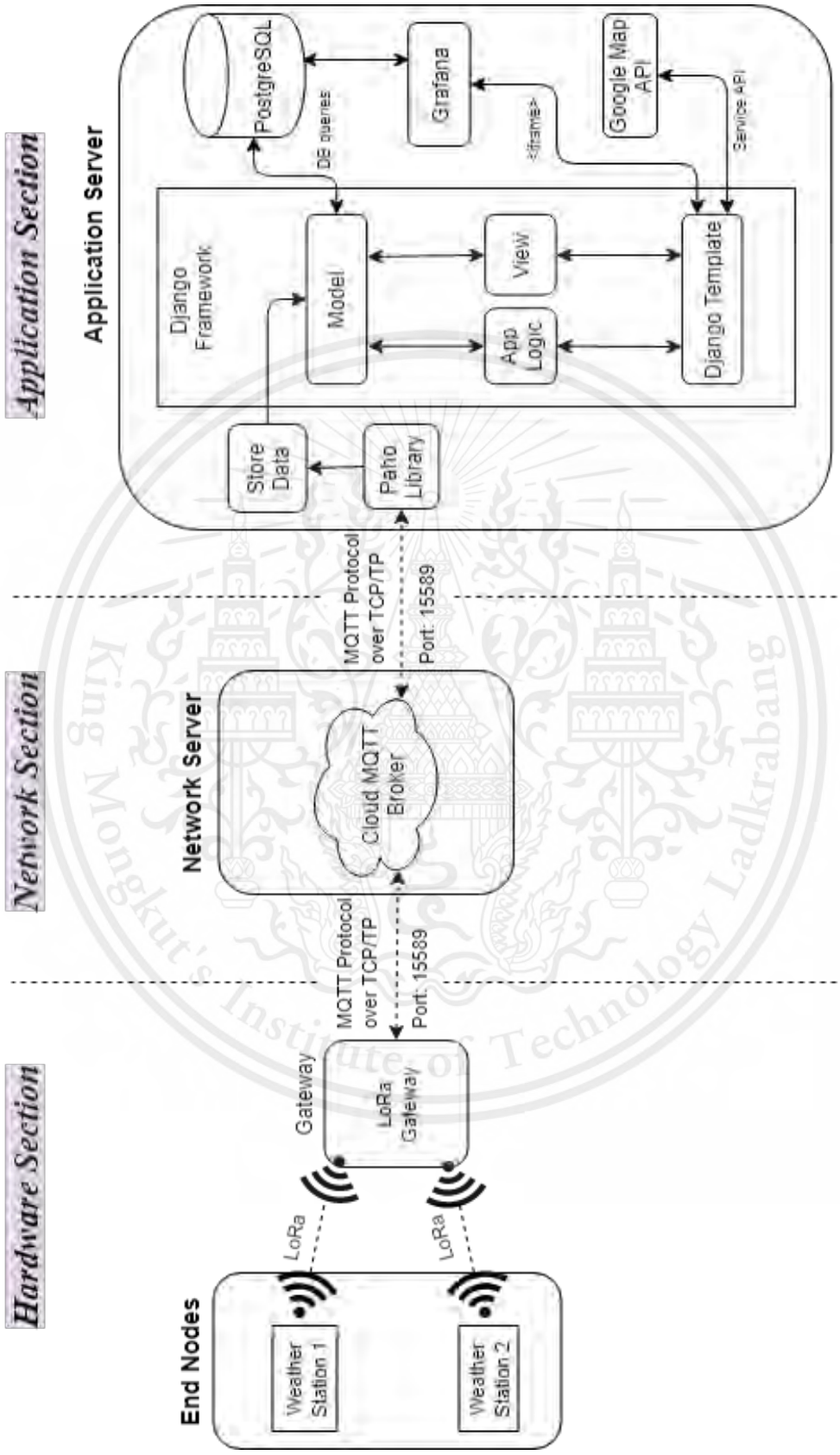

Figure 16: Overall System Architecture

There are 3 main sections of our architecture system, which are the hardware section, network section, and application section.

The hardware section consists of two part of devices which are the end node and the gateway. An end node part is a group of each weather station that is responsible for reading all sensory data and sending those data directly to the gateway via LoRa communication. The gateway part is a LoRa gateway station that receives all messages that those weather stations in the end node part are sending through LoRa protocol. The gateway also has the responsibility for sending those received messages to the MQTT cloud server with MQTT protocol which requires the gateway station to connect to the internet with Wi-Fi Module.

The Network section is an intermediate connection between the gateway station and the application server. MQTT protocol is used for getting and sending messages from the gateway to the application server. We use the CloudMQTT, a free distributed MQTT cloud server, to be our MQTT broker server.

Last section, the application section, our application will connect to CloudMQTT broker server with over TCP/IP port 15589 for receiving the desired message when the gateway sends the message to the broker. The Paho library is the Python MQTT client library that connects our server to the MQTT broker server and provides functions that manage the MQTT protocol for us like receive the message from the gateway in the specific channel called topic. After that, we will pass the received data we get from the Paho function model module in the Django server, then the backend server will store the value from that message into our PostgreSQL database server. Thus, the user can observe the overall air quality information that queries from our database in our representative web-based frontend with the visualization tool called Grafana. Grafana server will connect to our PostgreSQL database server apart from the model module of the Django framework, then our frontend web-based will access the tools of Grafana module by using iframe html tag in order to import the visualization graph into our own dashboard.

4.4 Django Data Model

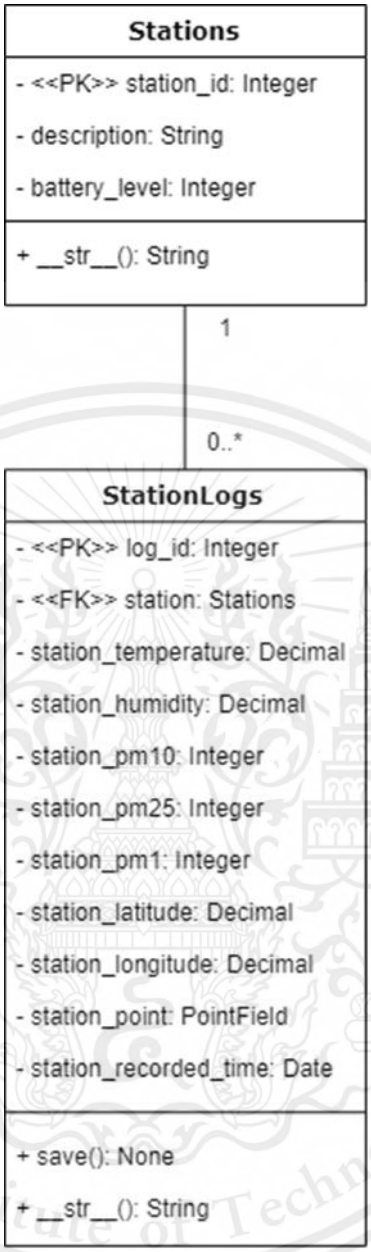

Figure 17: Django Data Model

The class diagram is the Django data model in Django framework that shows the relationship among the model in our server. We design our classes based on our desired database in the server. Due to the MVT design pattern of Django framework helps us to create our database based on our classes in 'model.py', we just have to design our classes according to our needed database design and let Django do their

work in implementing our database tables and relationship between entities in the database for us.

4.5 Communication Design

As we shown in system architecture diagram, our system uses LoRa protocol for communication between the weather station and the gateway and uses MQTT protocol for transporting the data message from the gateway to our server.

For LoRa protocol, our gateway device will be the gateway of the LoRa system, and all of our weather stations will be end devices of the LoRa system. The weather station will be set to send the data to the gateway every 15 minutes. The gateway will respond to weather station every time it received the data by sending the data forward the cloud server.

Before the gateway send the data to the cloud, the gateway has to connect to the internet in order to connect to the cloud server. As we use LoPy for our gateway, we use its built-in Wi-Fi module to connect to our provided Wi-Fi. Then we connect to the server and send the data via MQTT. We design our system how to publish/subscribe in MQTT protocol as a following figure.

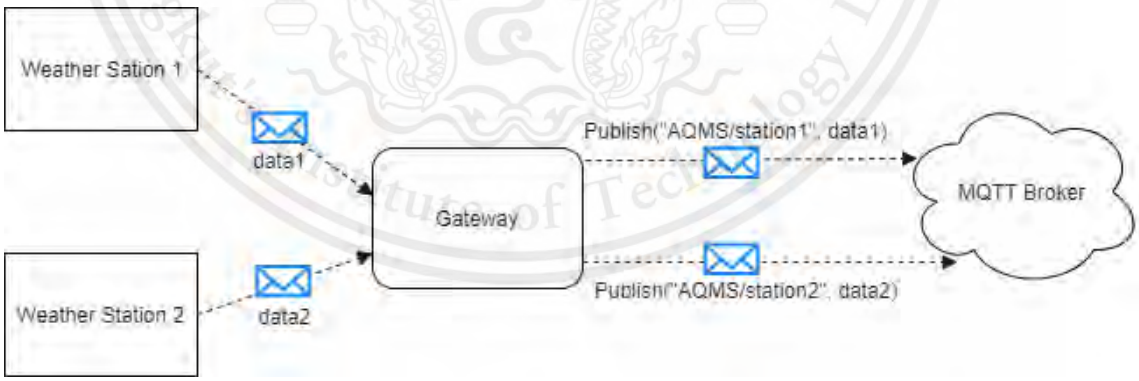

Figure 18: Design of Publishing in Hardware Part

For MQTT protocol, the topic for our system is “AQMS/” which stand for air quality monitoring system. For each station, the topic character after forward slash will be “station” follow by ID number of the station. For example, weather station 1

will publish the message as the topic “AQMS/station1”, station 2 will publish the message as the topic “AQMS/station2”.

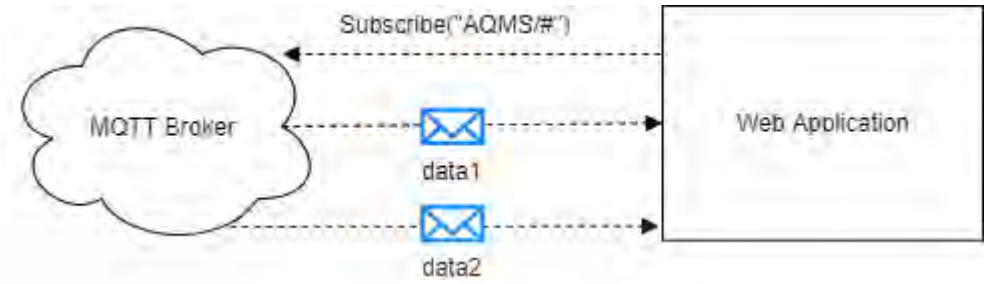

Figure 19: Design of Subscribing in Server Part

For the web server, our server will subscribe to the MQTT broker with the topic “AQMS/#” for receiving all the messages from all weather stations. The design of how our web server subscribe to the MQTT broker is shown in a above figure.

4.4 Sensor Dataset

We design a structure of the message in JSON format in order to gather all sensory data from the station and passing them to the MQTT cloud server. This way of structure will benefit with MQTT protocol which allows you to send a single string for each transmission. It makes us easier to extract the needed value from the message from the MQTT.

Station_ID	Temperature	Humidity	$PM_{2.5}$	PM_{10}	Latitude	Longitude

Figure 20: JSON Structure of Dataset

Table 4: Structure of Sensor Dataset

Field	Description
station_id	Number ID of the station
temperature	Temperature value
humidity	Relative Humidity value
pm25	Level of PM2.5
pm10	Level of PM10.0
latitude	Latitude of the station's location
longitude	Longitude of the station's location

Chapter 5
Development

This chapter describes the development process of our project. For the hardware section, we will particularize the hardware devices we used in our project. Then, we explain and communication protocols for data transmission between our weather station, gateway, the MQTT cloud server, and the application server. Finally, we show the sensor interface of our embedded and the development tools require to implement the web application.

5.1. Hardware Devices

For hardware devices, we use LoPy as the main microcontroller connected with Pycom Expansion Board 2.0 to utilize all extra pins to connect with other sensor board. We use Temp&Hum 2 Click Board to read temperature value and humidity value. We use GSM/GNSS Click Board for tracking the location of the station. We use the HPMA115S0 air quality sensor for reading $PM_{2.5}$ and PM_{10} concentrations. Next, we will introduce you to all the device information and specification that we are using for our station and gateway.

5.1.1 LoPy 1.0

LoPy is the only triple bearer MicroPython enabled microcontroller created by Pycom company with LoRa, Wi-Fi, and BLE. So, this board is the perfect IoT platform for your connected things. Combined with the latest Espressif chipset the LoPy offers a perfect combination of power, friendliness, and flexibility. LoPy can also set up as both LoRa end device and LoRa gateway.

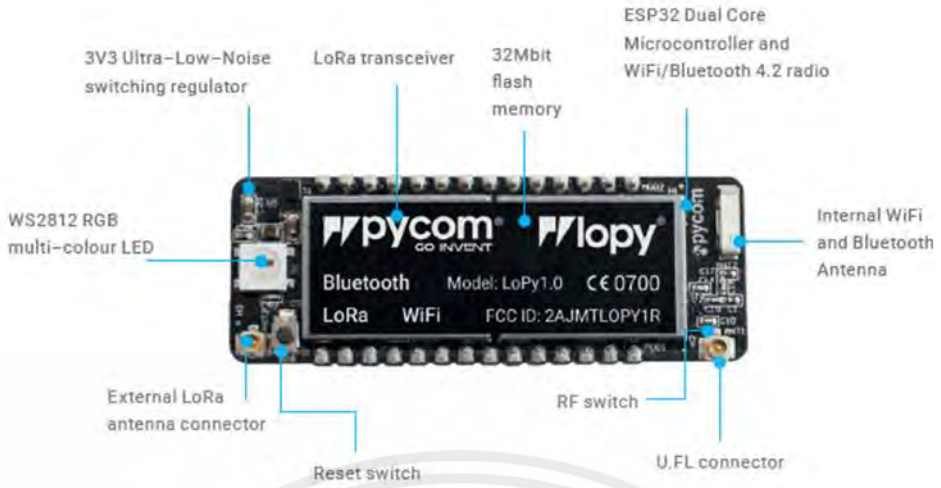

Figure 21: LoPy 1.0 Board

Specifications

- CPU – Xtensa® dual–core 32–bit LX6 microprocessor(s), up to 600 DMIPS
- Microcontroller System – Espressif ESP32
- Memory – RAM: 512KB / External flash: 4MB
- WiFi – 802.11b/g/n 16mbps
- Bluetooth – Low energy and classic

5.1.2 Pycom Expansion Board 2.0

The Pycom expansion board 2.0 is compatible with the WiPy 2.0, LoPy, SiPy, and FiPy and brings additional functionality. This kit contains a micro-USB connector for power and serial communications, a lithium polymer battery charger, reverse battery protection, a micro-SD card slot, a button, and a user LED. It has a JST style battery connector. The Expansion Board features a single-cell Li-Ion/Li-Po charger. When the board is being connected via the micro-USB connector, the expansion board will charge the battery. Our project uses this expansion board 2.0 for our LoPy board by simply insert the LoPy into the headers to access all the extra features.

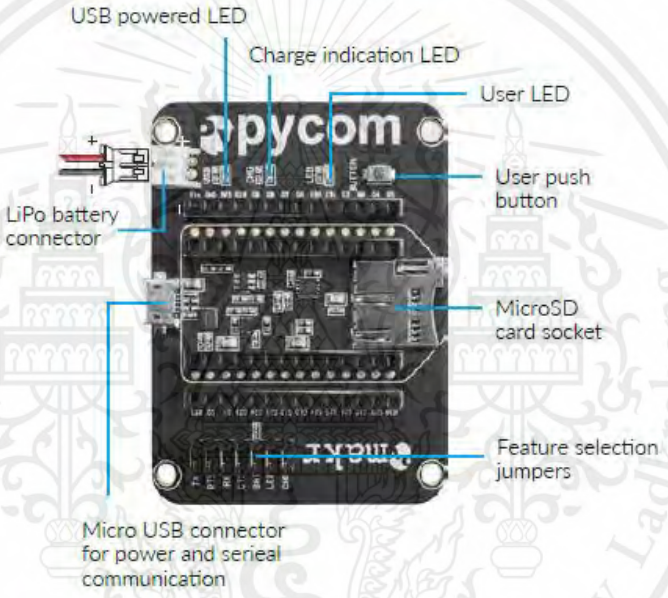

Figure 22: Pycom Expansion Board 2.0

5.1.3 Temp&Hum 2 Click

Temp&Hum 2 click is a smart temperature and humidity sensor click board, packed with features, that allow easy and simple integration into any design that requires accurate and reliable humidity and temperature measurements. It measures a wide range of temperature and relative humidity values with great accuracy.

The sensor IC used on the Temp&Hum 2 click is the Si7034, a digital humidity and temperature sensor IC with an I^2C interface, from Silicon Labs. It is a digital relative humidity and a temperature sensor that integrates temperature and humidity sensor elements, an analog-to-digital converter, signal processing, calibration, and data correction on the chip. Each chip is factory calibrated, and the calibration data is stored in its non-volatile memory.

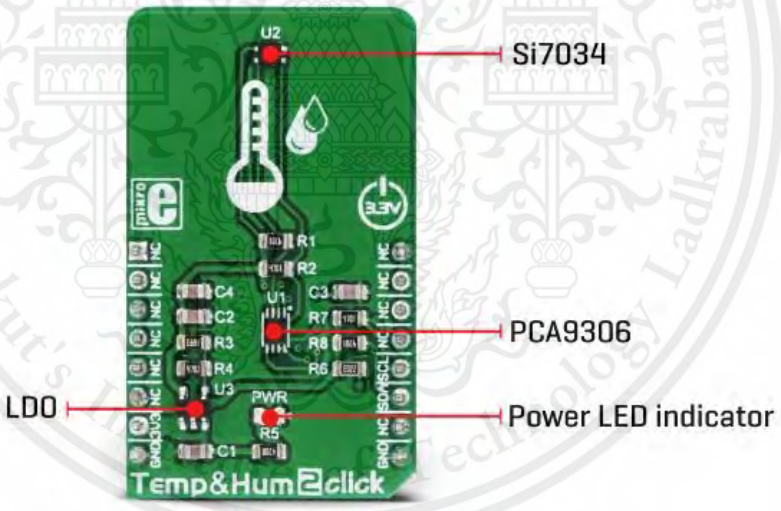

Figure 23: Temp&Hum 2 Click Board

5.1.4 GSM/GNSS Click

GSM/GNSS click board combines GPS and GLONASS(**GLO**bal **N**avigation **S**atellite **S**ystem) location tracking with GSM module capability for mobile communication and UART interface. When connected to a GPS antenna, it can receive GPS coordinates, time, and other information from orbiting satellites. The click can be used for all GSM functions which are calls, messages (SMS, MMS), and mobile internet. MC60 is a quad-band full-featured GSM/GPRS module using the LCC castellation package. With an extensive set of internet protocols, it has integrated GNSS technology for satellite navigation. The module can balance between positioning accuracy and power consumption according to the environmental and motion conditions. The typical power consumption is around 2.8mA.

Figure 24: GSM/GNSS Click Board

5.1.5 HPMA115S0

HPMA115S0 is a laser-based sensor that detects and counts particles using light scattering produce by the Honeywell. It uses a laser light technique to light at a particle from the air pulled from the detection chamber. The light detector will get the data from the recorded photo then converted it to a signal to calculate particle concentration.

Figure 25: HPMA115S0 (Air Quality Sensor)

It has many features as follow.

- Laser-based light sensing
- Concentration range: 0 µg/m3 to 1,000 µg/m3
- Response time: < 6s
- PM2.5, PM10 output
- Long life of 10 years of continuous use

5.2 Communication Development

In the hardware part, we use LoRa technology to communicate with the station and the gateway. We are developing our own local LoRa network to transmit the data. Here is the diagram of the weather station and gateway in more detail.

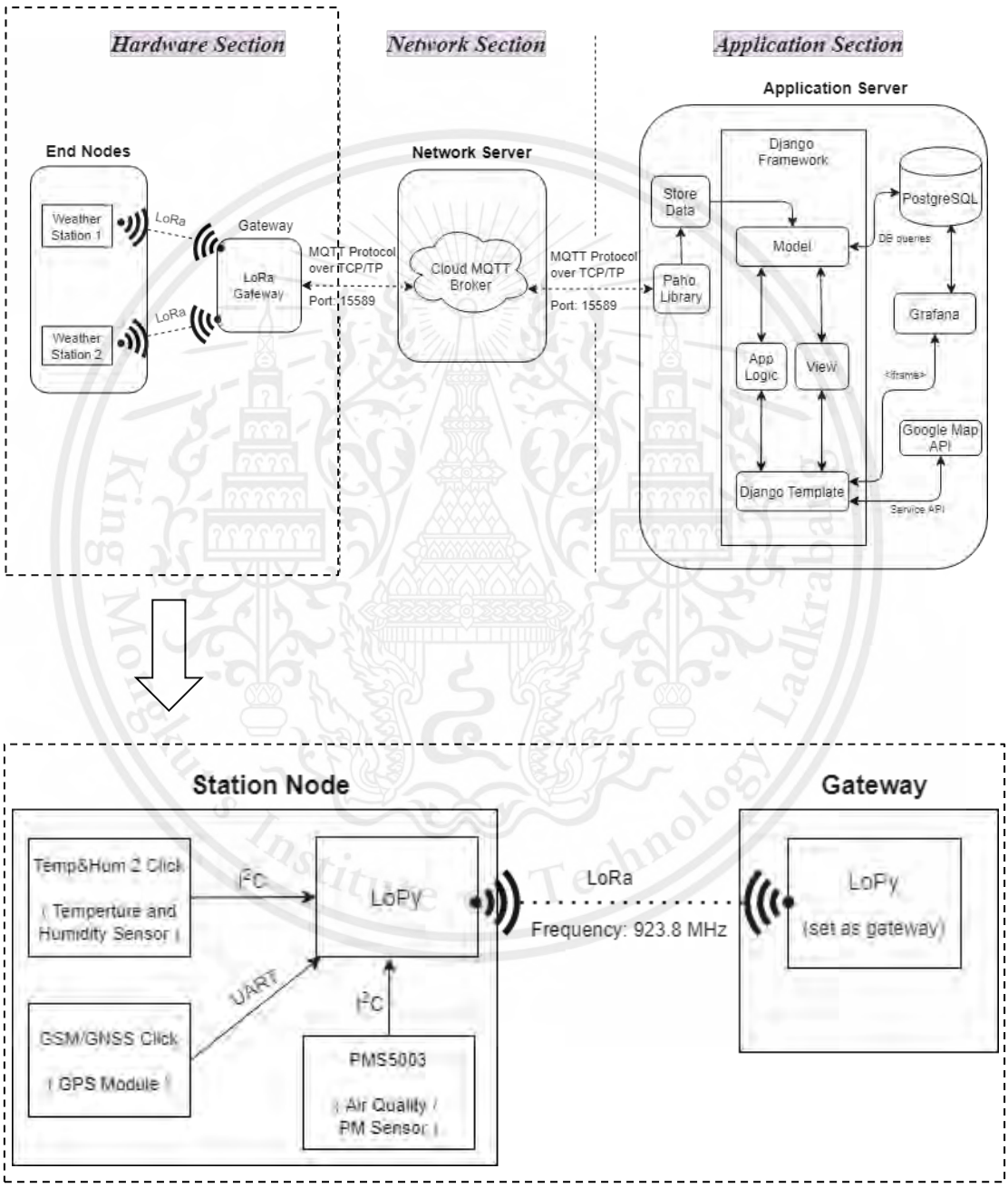

Figure 26: Hardware Network Diagram

For weather station, LoPy is the main microcontroller connected with all other sensor devices for reading air quality data and location. Temp&Hum 2 Click is used for reading temperature and humidity data using I2C serial protocol to connect to LoPy board. GSM/GNSS Click is able to function both mobile communication and location tracking, but we only use GPS module of this board for location tracking connecting with LoPy using UART. For air quality sensor, we use HPMA115S0 sensor to read air quality such as $PM_{2.5}$ air quality data. As we known that LoPy board can set to be both node and gateway of the LoRa network, so we set LoPy in weather station to be sensor node. For our Gateway, we use only LoPy board that set to be gateway of the LoRa network to communicate with all weather stations using LoRa. This is our implementation of the message structure of the sensory data in station.

```
while True:
    # Package send containing a simple string
    msg = ""
    msg = msg + "station_id:{}".format(station_id) + ","
    msg = msg + "temperature:{}".format(temperature) + ","
    msg = msg + "humidity:{}".format(humidity) + ","
    msg = msg + "pm25:{}".format(pm25) + ","
    msg = msg + "pm10:{}".format(pm10) + ","
    msg = msg + "latitude:{}".format(latitude) + ","
    msg = msg + "longitude:{}".format(longitude) + ","

    print(msg)

    pkg = struct.pack( _LORA_PKG_FORMAT % len(msg), DEVICE_ID,
                       len(msg), msg )
    lock.acquire()
    lora_sock.send(pkg)
    lock.release()
```

Station_ID	Temperature	Humidity	$PM_{2.5}$	PM_{10}	Latitude	Longitude

After the message was constructed, we send the message to the gateway via LoRa communication protocol. Then the gateway will receive a LoRa package from station by the following implementation.

```
while True:
    lock.acquire()
    recv_pkg = lora_sock.recv(512)
    lock.release()
    if (len(recv_pkg) > 2):
    # Extract information from lora message
    recv_pkg_len = recv_pkg[1]
    device_id,pkg_len,msg = struct.unpack(_LORA_PKG_FORMAT %
                                          recv_pkg_len, recv_pkg)
    msg = msg.decode('utf-8')

    # Check whoose package was? By using device ID
    if device_id == 0x01:
        lora_dictionary_update(lora_message=msg,lora_dict=dict_node1)
     elif device_id == 0x02:
        lora_dictionary_update(lora_message=msg,lora_dict=dict_node2)
```

Finally, the gateway will connect to the MQTT server and publish the data to the MQTT server

```
# MQTT Initialize
# CloudMQTT Broker
MQTT_BROKER_URL = "postman.cloudmqtt.com"
MQTT_BROKER_PORT = 15589
MQTT_BROKER_USER = "nswnnfoc"
MQTT_BROKER_PWD = "HbdSr-1z9oYG"
client = MQTTClient(client_id=MQTT_LORAGATEWAY_ID,
                    server=MQTT_BROKER_URL, port=MQTT_BROKER_PORT,
                    ssl=False, user=MQTT_BROKER_USER,
                    password=MQTT_BROKER_PWD, keepalive=60)
client.set_callback(sub_cb)
client.connect()
```

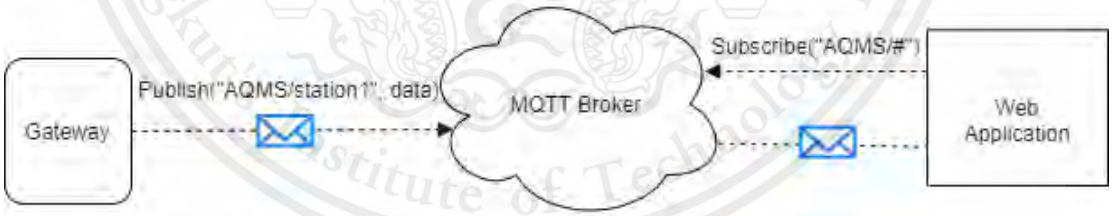

Figure 27: MQTT Communication Diagram from Gateway to Web Application

We implement the gateway to publish whenever it received data from the station, process the data belong to which station and publish with the MQTT topic of its station. The web server will get the data when the gateway publishes to the MQTT broker and be able to identify what station sending this data from its MQTT topic.

5.3 Hardware Development

We are first needed to test our embedded hardware reading sensor data and check the LoRa communication between the weather station and gateway. The weather station will send all the sensing values to the gateway, then the gateway passed those receiving data to the MQTT broker. In LoRa communication, the gateway can only receive the message in bits and bytes which convert into string text, so we implement the weather station to wrap all the sensing values into a long string text that matches JSON standard format. The message will stay on that string format until it arrives Django server, the message will be converted into the real JSON format in the backends in order to get each sensing data easily from the JSON format.

5.3.1 Weather Station

We test the weather station by monitoring the reading terminal on our laptop so that we can see all the sensors reading and look at how accurate the data information. We try to send the sensory data to the gateway via LoRa.

Our sensor node consists of the following components:

- LoPy as main microcontroller board.
- Pycom expansion board 2.0 with the LoPy board.
- Temp&Hum 2 Click board as a temperature and humidity sensor
- HPMA115S0 as an air quality sensor.
- GSM/GNSS Click board as a location tracking sensor.

Both the Temp&Hum and the air quality sensor are sending the sensing value to LoPy through I2C serial communication. GPS module is sending the corresponding location of the weather station to LoPy through UART serial communication. In the end, LoPy compresses all the sensing value from those connected serial port into a long string that matches JSON format, then sends it to the gateway with LoRa communication.

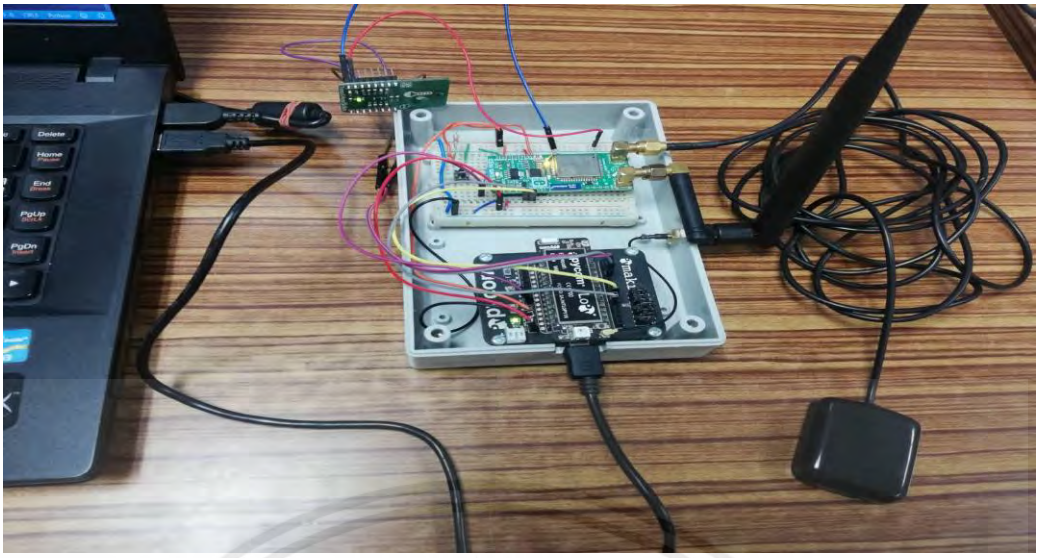

Figure 28: Testing of Weather Station

```
ACK OK
Timeout for ACK, No response from station
button_value:1,temperature:27.08977,humidity:65.28778,latitude:13.7292,longtitude:100.775,
ACK OK
ACK OK
Timeout for ACK, No response from station
button_value:1,temperature:27.07108,humidity:65.4129,latitude:13.72927,longtitude:100.775,
ACK OK
Unhandled exception in thread started by <function thread_read_gps_coordinate at 0x3ffe7c40>
Traceback (most recent call last):
```

Figure 29: Sensor Reading Test of The Weather Station

5.3.2 Gateway

We test our gateway by setting LoRa gateway to receive the data from the weather station, connect to Wi-Fi, connect to MQTT cloud server and send the data to the MQTT cloud server.

Figure 30: Testing of LoRa Gateway

```
ACK OK
ACK OK
Previous board is not available anymore
> Failed to connect (Error: Port is not open). Click here to try again.
Connecting to COM5...
Connected to WiFi
```

Figure 31: Wi-Fi Connection Testing of The Gateway

5.4 Software Development

5.4.1 MQTT system

This is our MQTT settings code for connecting to MQTT cloud server.

```
# MQTT Settings
# CloudMQTT Broker
MQTT_Broker = "postman.cloudmqtt.com"
MQTT_Port = 15589
MQTT_Username = "nswnnfoc"
MQTT_Password = "HbdSr-1z9oYG"
Keep_Alive_Interval = 60
MQTT_Topic = "AQMS/#"
```

There are the important variables as follow:

- MQTT_Broker is the URL of the cloud server.
- MQTT_Topic is the string of the topic that our web server subscribes
- MQTT_Username and MQTT_Password are the information the MQTT cloud server provides us for connecting the client to the CloudMQTT broker server.
- Keep_Alive_Interval is time period in seconds that the broker will check whether the connect is still open and working or not. Set to 0 to make it disabled.
- MQTT_Topic is name of the channel that the clients subscribe or publish to the broker. Our application server subscribes with the topic “AQSM / # ”. The hash symbol indicates that our server is subscribe to all channel that has the topic “AQMS” level as the top level.

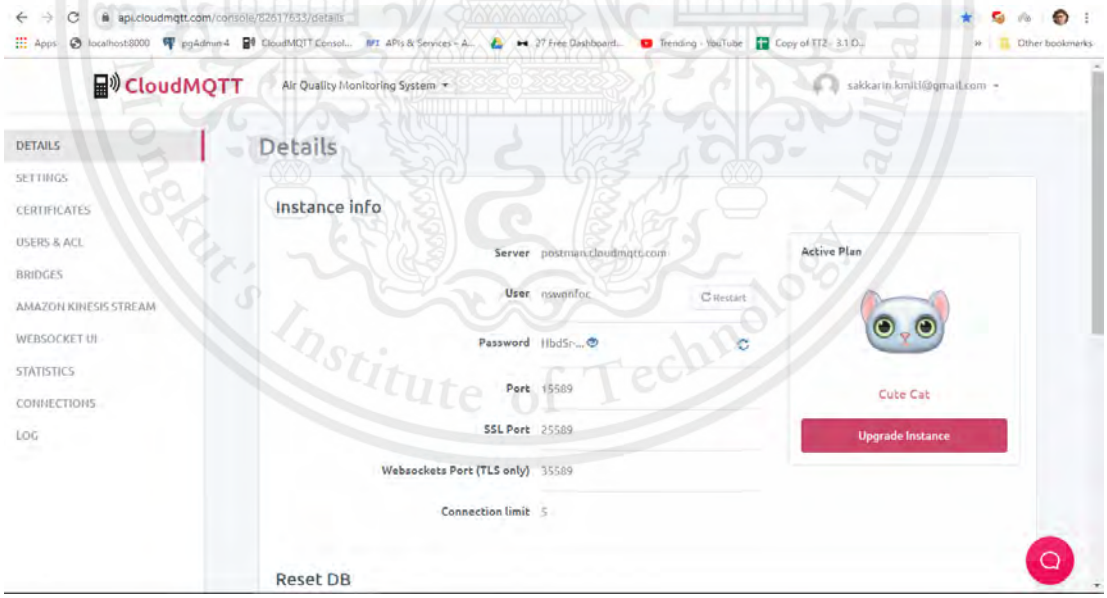

Figure 32: Server Information of CloudMQTT Website

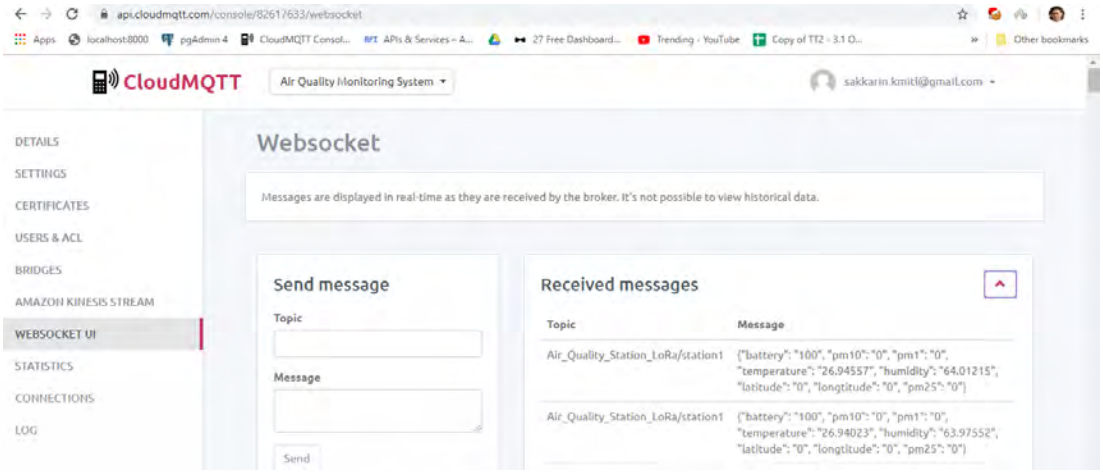

Figure 33: WebSocket Page of CloudMQTT Website

5.4.2 Grafana

From the previous semester, our system lacks capability of displaying history graph of the weather information. So, we try to develop history graph by using Grafana which is popular visualization tool for graph presentation. Grafana can interact with our exist PostgreSQL database and display all of our data in history graph. The server of Grafana is local, so we can manage and develop very easily. We configure the Apache server to route to the Grafana by access our domain name follow by slash (/) grafana.

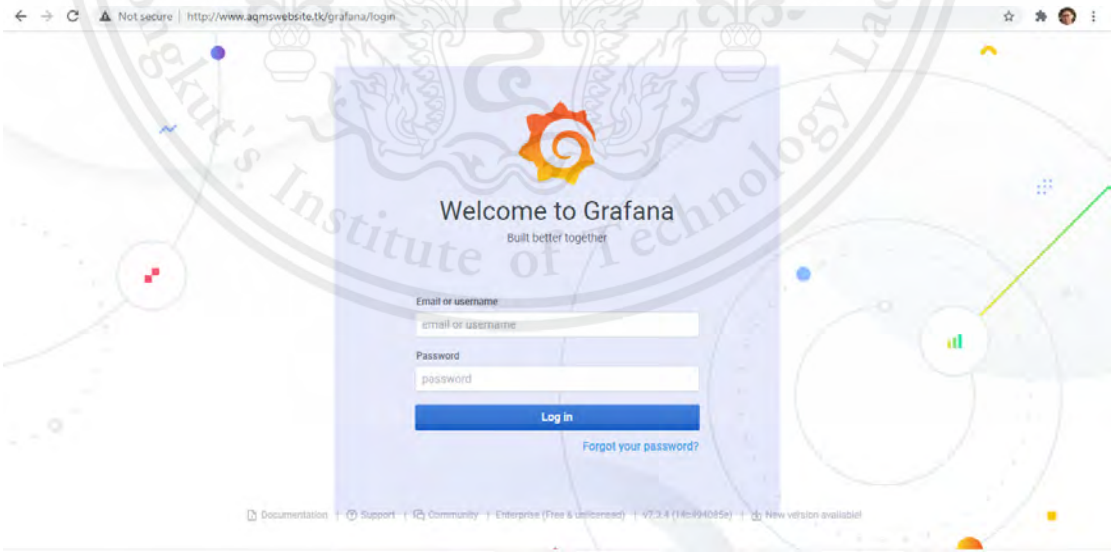

Figure 34: Grafana Login Page

The figure below shows Grafana dashboard that we created. Grafana server connects to our PostgreSQL database server and then make data queries with time-series data in order to display in data in visualization graph and the other panels that available in Grafana plugin

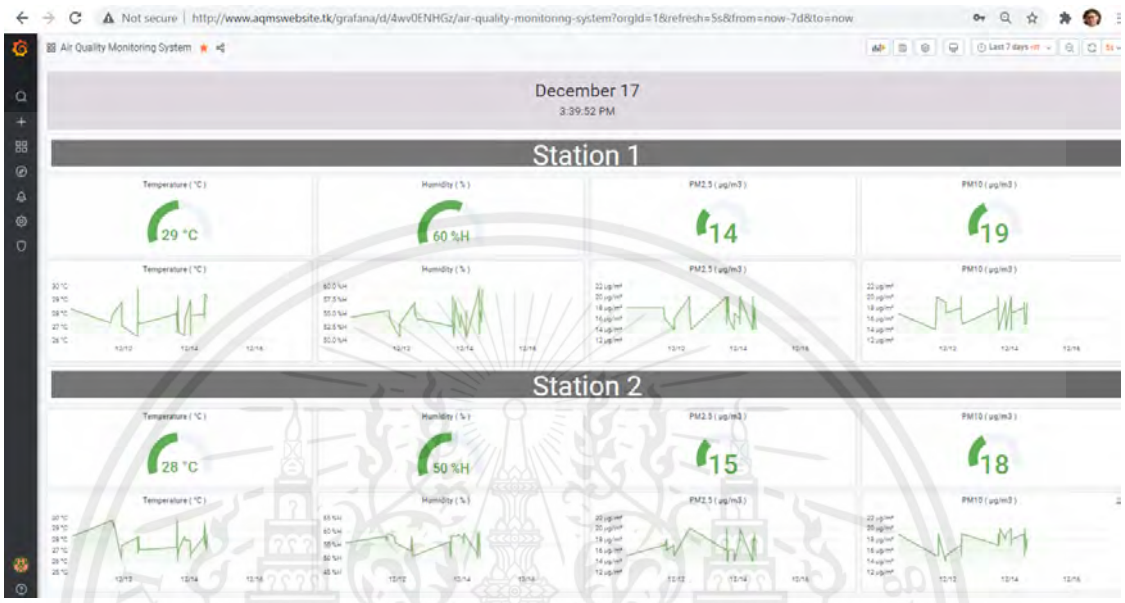

Figure 35: Grafana Dashboard

5.4.3 Dashboard Page

This page is the main dashboard of our system. We try to use Grafana in order to display the history graph of our data. Grafana graph can also display the data in real-time according to our database. We use component of HTML called iframe in order to import Grafana graph into our existing dashboard. The dashboard can also change theme to dark mode.

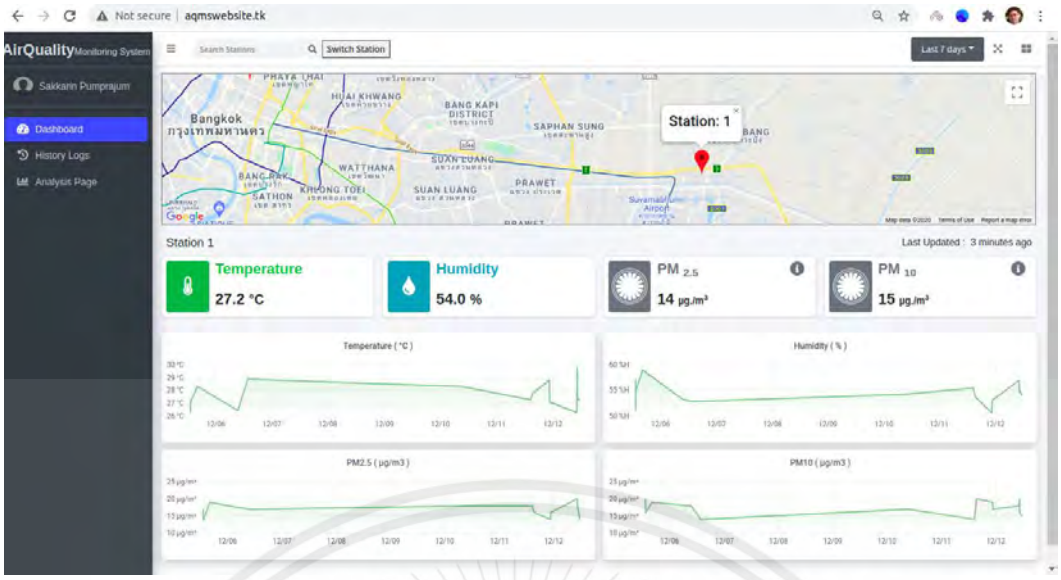

Figure 36: Integrate Grafana Graph into Our Dashboard

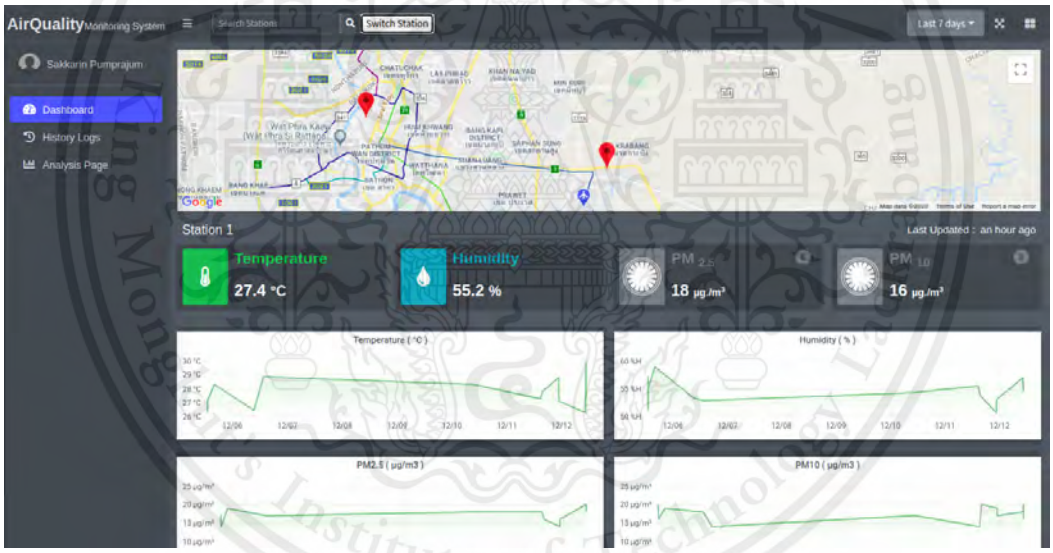

Figure 37: Dashboard in Dark Mode

We develop the dropdown button to change the time range of the Grafana graph. The user can change the time range of the graph very easily by clicking at the dropdown button on the top right bar.

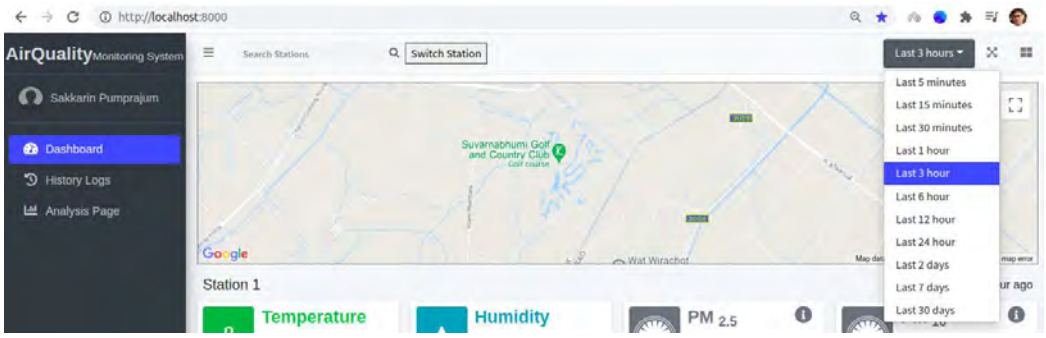

Figure 38: Time Range Dropdown in Dashboard

We also develop the tooltip that inform the information of the $PM_{2.5}$ concentration, and PM_{10} concentration

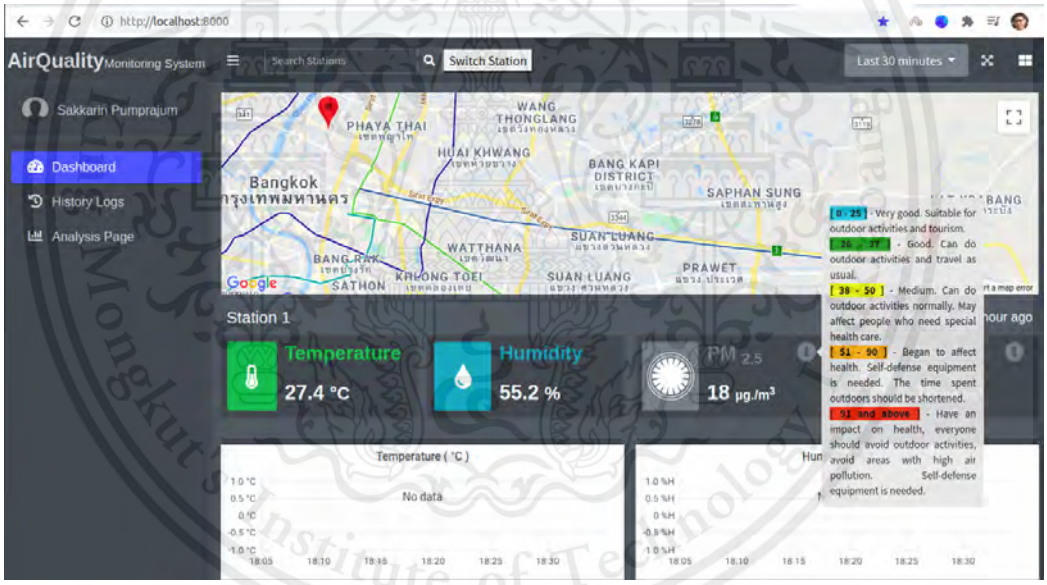

Figure 39: Tooltip Information in Dashboard

Chapter 6
Preliminary Result

This chapter demonstrate the result of our project.

6.1 Web Application Results

6.1.1 Dashboard Page

This page is the main page of our web application. After the user successfully login into the system, it shows overview of the air quality information and Google Maps interface that displays every available station of our system. It also displays the Grafana graphs that indicate the history of temperature and relative humidity.

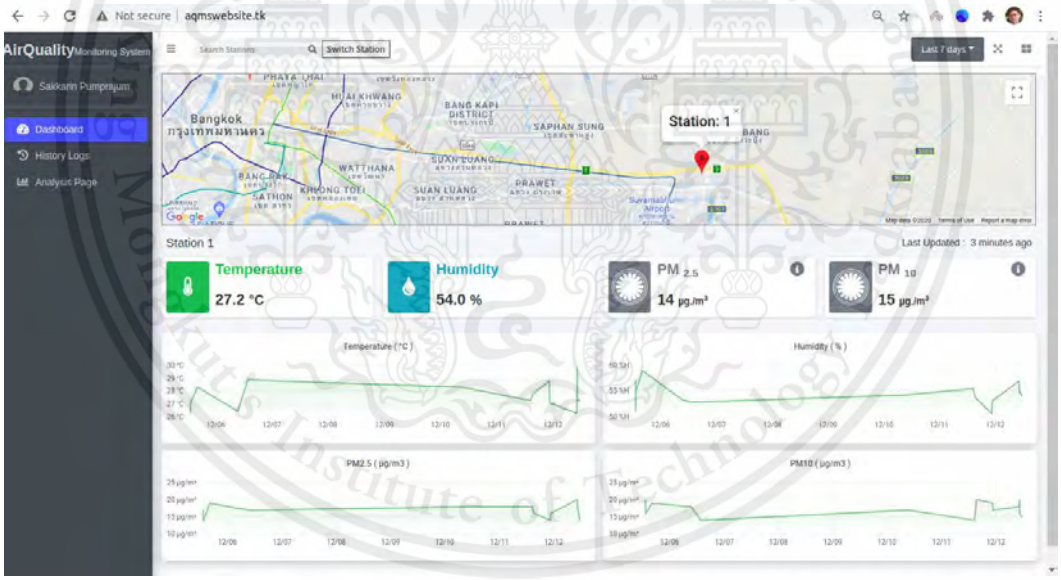

Figure 40: Dashboard Page

6.1.2 Data Management Database

This page displays all the history log in our database sort in time order.

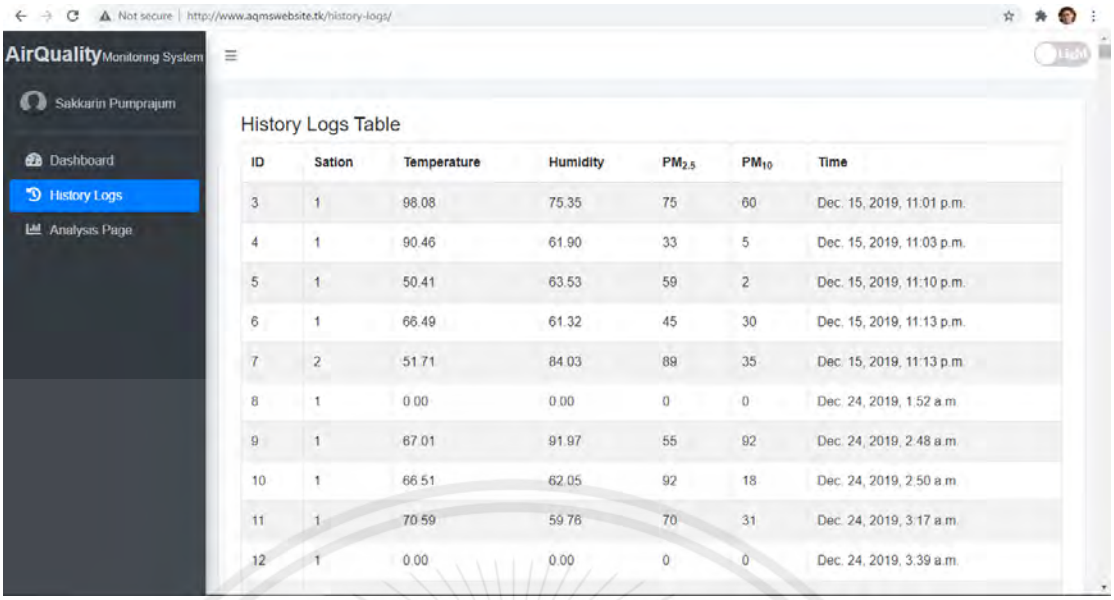

Figure 41: Log History Page

6.1.3 Data Management Database

Data Management Service is used to manage user accounts. Additionally, we use PostgreSQL to manage weather data.

Figure 42: Administrator Page

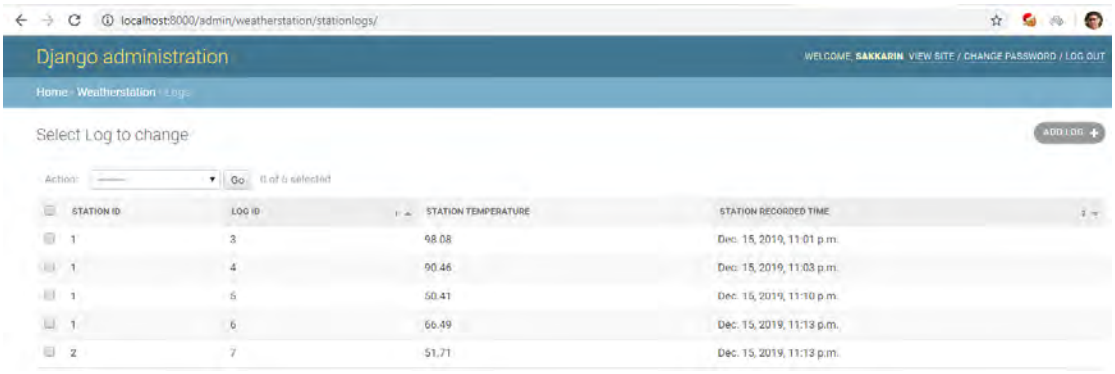

Figure 43: Database of Station Log

6.1.4 Sign In Page

This page allows user to fill in username and password in order to log in to the system. If the user does not have account, they can create one by clicking at the Sign up button.

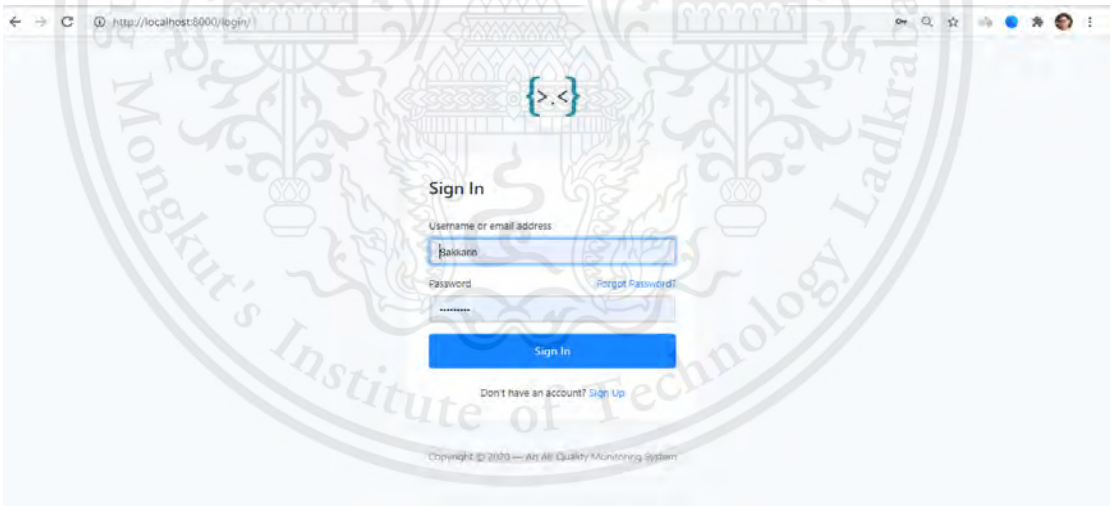

Figure 44: Sign In Page

6.1.3 Registration Page

The page allows user to fill in their information in order to create an account of the system.

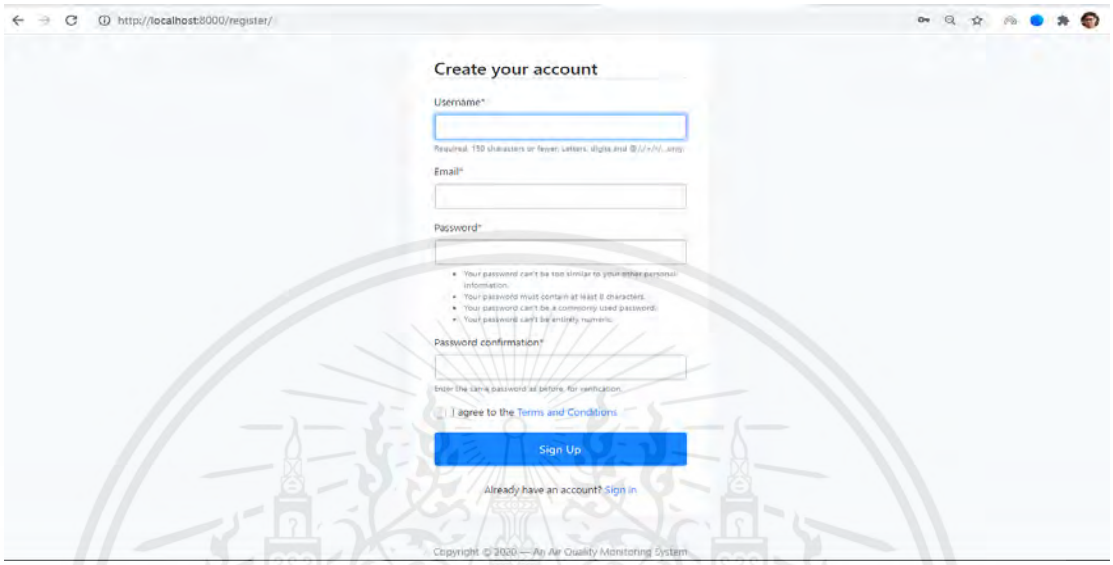

Figure 45: Registration Page

6.2 Hardware Results

6.2.1 Weather Station

The weather station is the station node that consists of LoPy as LoRa node, Temp&Hum sensor, GPS module, and air quality ($PM_{2.5}$) sensor.

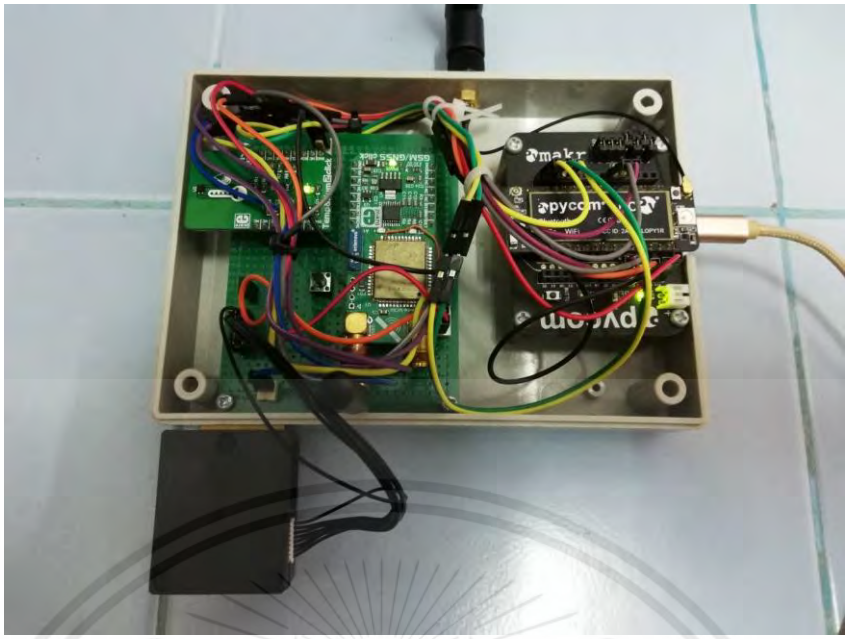

Figure 46: Weather Station

6.2.2 Gateway

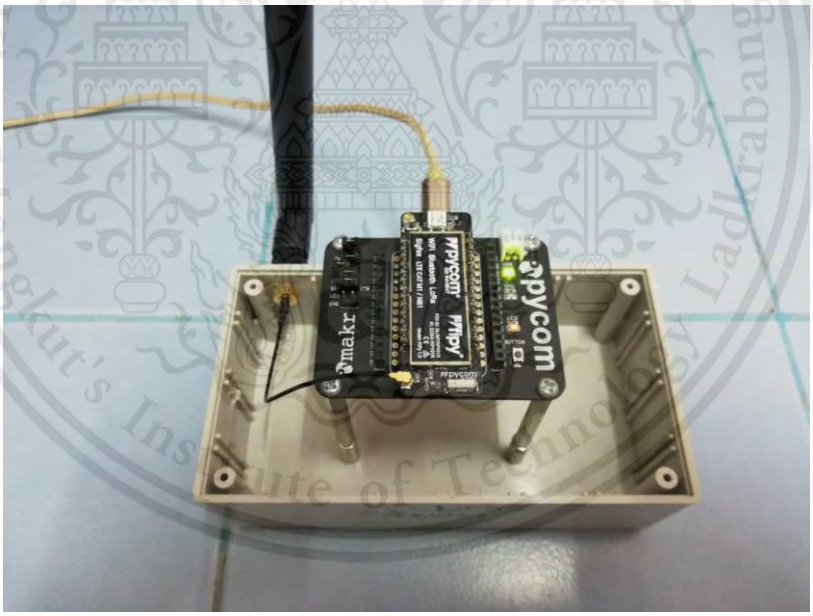

Figure 47: Gateway

Bibliography

[1] Chowdhury, Z., Zheng, M., Schauer, J.J., Sheesley, R.J., Salmon, L.G., Cass, G.R., Russell A.G., 2007. Speciation of ambient fine organic carbon particles and source apportionment of PM2. 5 in Indian cities., vol. 112, no. D15.

[2] Pathak, R.K., Wu, W.S., Wang, T., 2009. Summertime PM 2.5 ionic species in four major cities of China: nitrate formation in an ammonia-deficient atmosphere., vol. 9, no. 5, pp. 1711–1722

[3] Pongpiachan, S., Choochuay, C., Chalachol, J., Kanchai, P., Phonpiboon, T., Wongsuesat, S., Chomkhae, K., Kittikoon, I., Hiranyatrakul, P., Cao, J., Thamrongthanyawong, S., 2013. Chemical characterisation of organic functional group compositions in PM2. 5 collected at nine administrative provinces in northern Thailand during the Haze Episode in 2013., vol. 14, pp. 3653–3661.

[4] Devaraju, J.T., Suhas, K.R., Mohana, H.K., Patil, V.A., 2015, Wireless Portable Microcontroller based Weather Monitoring Station.

[5] Product Manual v1.xEx – 802.15.4 Protocol, XBee/XBee-PRO RF Modules, 2009

[6] Medilla Kusriyanto, Agusti Anggara Putra., 2018. Weather Station Design Using IoT Platform Based On Arduino Mega

[7] Tony DiCola, 2019. MicroPython Basics: What is MicroPython Available: https://learn.adafruit.com/micropython-basics-what-is-micropython/overview

[8] Semtech, 2019. What is LoRa? [online], Available: https://www.link-labs.com/blog/what-is-lora

[9] Brian Ray, 2018. What Is LoRa? A Technical Breakdown [online], Available: https://www.link-labs.com/blog/what-is-lora https://www.link-labs.com/blog/what-is-lora

[10] Hivemq.com, 13 October 2019. "Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3" [online], Available: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment

[11] Python Software Foundation, 2020. Paho MQTT client library description, Available: https://pypi.org/project/paho-mqtt/

[12] uMQTT Client Package, 2017, uMQTT library description and license, Available: https://kroesche.github.io/umqtt/index.html

Appendix A
Use Case Description

A.1 Use Case Description

Use Case: Sign up

Primary Actor: User

Main Success Scenario: The user fills up the required form and submit registration. The server will examine information in order to create account, and notify to the user whether it succeeds or fails.

Exception scenario 1: Some information is incorrect. The system will ask the user to correct the incorrect information.

Use Case: Log in

Primary Actor: User

Main Success Scenario: The user fills in username and password. The server will verify the information and display the pop-up window to the user whether it succeeds or fails.

Alternate scenario 1: Password is incorrect. The system will ask the user to fill in their username and password again.

Use Case: Log out

Primary Actor: User

Main Success Scenario: The user selects logout option. The server will display the pop-up window to the user when it succeeds to log out.

Use Case: Change password

Primary Actor: User

Main Success Scenario: The user selects change password option. The system will ask the user to fill in the current password and new password. The system will examine the password and display the pop-up window to the user whether it succeeds or fails. If it succeeds, the system will update a new password and notify the change to the user via email.

Alternate scenario 1: Either the current password or new password is incorrect. The system will ask the user to fill in again.

Use Case: Edit profile

Primary Actor: User

Main Success Scenario: The user selects edit profile option. After the user completed editing his profile, the system will update the change.

Use Case: View all data information

Primary Actor: User

Main Success Scenario: After the user logs in to the system successfully, the system will redirect the user to the dashboard that represent all things of information.

Alternate scenario 1: The user is in the edit page, then the user selects the home button. The system will redirect the user to the main page which is the dashboard for viewing all the data information.

Use Case: Select a weather station

Primary Actor: User

Main Success Scenario: The user selects the weather station that they want to observe, the system will automatically change to all data on the dashboard according to the selected weather station.

Use Case: Create user

Primary Actor: Administrator

Main Success Scenario: An administrator selects the create user option. The system allows the administrator to create new user to the system.

Use Case: Update user

Primary Actor: Administrator

Main Success Scenario: : An administrator selects the update user option. The system allows the administrator to update user information to the system.

Use Case: Delete user

Primary Actor: Administrator

Main Success Scenario: An administrator selects the delete user option. The system allows the administrator to delete the existing user from the system.

Use Case: Find user

Primary Actor: Administrator

Main Success Scenario: : An administrator selects the find user option. The system allows the administrator to find the existing user in the system.

Use Case: View session

Primary Actor: Administrator

Main Success Scenario: : An administrator selects the view session option. The system allows the administrator to view the session of every user in the system.

Use Case: Inspect a weather station condition

Primary Actor: Engineer

Main Success Scenario: An engineer checks a weather station hardware and inspects the condition of the station.

Use Case: Change battery

Primary Actor: Engineer

Main Success Scenario: After an engineer inspects the condition of the station. The engineer can change battery of the station.

Use Case: Update weather station

Primary Actor: Engineer

Main Success Scenario: After an engineer inspects the condition of the station. The station is not performed correctly, so it needs to update by the engineer. The engineer will pick out the weather station to update the hardware.

Use Case: Return weather station

Primary Actor: Engineer

Main Success Scenario: After an engineer update the weather station completely. The engineer will put the weather station at the same point where it was taken to make an update.

Use Case: Plant a new weather station

Primary Actor: Engineer

Main Success Scenario: The engineer carries the new weather station to the desired place to plant the new station and set up the hardware.