R e p o r t

IoT term project report

제출일	2021. 12. 16
전공	임베디드시스템전공
과목	IoT기초
학번	2019148039
담당교수	엄수홍
이름	이준희

목차

Ⅰ. 서론

1. 연구 개발 배경

살면서 한 번쯤은 기상청 날씨와 현재 날씨가 안 맞아 우산을 안 가져왔는데 비가 온다거나, 비가 안 온다해서 창문을 열어놓고 나왔는데 비가 온다거나 하는 경험이 있을 것입니다. 거기다 최근 코로나까지 겹치면서 항공기 운항 편수가 급감해 기상 정보 수집에도 차질이 생겨 더더욱 기상예보가 안 맞는 일이 벌어지고 있습니다. 이를 조금이나마 해결하고자 이번 주제를 선정하게 되었습니다.

2. 연구의 목표와 방향

우리가 흔히 보는 날씨는 예보로 실시간 데이터가 아닙니다. 따라서 현재 비가 오더라도 날씨 앱이나 기상 사이트에 구름 많음으로 표시되는 이유가 이 때문입니다. 이를 해결하고자 날씨 예보가 아닌 실시간 날씨 데이터를 사용자에게 제공하는 방식을 생각하게 되었습니다. 이 프로젝트의 목표는 IoT의 전반적인 이해를 위해 통합 시스템을 구축해 보는 것을 목표로 진행하였습니다. Network를 구성하고 Server를 구축해 사용자가 어디에 있든 방 내부 온습도와 날씨 데이터를 IoT 장비를 통해 받아볼 수 있으며 또 원격으로 제어까지 가능하게 하는 것이 이번 프로젝트의 진행 방향입니다.

3. 연구계획 일정

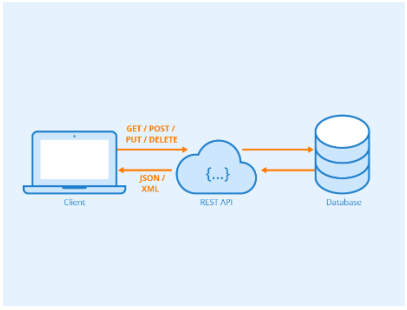
11/25 – DDNS, 라우터 포트포워딩, 보안 규칙, 방화벽, 외부 망 Setting
11/26 - Docker => MySQL DataBase, PHP Web Server, Message Server 생성
11/27 – DB 설계 및 Table, Column 생성
11/28 – ESP8266 DB연결 테스트
11/29 – ESP8266 DB Query Insert 테스트
11/30 – ESP8266 Json 테스트
12/01 – ESP8266 API Parsing and Json data extract
12/02 – ESP8266 extracted data DB Query Insert test
12/03 – PHP web 설계
12/04 – Web Site 개설
12/05 – PHP DataBase 연동 및 화면에 출력
12/06 – Web publishing(Html, CSS, JS, jQuery, Ajax)
12/07 – Python Message Server Programming
12/09 – Servomotor test
12/11 – Python Message servmotor control
12/13 – 실생활 테스트

Ⅱ. 본론

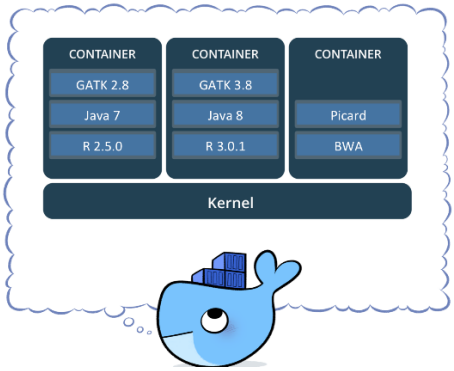
1. 사용 기술

본 프로젝트에서 MCU로 ESP8266을 사용하였으며 Sensor및 하드웨어로는 WiFi Module, DHT22, LCD, Servomotor를 사용하였습니다. 그 밖 Server와 API로는 Docker, Maria DataBase, PHP Web Server, Python Message Server, Open Weather Map API와 Telegram Bot API를 사용하였습니다.

서버는 제공하다 라는 뜻을 가진 serve 에 er 를 붙인 단어로 클라이언트(사용자 컴퓨터)에게 여러 가지 서비스를 제공하는 것을 말합니다. 서버도 컴퓨터이기 때문에 OS 가 필요한데 이 프로젝트에 사용한 서버는 Debian 계열 리눅스를 사용하였습니다. 리눅스는 우리가 흔히 사용하는 macOS 나 Windows 와 같은 운영체제로 오픈소스 OS 이며 강력한 네트워크망 구축과 Multitasking, Multithread, 멀티유저시스템을 지원하는 서버 운영체제에 특화된 OS 입니다.
우리가 흔히 사용하는 Windows 도 Windows Server 운영체제가 따로 존재하며, macOS 의 기반인 Unix 도 서버 OS 로 많이 사용되나 둘 다 유료이기 때문에 리눅스를 선택하였습니다.



API 는 Application Programming Interface 의 약자로 흔히 레스토랑에 빗대어 표현하는데 손님(프로그램)이 웨이터(API)에게 주문하면 웨이터는 주문 내역을 주방(API 제공 서버)에 가져다줍니다. 그럼 주방에서 요리해서 웨이터에게 주면 웨이터는 손님에게 음식을 가져다줍니다. 이처럼 손님은 주방에서 무슨 일이 일어나는지 모르지만 주문한 음식을 제공받아 식사만 하면 되게 하는 게 API 의 기능이라 할 수 있습니다. 가져다 쓰려는 API 기능을 어떻게 구현했는지는 모르지만, API 가 가져다주는걸 사용만 하면 되기 때문에 편하게 프로그램을 제작할 수 있습니다.
이 프로젝트에서는 Open Weather Map API 와 Telegram Bot API 를 사용하여 실시간 날씨 정보를 받아오고 Telegram 을 통해 서보모터를 제어할 수 있게 구현하였습니다.

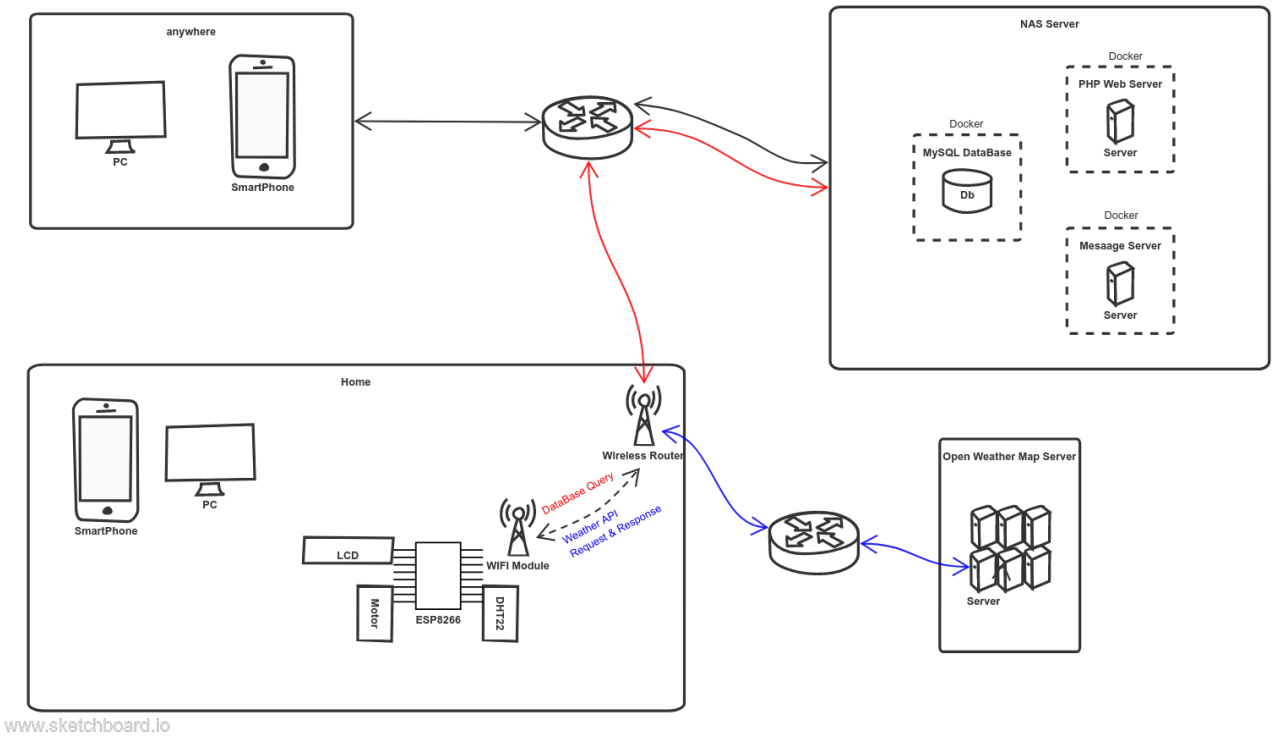


Docker는 Environment disparity를 해결해 주는 소프트웨어로 Environment disparity란 쉽게 말해 윈도우와 리눅스처럼 다른 머신 에서도 같은 환경을 구현할 수 있게 해주는 것을 말합니다. 또한 Docker에는 컨테이너 라는 개념이 존재하는데 각 컨테이너는 분리되어 있으며 한 개의 서버에 각기 다른 수많은 컨테이너를 갖을 수 있어 이 프로젝트에서는 MySQL DB와 PHP Web Server, Python Message Server 3개의 컨테이너를 사용하였습니다.

DataBase는 다수의 사용자가 사용하는 데이터들의 공유와 운영을 위해 저장해 놓는 공간을 말합니다. 프로그래머가 필요에 의해 프로그램에 넣어 놓은 데이터 등 필연적으로 많은 데이터가 생성되는데 데이터베이스를 사용하지 않으면 이 데이터들은 프로그램을 종료하는 순간 전부 날아가게 됩니다. 이런 현상을 방지하기 위해 데이터들을 데이터베이스에 넣고 보관하는 방법을 사용합니다. DB에도 다양한 종류가 있는데 MariaDB를 사용한 이유는 MySQL 코드 기반으로 한 오픈소스 RDBMS 소프트웨어이기 때문에 사용하였습니다. 이 프로젝트에서는 Open Weather Map API에서 받아온 데이터와 DHT22데이터를 통합하고 서보모터를 제어하는 데 사용됩니다.

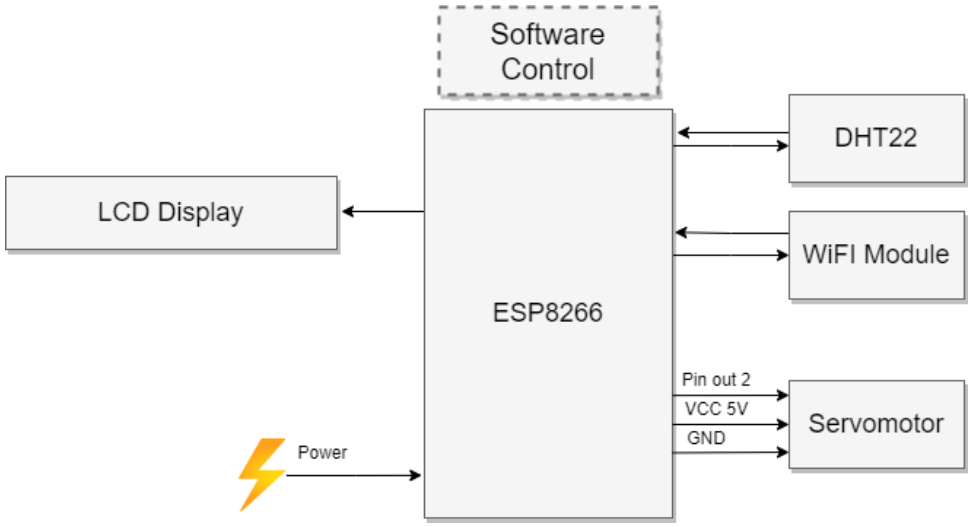
PHP는 웹페이지를 만들 때 서버 측에서 실행되는 서버 사이드 언어입니다. C언어를 기반으로 만들어졌으며 동적 웹페이지를 쉽고 빠르게 만들 수 있도록 해주는 데 그 목적을 가지고 있는 언어입니다. PHP로 작성된 코드를 HTML 스크립트 안에 추가하면, 웹서버는 해당 PHP 코드를 해석하여 동적 웹페이지를 생성합니다. 서버 사이드를 보통 Backend라고 말하며 Backend언어에도 Nodejs, Rails, Django, SpringBoot 등 다양하게 있지만 간단한 사이트를 만들기에는 PHP가 가장 적합하며 컴파일을 하지 않고 HTML 스크립트 안에 추가하면 바로 사용이 가능하기 때문에 사용하였습니다. 이 프로젝트에서 PHP 웹서버는 DB와 연동되어 ESP8266에서 DB에 insert 한 데이터들을 뽑아 웹페이지에 뿌려주는 역할을 합니다.

2. Concept Map



이 프로젝트의 Concept Map입니다.

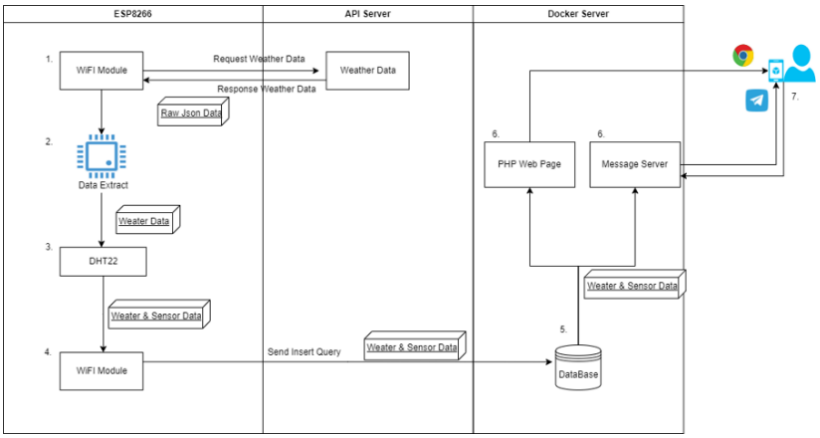
3. 하드웨어 구성도



하드웨어 구성도입니다.

ESP8266 은 LCD Display 와 DHT22, WiFi Module, Servomotor 가 연결되어 있습니다.
Servomotor 는 MG995 모델을 사용하여 동작전압은 최소 4.8V 에서 최대 7.2V 까지이며 8.5 kgf·cm (4.8 V), 10 kgf·cm (6V)의 토크 값을 가지고 있습니다. 실제로 창문을 닫기 위해 토크 값을 계산하여 구매하였습니다.

4. 동작 시나리오



ESP8266 에는 WiFi Module 과 DHT22 온습도센서, LCD, Servomotor 가 연결되어 있고 WiFi Module 을 통해 Open Weather Map API Server 에 접속하여 현재 날씨 데이터를 요청합니다.

API Server 에서 응답한 데이터는 JSON 형식으로 오며 이 JSON 에서 필요한 정보만을 split 하여 DHT22 에서 얻은 온습도 데이터와 함께 DB Weather Table 에 저장시킨 뒤 LCD 화면에 출력시킵니다.

DB Weather Table 에 저장된 데이터는 PHP Web Server 와 Telegram Message Server 에서 사용되며 언제 어디서든 어떤 플랫폼에서든 실시간 날씨와 방 내부 온도를 체크할 수 있습니다.

모터 제어는 Telegram 으로 정해놓은 특정 명령어를 전송하면 그 명령어는 또다시 DB doorState Table 로 들어가며 ESP8266 에서 현재 창문 상태와 DB 창문 상태가 다르면 모터를 움직여 창문을 닫습니다. 반대로 창문을 열 수도 있습니다.

5. 네트워크 구성

서버가 외부와 데이터를 주고받기 위해서는 네트워크망을 구성해야 합니다. DDNS 도메인 등록과 라우터 구성, 게이트웨이, 네트워크 인터페이스, 방화벽 구성, 트래픽 제어, HTTP/HTTPS, TLS/SSL, 인증서 등 많은 부분이 있으나 본 과목의 취지와는 어긋나므로 생략하고 포트 포워딩 부분에 대해서만 언급하도록 하겠습니다.

우선 서버에서 사용할 서비스의 포트를 열어주고 라우터에서 포트 포워딩(포트 매핑)을 해줍니다.

Port 란 컴퓨터의 LAN 선은 하나인데 통신을 필요로 하는 프로그램이 다수일 때 이 다수의 프로그램을 구별할 수 있는 번호를 Port 라고 부릅니다. 포트 포워딩은 컴퓨터 네트워크에서 패킷이 라우터나 방화벽 같은 네트워크 게이트웨이를 통과하는 동안 네트워크 주소를 변환해주는 것을 의미합니다. 즉 라우터에 어떠한 서비스 포트로 요청이 오면 어디로 연결해야 하는지 이정표를 달아주는 것을 포트 포워딩이라 합니다.

이 프로젝트에서는 Server, DB, phpMyAdmin, SSH, Web 총 5 개의 포트 포워딩을 진행하였습니다.

Tera_SSH	192.168.0.4	TCP	TCP
DataBase	192.168.0.4	TCP(3307~3307)	TCP(3307~3307)
PHPAdmin	192.168.0.4	TCP(8080~8080)	TCP(8080~8080)
NAS_VPN	192.168.0.4		
WebDAV(http)	192.168.0.4		
WebDAV(htt...	192.168.0.4		
WebPage	192.168.0.4	TCP(81~81)	TCP(81~81)

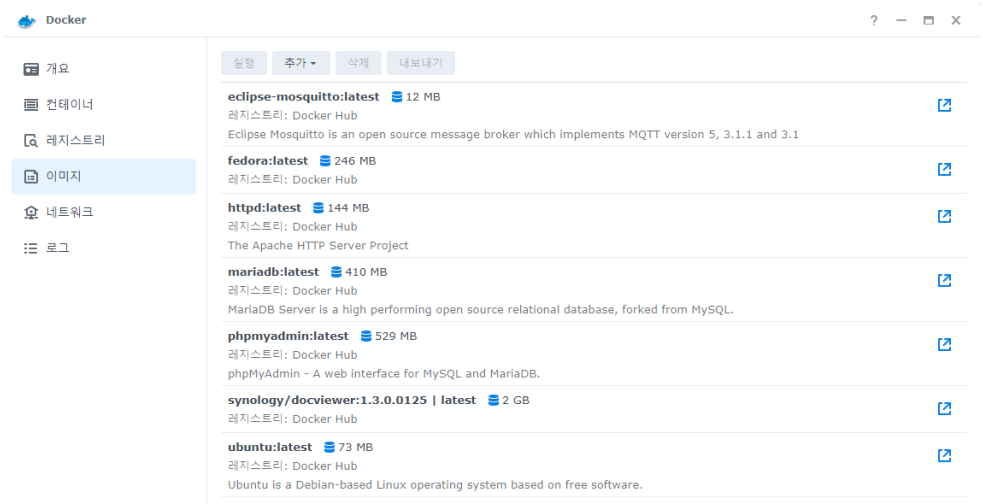
서버포트 및 SSH포트는 보안상 공개하지 않습니다.

웹페이지는 81 번 포트로 할당하였고 DB 는 3307 포트로 할당하였습니다.

SSH는 Telegram bot을 데몬 프로세스로 만들 때 사용하기 위해 열어 두었고 PHPAdmin은 phpMyAdmin을 내부망에서 사용하기 위해 열어 두었습니다.

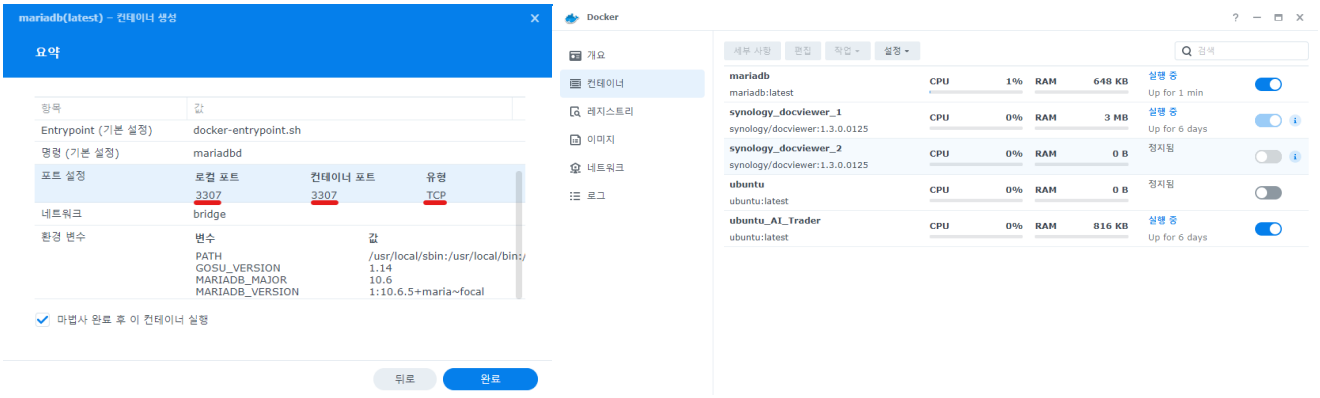
6. 서버 구성

서버에서 Docker 를 사용하여 PHP Web Server 컨테이너와 Maria DB 컨테이너, Telegram Message Server 컨테이너로 나누기 위해 필요한 이미지들을 다운받아줍니다.



위 사진은 Docker 에서 필요한 이미지 다운이 완료된 사진입니다.

필요한 이미지들이 다운되었으면 컨테이너를 구성합니다. 컨테이너 구성 시 포트는 라우터에서 설정한 포트를 사용합니다.



컨테이너를 실행하여 서버구성을 완료하고 각종 설정을 진행합니다.

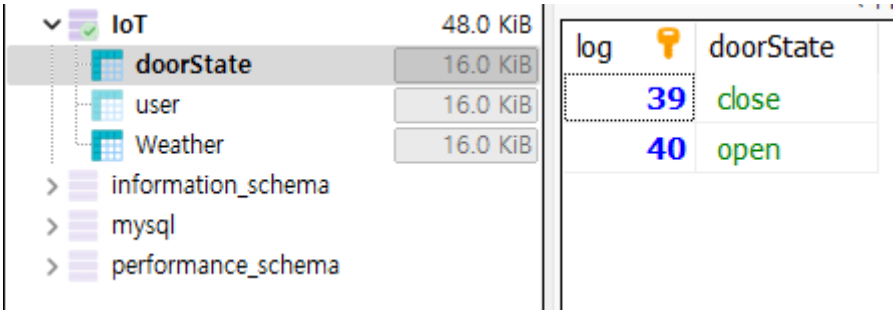
7. DB 구성

학번	이름	학과명	성별	성적
20051201	김철수	컴퓨터	남	85
20050315	이영희	수학	여	95
20040527	홍길동	과학	남	88
20031209	이영희	컴퓨터	여	95
20040307	박희동	수학	남	82

DB는 Table과 식별자, 어트리뷰트, 튜플로 나누어집니다. 하나의 테이블에는 여러 어트리뷰트가 들어가며 이 어트리뷰트를 통해 데이터를 분리합니다. 어트리뷰트는 다른 말로 칼럼(Column)이라고도 부릅니다.

Num	Weather	Temperature	FeelsTemp	MinTemp	MaxTemp	Humidity	Wind	innerTemperature	innerHumidity
2	Clouds	9.10	8.32	9.10	9.10	57.00	1.80	23.50	47.70
3	Clouds	9.10	8.32	9.10	9.10	57.00	1.80	23.60	46.70
4	Clouds	9.10	8.32	9.10	9.10	57.00	1.80	24.00	45.80
5	Clouds	8.65	8.65	8.65	8.65	60.00	0.74	24.40	45.10
6	Clouds	8.65	8.65	8.65	8.65	60.00	0.74	24.40	45.20
7	Clouds	8.65	8.65	8.65	8.65	60.00	0.74	24.60	45.00
8	Clouds	8.73	8.73	8.73	8.73	59.00	0.80	24.90	44.40
9	Clouds	8.73	8.73	8.73	8.73	59.00	0.80	24.80	44.40

위 사진은 이번 프로젝트에서 사용하는 Weather Table입니다.
Num은 autoincrement의 속성을 주어 자동으로 데이터가 들어갈 때 마다 1씩 증가합니다.
칼럼으로는 Weather(구름양), Temperature(온도), FeelsTemp(체감온도), MinTemp(최저온도), MaxTemp(최고온도), Humidity(습도), Wind(바람), innerTemperature(DHT22온도값), innerHumidity(DHT22습도값)으로 나누어 집니다.

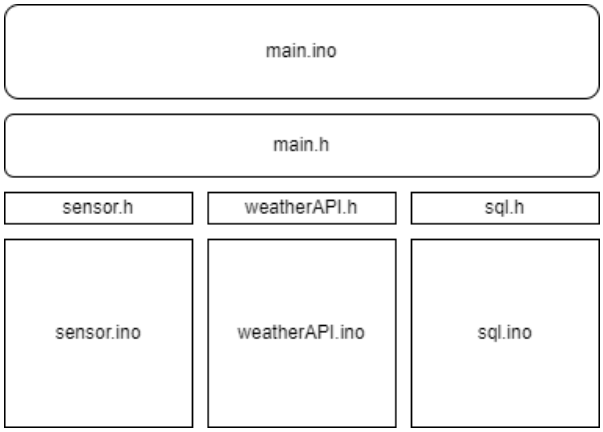


doorState Table은 텔레그램을 통해 사용자가 창문을 열고 닫고 하게 하기 위해 만들었습니다.

8. 코드리뷰

프로젝트는 ESP8266(C), Web Page(PHP)와 Telegram Bot(Python) 총 3 개의 프로그램이 존재합니다.
하지만 본 교과목은 IoT 과목이기 때문에 ESP8266 코드 리뷰를 중점적으로 나머지 2 개의 코드는 간략하게만 설명하고 넘어가도록 하겠습니다. 모든 코드는 GitHub 에 올려 두었습니다. (깃허브 주소는 코드리뷰 마지막에 있습니다)

8_1. ESP8266



프로그램은 위 사진과 같이 구성되어 있습니다. 기능별로 분리하여 재사용 가능하며 독립적으로 상호작용이 가능하도록 코드를 구성하였습니다. (아래 코드 리뷰의 코드는 설명을 위해 중요하지 않은 부분과 예외처리 부분은 삭제한 버전입니다.)

main.h

```
#ifndef __MAIN_H__
#define __MAIN_H__

#include "sql.h"
#include "weatherAPI.h"
#include "sensor.h"
#include <TaskScheduler.h>

#define SAME 0
#define _30MIN 1800000

char ssid[] = "Secure";
char password[] = "1CF0011189";
void dbUpdateTask();
#endif
```

필요한 헤더 파일을 include 시켜 주고 dbUpdateTask() 함수는 main.ino에 선언되어 있으며 TaskScheduler를 통해 30분마다 DB에 insert 시키는 작업을 수행합니다.

main.ino – global setting

```
#include "main.h"

SQL_Column sqlColumn;

Task dbUpdate_Task(_30MIN, TASK_FOREVER, &dbUpdateTask);
Scheduler dbUpdateScheduler;

String nowWindowState="NULL";
```

전역 변수 및 전역 클래스와 구조체 설정으로 SQL_Column 구조체를 선언하고
Task dbUpdate_Task(_30MIN, TASK_FOREVER, &dbUpdateTask)와 Scheduler dbUpdateScheduler 를 통해 dbUpdateTask 함수를 매 30 분마다 호출합니다. SQL_Column 구조체는 sql.h 내부에 존재합니다.
nowWindowState 는 처음 실행시 DB doorState Table 에 있는 최신값으로 변경되며 추후 현재 상태와 DB 상태를 비교하는데 사용됩니다.

main.ino – setup()

```
void setup()
{
  Serial.begin(115200);

  initSensor();

  WiFi.init(&EspSerial);
  WiFi.begin(ssid, pass);

  dbUpdateScheduler.init();
  dbUpdateScheduler.addTask(dbUpdate_Task);
  dbUpdateScheduler.enable();

  clearLcd();
}
```

initSenor()함수로 사용할 하드웨어 및 센서를 선언합니다. initSensor 함수는 sensor.ino 내부에 존재합니다.
WiFi.init(&EspSerial)과 WiFi.begin(ssid, pass)을 통해 와이파이에 연결하며 dbUpdateScheduler.init(), dbUpdateScheduler.addTask(dbUpdate_Task), dbUpdateScheduler.enable 로 전역선언 해두었던 dbUpdate_Task 를 스케줄러에 등록시킵니다.
등록이 완료되면 clearLcd()함수를 통해 LCD 를 초기화시킵니다.

main.ino – void loop()

```
void loop()
{
  // dbUpdateTask() 함수 스케줄러 실행
  dbUpdateScheduler.execute();
  String dbWindowState;

  getWeatherData(&sqlColumn); // 날씨 데이터
  getDhtData(&sqlColumn);     // 온습도 센서 데이터

  printLcd(&sqlColumn);       // LCD 최신버전으로 다시 출력

  getDB_WindowState(&dbWindowState);  // DB 창문 상태

  if(nowWindowState.compareTo(dbWindowState) != SAME)
  {
    moveMotor(&dbWindowState);
    nowWindowState = dbWindowState;
  }
  delay(1000);
}
void dbUpdateTask()
{
  getWeatherData(&sqlColumn); // 날씨 데이터 받아옴
  getDhtData(&sqlColumn);     // 온습도 센서 데이터 읽어옴
  insertDB(&sqlColumn);       // DB 에 넣음
  clearLcd();                 // 이전 LCD 화면 지우고
  printLcd(&sqlColumn);       // LCD 최신버전으로 다시 출력
}
```

loop에서는 setup에서 생성했던 스케줄러를 실행시킵니다. dbWindowState변수는 DB doorState Table 최신값을 받아오기 위해 사용됩니다. getWeatherData(&sqlColumn)는 Open Weather Map API의 실시간 날씨 데이터를 받아오며 최종적으로 DB에 들어갈 sqlColumn의 주소를 넘겨주어 return 없이 sqlColumn의 값을 변경시킵니다. getWeatherData함수는 weatherAPI.ino에 존재합니다.
getDhtData(&sqlColumn) 또한 sqlCloumn에 DHT22센서의 데이터를 넣습니다.
printLcd(&sqlColumn)는 LCD에 실시간 날씨와 Sensor데이터를 출력시킵니다.
getDhtData함수와 printLcd함수는 sensor.ino내부에 존재합니다.
getDB_WindowState(&dbWindowState)를 통해 현재 사용자가 문을 열거나 닫도록 하였는지 값을 받아옵니다.

if문을 통해 이전에 전역 선언해 두었던 nowWindowState와 비교하여 값이 서로 다르면 nowWindowState 변수를 dbWindowState와 같게 설정하여 moveMotor 함수를 통해 실시간으로 창문을 열고 닫을 수 있게 만들었습니다.
void dbUpdateTask()함수 내부에 존재하는 insertDB(&sqlColumn)을 통해 DB에 날씨와 DHT22센서 데이터를 insert시켜 텔레그램과 웹사이트를 통해서 방 내부 온습도와 날씨 데이터를 볼 수 있습니다.

sensor.h

```
#ifndef __SENSOR_H__
#define __SENSOR_H__

#include "OLED.h"
#include "DHT.h"
#include <Wire.h>
#include <Servo.h>

#define DHTPIN 14
#define DHTTYPE DHT22

#define SERVOPIN 2

#define CLOSE 0
#define OPEN 180

void initSensor();
void getDhtData(struct SQL_Column *sqlColumn);
void clearLcd();
void printLcd(struct SQL_Column *sqlColumn);
void moveMotor(String* dbWindowState);

#endif
```

Sensor와 LCD, 모터를 움직이기 위해 필요한 헤더파일과 각종 메크로, sensor.ino에 있는 함수를 선언합니다.

sensor.ino – initSensor(), getDhtData()

```
#include "sensor.h"

DHT dht(DHTPIN, DHTTYPE);
OLED display(4, 5);
Servo servo;

void initSensor()
{
  char chrLcdDisplay[10];
  dht.begin();
  servo.attach(SERVOPIN);
  display.begin();
  display.print("Booting...",0,0);
  display.print("IOT KIT VER", 2,0);
  sprintf(chrLcdDisplay, "%.1f",2.0);
  display.print(chrLcdDisplay,2,8);
}

void getDhtData(struct SQL_Column *sqlColumn)
{
    sqlColumn->dhtTemperature = dht.readTemperature();
    sqlColumn->dhtHumidity = dht.readHumidity();
    sqlColumn->strDhtTemperature = String(sqlColumn->dhtTemperature);
    sqlColumn->strDhtHumidity = String(sqlColumn->dhtHumidity);
    sqlColumn->strDhtTemperature.toCharArray(sqlColumn->chrDhtTemperature,sqlColumn
                                                    ->strDhtTemperature.length()+1);
    sqlColumn->strDhtHumidity.toCharArray(sqlColumn->chrDhtHumidity,sqlColumn
                                                    ->strDhtHumidity.length()+1);
}
```

void initSensor()함수에서 필요한 센서들을 설정해줍니다.
void getDhtData(struct SQL_Column *sqlColumn) 함수에서는 DHT22의 온습도 데이터를 읽어들인뒤 SQL_Column 구조체에 있는 데이터 형식에 맞게 형변환 시켜 넣어줍니다.

sensor.ino – clearLcd(), printLcd(), moveMotor

```
void clearLcd()
{
  display.print("                    ",0,0);
  display.print("                    ",1,0);
  display.print("                    ",2,0);
  display.print("                    ",3,0);
}

void printLcd(struct SQL_Column *sqlColumn)
{
  display.print(sqlColumn->chrWeather, 0, 0);
  display.print(sqlColumn->chrTemperture,1, 0);
  display.print(sqlColumn->chrHumidity,1, 9);
  display.print("IN ",3,0);
  display.print(sqlColumn->chrDhtTemperature,3,3);
  display.print(sqlColumn->chrDhtHumidity,3, 9);
}

void moveMotor(String* dbWindowState)
{
  if(dbWindowState->compareTo("open")) { servo.write(OPEN); }
  else { servo.write(CLOSE); }
}
```

clearLcd()함수를 통해 LCD화면을 필요할 때 마다 clear시켜줍니다.
void printLcd(struct SQL_Column *sqlColumn)함수를 통해 날씨 데이터와 DHT22 온습도 데이터를 출력합니다.
void moveMotor(String* dbWindowState)함수로 사용자의 요청 여부에 따라 모터를 움직입니다.

sql.h – 선언부

```
#ifndef __SQL_H__
#define __SQL_H__

#include "defines.h"
#include "Credentials.h"
#include <MySQL_Generic.h>

#define USING_HOST_NAME true
#if USING_HOST_NAME
  extern const char SERVER[] = "서버주소";
#else
  IPAddress server(IP);
#endif

extern const char USER[] = "DB Admin ID";
extern const char PASSWD[] = "DB Admin Password";

extern const uint16_t SERVER_PORT = 3307;
extern const char DATABASE[] = "IoT";
extern const char INSERT_TABLE[] = "Weather";
extern const char SELECT_TABLE[] = "doorState";
```

서버 주소 및 DB 관리자 id와 password는 보안상 공개하지 않습니다.

sql.h파일에서는 DB와 연결하기 위해 서버 주소 및 아이디와 패스워드 및 포트를 설정해줍니다.
포트는 포트포워딩 해 놓은 3307번 포트를 통해 접속합니다.
또한 insert문과 select문에 쓰일 Table Name도 설정해 주었습니다.

sql.h – 구조체 및 함수

```
typedef struct SQL_Column
{
  String strWeather;
  String strFeelsLike;
  String strTempMin;
  String strTempMax;
  String strTemperature;
  String strHumidity;
  String strWind;

  float dhtTemperature;
  float dhtHumidity;
  String strDhtTemperature;
  String strDhtHumidity;

  char chrWeather[10];
  char chrTemperture[20];
  char chrHumidity[10];
  char chrDhtTemperature[10];
  char chrDhtHumidity[10];

}SQL_Column;

void updateInsertSqlQuery(struct SQL_Column* sqlColumn);
void insertDB(struct SQL_Column* sqlColumn);
void getDB_WindowState(String* dbWindowState);

#endif
```

SQL_Column 구조체는 DB에 직접적으로 insert 시킬 변수와 LCD에 출력시킬 변수가 들어 있습니다.
void updateInsertSqlQuery(struct SQL_Column* sqlColumn) 함수는 최종적으로 DB에 insert 되기 전에 sql쿼리문을 최신으로 유지해주는 작업을 실행합니다.
void insertDB(struct SQL_Column* sqlColumn) 함수는 DB Weather table에 날씨 데이터와 DHT22 온습도 데이터를 실질적으로 insert 시키는 작업을 진행합니다.
void getDB_WindowState(String* dbWindowState) 함수는 DB doorState table에서 사용자의 요구사항을 가져옵니다.

sql.ino – updateInsertSqlQuery

```
#include "sql.h"

MySQL_Connection conn((Client *)&client);

String INSERT_SQL;

void updateInsertSqlQuery(struct SQL_Column* sqlColumn)
{
  INSERT_SQL = String("INSERT INTO ") + DATABASE + "." + INSERT_TABLE +
  "(Weather, Temperature, FeelsTemp, MinTemp, MaxTemp, Humidity, Wind, innerTemperature,
innerHumidity) VALUES ('"
  + sqlColumn->strWeather + "','"
  + sqlColumn->strTemperature + "','"
  + sqlColumn->strFeelsLike + "','"
  + sqlColumn->strTempMin + "','"
  + sqlColumn->strTempMax + "','"
  + sqlColumn->strHumidity + "','"
  + sqlColumn->strWind + "','"
  + sqlColumn->strDhtTemperature + "','"
  + sqlColumn->strDhtHumidity
  + "')";
}
```

sql.ino 에서 MariaDB 에 연결하기 위한 connection 을 생성하고 DB insert 에 사용하는 쿼리문을 INSERT_SQL 변수로 만들어 줍니다.

updateInsertSqlQuery(struct SQL_Column* sqlColumn)함수 호출시 INSERT_SQL 변수를 최신 query 문으로 업데이트 시켜줍니다.

sql.ino – insertDB, getDB_WindowState

```
void insertDB(struct SQL_Column* sqlColumn)
{
  if (conn.connectNonBlocking(SERVER, SERVER_PORT,(char*)USER,(char*)PASSWD)!=RESULT_FAIL)
  {
    MySQL_Query query_mem = MySQL_Query(&conn);
    if (conn.connected())
      updateInsertSqlQuery(sqlColumn);
    query_mem.execute(INSERT_SQL.c_str())
    conn.close();
  }
}
void getDB_WindowState(String* dbWindowState)
{
  if (conn.connectNonBlocking(SERVER, SERVER_PORT,(char*)USER,(char*)PASSWD)!=RESULT_FAIL)
  {
    String query = "SELECT * FROM IoT.doorState ORDER BY log DESC LIMIT 1";

    row_values *row = NULL;
    String result;

    MySQL_Query query_mem = MySQL_Query(&conn);
    query_mem.execute(query.c_str())
    query_mem.get_columns();
    row = query_mem.get_next_row();
    result = row->values[1];
    query_mem.close();
    *dbWindowState = result;
    conn.close();
  }
}
```

insertDB 함수에서 DB 연결을 체크하고 updateInsertSqlQuery 함수를 통해 INSERT_SQL 변수를 최신 query 로 업데이트 시킨뒤 query_mem.execute(INSERT_SQL.c_str())로 DB 에 insert 시키고 conn.close()로 연결을 종료합니다.

getDB_WindowState 함수는 insertDB 함수와 마찬가지로 DB 와의 연결을 체크하고 SELECT * FROM IoT.doorState ORDER BY log DESC LIMIT 1; query 를 통해 사용자의 최신 창문 요청상태를 받아옵니다.

weatherAPI.h – 변수 및 함수

```
#ifndef __WEATHER_API_H__
#define __WEATHER_API_H__

#include <ArduinoJson.h>

extern const String APIKEY = "5a828a9d299c48976f658c048017769f";
extern const String CITYID = "1835848"; // Seoul, KR

extern const char SERVER_NAME[]="api.openweathermap.org";

void getWeatherData(struct SQL_Column *sqlColumn);

#endif
```

weatherAPI.h 파일에서는 Open Weather Map 사이트에서 발급받은 키와 날씨를 받아올 지역 및 API 서버 주소와 getWeatherData 함수를 선언하였습니다. getWeatherData 함수는 발급받은 키로 API 에 날씨 데이터를 요청하고 응답 받은 JSON 형식의 데이터를 필요한 부분만 split 하여 sqlColumn 구조체에 값을 넘겨줍니다.

weatherAPI.ino – getWeatherData

```
#include "weatherAPI.h"

void getWeatherData(struct SQL_Column *sqlColumn)
{
  String strResult;

  if (client.connect(SERVER_NAME, 80))
  {
    client.println("GET /data/2.5/weather?id="+CITYID+"&units=metric&APPID="+APIKEY);
    client.println("Host: api.openweathermap.org");
    client.println("User-Agent: ArduinoWiFi/1.1");
    client.println("Connection: close");
    client.println();
  }
```

client connection 을 통해 연결을 체크하고 API 에 날씨데이터를 요청하는 코드입니다.

```
while (client.connected() || client.available())
  {
    char clnt_read = client.read();
    strResult = strResult+clnt_read;
  }
  client.stop();
  strResult.replace('[', ' ');
  strResult.replace(']', ' ');
```

데이터가 끝날 때 까지 위에서 선언한 strResult 버퍼에 clnt_read 의 데이터를 써주고 데이터가 끝나면 연결을 종료합니다. JSON strResult 버퍼에 있는 '[' 와 ']'를 공백문자로 치환해줍니다.

```
char jsonArray [strResult.length()+1];
strResult.toCharArray(jsonArray,sizeof(jsonArray));
jsonArray[strResult.length() + 1] = '\0';

StaticJsonBuffer<1024> json_buf;
JsonObject &root = json_buf.parseObject(jsonArray);
```

strResult 버퍼의 Raw 데이터를 JSON 데이터로 가공합니다.

```
String location = root["name"];
String country = root["sys"]["country"];
float temperature = root["main"]["temp"];
float feels_like = root["main"]["feels_like"];
float temp_min = root["main"]["temp_min"];
float temp_max = root["main"]["temp_max"];
float humidity = root["main"]["humidity"];
float wind = root["wind"]["speed"];
String weather = root["weather"]["main"];
String description = root["weather"]["description"];
float pressure = root["main"]["pressure"];
```

가공된 데이터 중 필요한 데이터를 선별합니다.

```
sqlColumn->strWeather = weather;
sqlColumn->strTemperature = String(temperature);
sqlColumn->strHumidity = String(humidity);
sqlColumn->strWind = String(wind);
sqlColumn->strFeelsLike = String(feels_like);
sqlColumn->strTempMin = String(temp_min);
sqlColumn->strTempMax = String(temp_max);

weather.toCharArray(sqlColumn->chrWeather, weather.length()+1);
sprintf(sqlColumn->chrTemperture, "%.2f", temperature);
sprintf(sqlColumn->chrHumidity, "%.2f", humidity);
```

선별된 데이터를 sqlCloumn 에 맞게 형변환 후 구조체에 넣어줍니다.

8_2. Telegram bot

dbconn.py

```
import sys
import pymysql
import numpy as np

server= 'SERVER URL'
port = int(3307)
user = 'Admin ID'
passwd = 'Admin PASSWORD'
database = 'IoT'
autocommit = False

class DataBase:
    def connectDB():
        try:
            conn = pymysql.connect(user = user, password = passwd, host = host,
                                   port = port, database = database, autocommit = autocommit)
        except pymysql.Error as error:
            print(f"Error connection to MariaDB : {error}")
            sys.exit(1)
        return conn

    def getWeatherData():
        conn = DataBase.connectDB()

            ...
    def insertCloseWindow():
        conn = DataBase.connectDB()
            ...
    def insertOpenWindow():
        conn = DataBase.connectDB()
            ...
```

DataBase 기능을 하는 함수를 Class 로 묶어서 보다 직관적으로 사용할 수 있게 하였습니다.
connectDB()에서는 conn 객체를 반환하며 getWeatherData(), insertCloseWindow(), def insertOpenWindow(), 함수에서 DB 연결에 사용됩니다.

telegram_command.py

```
import telegram
from telegram.ext import Updater, MessageHandler, Filters, CommandHandler
from threading import Thread
from datetime import datetime
from dbconn import *

access = "Telegram Access KEY"
secret = "Telegram Secret KEY"
myToken = "Telegram Token"
bot_token = 'Telegram bot Token'
bot = telegram.Bot(token=bot_token)

chat_id = 'Chatting room id'

#명령어 정의
def help_command(update, context) :
    bot.sendMessage(chat_id=chat_id, text='아침 7 시마다 날씨를 자동으로 알려드립니다.\n\n'
                    +'명령어\n'
                    +'\'날씨\' 를 입력하시면 현재 날씨를 알려드립니다.\n'
                    +'\'창문닫아줘\' 를 입력하시면 창문을 닫습니다.\n'
                    +'\'창문열어줘\' 를 입력하시면 창문을 엽니다.\n\n'
                    +'추가적으로 비가 오면 자동으로 비 알람문자를 보내드립니다.\n')

# message reply function
def get_message(update, context) :
    getMsg = update.message.text
    try:
        if getMsg[0:] == '창문닫아줘':
            DataBase.insertCloseWindow()
            update.message.reply_text("창문을 닫습니다.")

        elif getMsg[0:] == "창문열어줘":
            DataBase.insertOpenWindow()
            update.message.reply_text("창문을 엽니다.")
        . . .
```

```
#새 메시지 확인
updater = Updater(bot_token, use_context=True)


# 메세지중에서 command 제외
message_handler = MessageHandler(Filters.text & (~Filters.command), get_message)
updater.dispatcher.add_handler(message_handler)


# 응답 커맨드 정의
help_handler = CommandHandler('help', help_command)
updater.dispatcher.add_handler(help_handler)

updater.start_polling(timeout=1, clean=True)
updater.idle()
```

telegram_command.py 에서는 사용자가 bot 에게 보낸 명령어를 판별하고 그에 따라 ESP8266 으로 창문을 열거나, 닫거나, 날씨를 알려주는 기능을 수행합니다.

telegram_msg.py

```
def morning_message():
    data = DataBase.getWeatherData()
    ...
    bot.sendMessage(chat_id=chat_id, text=f'{date}\n좋은 아침이에요!\n현재 날씨는
{cloud}\n기온은 {data[2]}도 이고 습도는 {data[6]}% 입니다.\n'+
                    f'{feels}\n\n현재 방 내부 온도는 {data[8]}도 이며 습도는 {data[9]}% 입니다.')

def rainCheck():
    rainData = DataBase.getWeatherData()
    ...
    if(rainCompare == "Rain"):
        if(isRain != rainCompare):
            isRain = rainCompare
            bot.sendMessage(chat_id=chat_id, text=f'비 안내 문자입니다.\n현재 비가와요 우산을
                                                                    챙기세요!')
    else:
        isRain = rainCompare
schedule.every().day.at("07:00").do(morning_message)
schedule.every(5).seconds.do(rainCheck)
```

```
while True:
    try:
        schedule.run_pending()
    except Exception as error:
        print(error)
        bot.sendMessage(chat_id='chat_id', text=f'Code Error! \n {str(error)}')
        pass
    time.sleep(1)
```

telegram_msg.py 에서는 스케줄러 기능을 사용해 아침 7 시마다 자동으로 날씨를 알려주며 비가 갑자기 내릴 시 비 알람 문자를 전송해 줍니다.

8_3. PHP Web page

```
<?php
session_start();
$conn = mysqli_connect("DBServerIP", "Admin ID", "Admin Password", "IoT", 3307);
$sql = "SELECT * FROM Weather ORDER BY num DESC LIMIT 1;";
$result = mysqli_query($conn, $sql);
$row = mysqli_fetch_array($result);
$Weather = $row['Weather'];
$Temperature = $row['Temperature'];
$FeelsTemp = $row['FeelsTemp'];
$MinTemp = $row['MinTemp'];
$MaxTemp = $row['MaxTemp'];
$Humidity = $row['Humidity'];
$Wind = $row['Wind'];
$innerTemperature = $row['innerTemperature'];
$innerHumidity = $row['innerHumidity'];
?>
```

php 에서도 마찬가지로 날씨와 센서 데이터를 웹에 띄워주기 위해 DB 와 연결 후 최신 데이터를 불러옵니다.

```
        생략
<div id="secondLine">
        <div id="temperature" class="secondState">
            <!--외부 온도 나오는 부분-->
            외부 온도
            <br>
            <p style="color:#32a1ff"><?=$Temperature?>℃</p>
        </div>
        <div id="minTemp" class="secondState">
            <!--최저 기온 나오는 부분-->
            최저 온도
            <br>
            <p style="color:#32a1ff"><?=$MinTemp?>℃</p>
        </div>
        <div id="maxTemp" class="secondState">
            <!--최고 기온 나오는 부분-->
            최고 온도
            <br>
            <p style="color:#32a1ff"><?=$MaxTemp?>℃</p>
        </div>
        생략
```

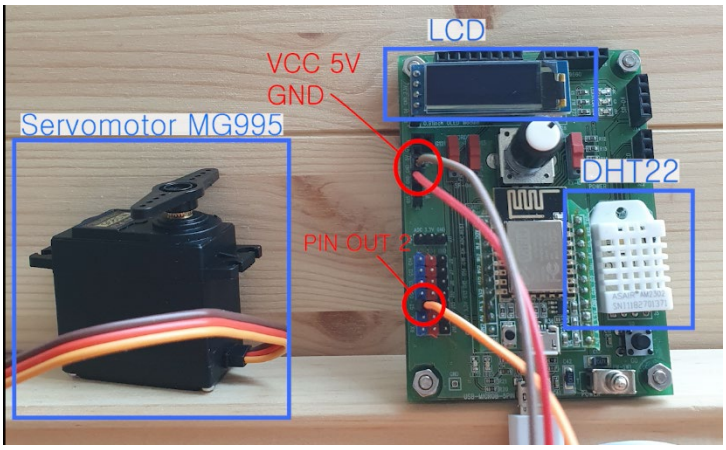
php 로 받아온 데이터를 HTML 을 통해 실제 화면에 띄워주는 역할을 하게 됩니다.

위의 원본 코드들은 GitHub 를 통해 확인할 수 있습니다.

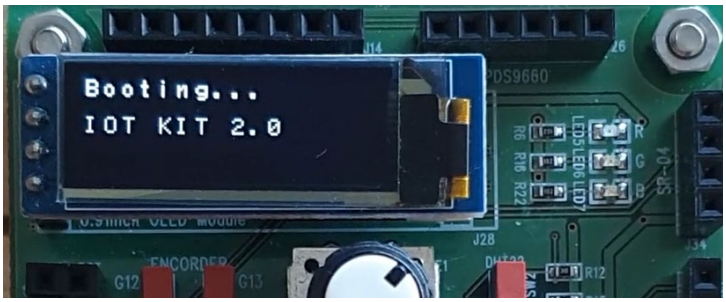
https://github.com/Backtrack404/ESP8266-IoT-TermProject

Ⅲ. 결론

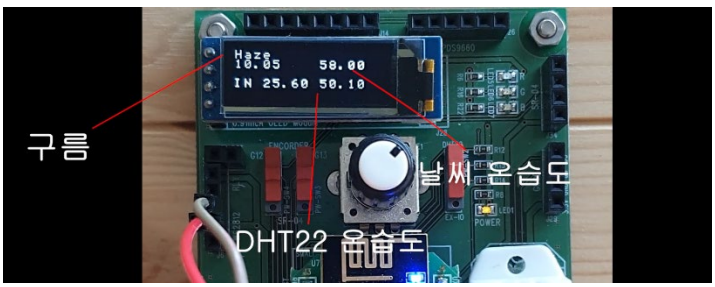
1. 최종동작



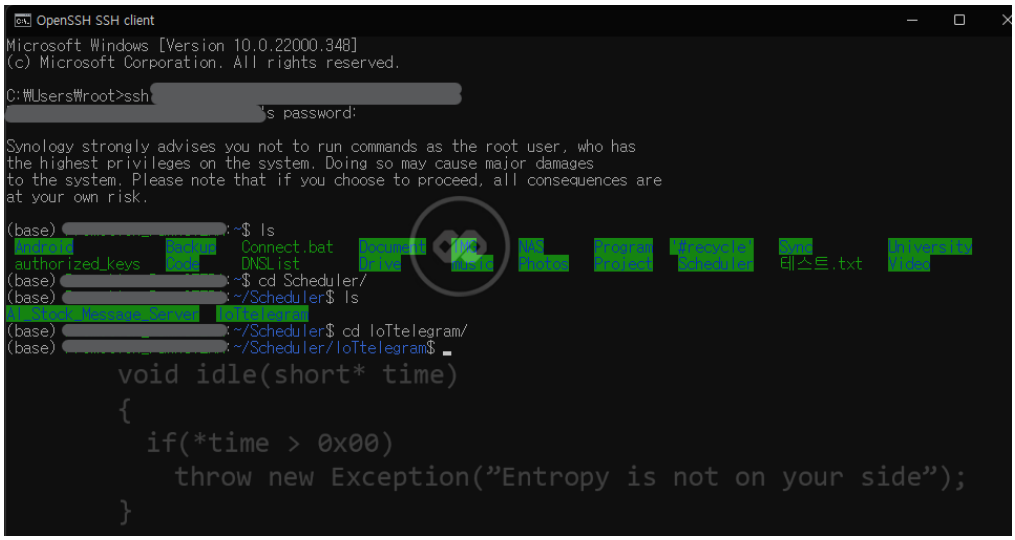
하드웨어는 하드웨어 구성도와 같이 서보모터는 5V VCC 와 GND, 2 번핀에 연결하였습니다.



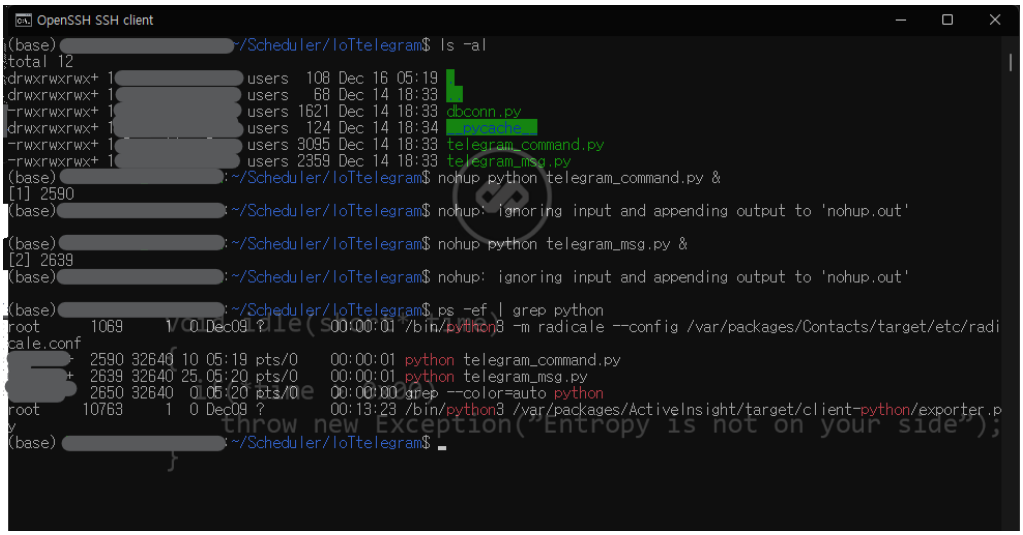
장비를 켜게 되면 와이파이를 잡을 때까지 Booting 문구가 LCD 에 표시됩니다.



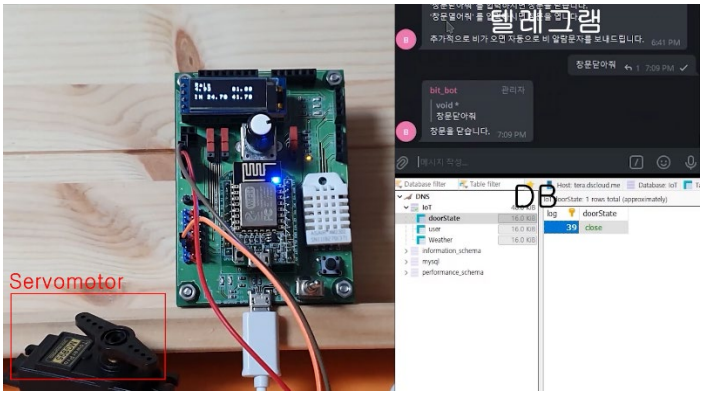
부팅이 완료되면 날씨 API 에서 받아온 데이터와 DHT22 온습도 데이터를 LCD 에 표시합니다.



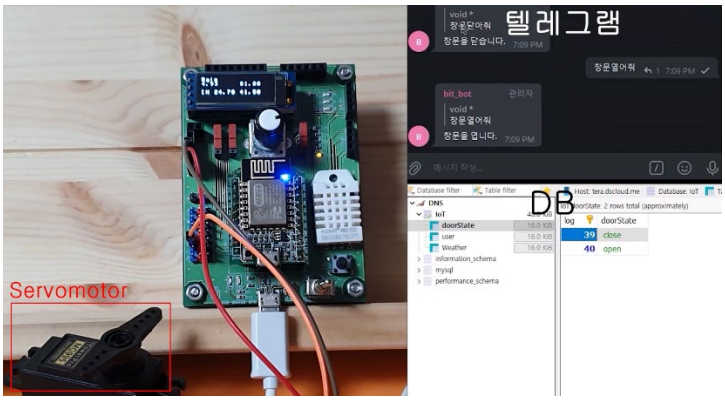
프로그래밍한 Telegram Bot 을 활성화시키기 위해 ssh 로 서버에 접속합니다.



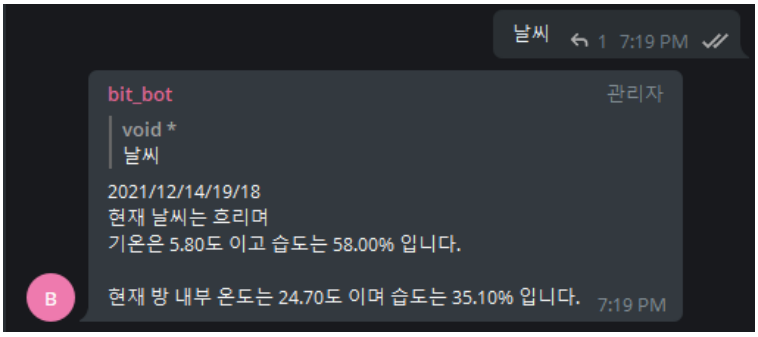
nohup 과 & 옵션을 통해 telegram_command.py 와 telegram_msg.py 를 백그라운드에서 데몬 프로세스로 만들어 줍니다.



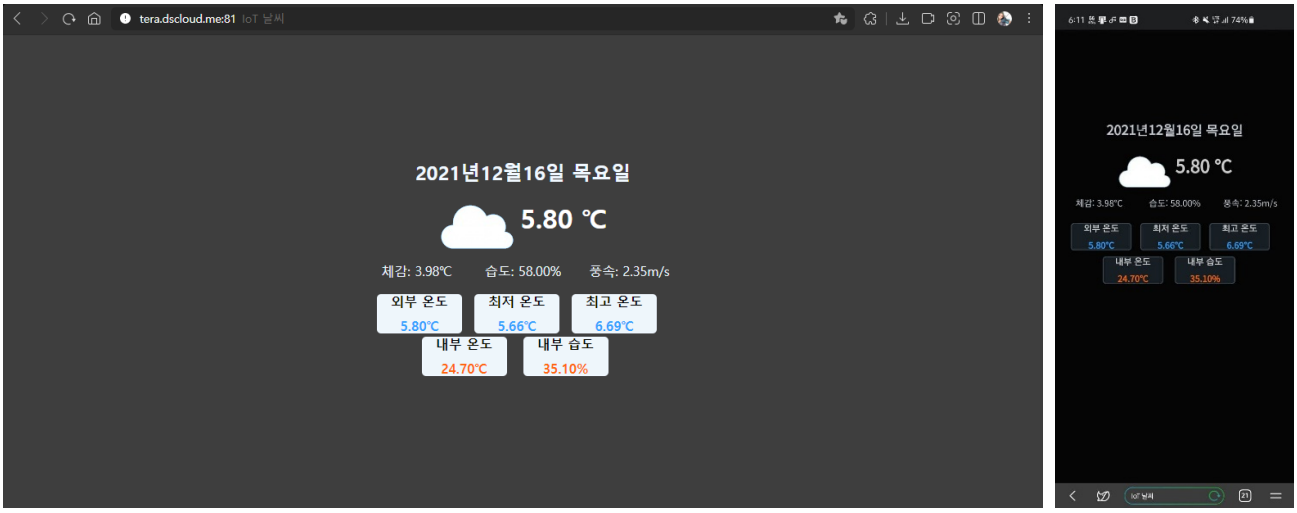
사용자가 창문을 닫아 달라는 문자를 보내게 되면 DB 에 close 가 올라가고 ESP8266 이 이를 읽어 서보모터를 움직입니다.



창문을 여는 것도 마찬가지로 사용자가 창문을 열어 달라는 문자를 보내게 되면 DB 에 open 이 올라가게 되고 ESP8266 에서 읽어 현재 상태와 DB 상태를 판별 후 다르다면 서보모터를 움직입니다.



날씨 명령어를 통해서 날씨 및 방 내부 온습도 확인이 가능합니다.



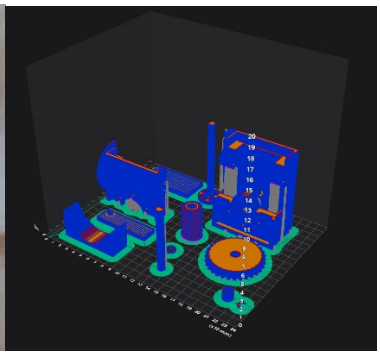
날씨 및 DHT22 온습도 데이터는 포트포워딩한 81 번 포트를 통해 PC 웹과 모바일 웹으로도 볼 수 있습니다.

2. 추가사항

모터 및 기어 3D 프린팅

모터 2 개로 창문을 닫으려고 시도하다 모터 기어가 부러져 결국 하나밖에 사용할 수 없게 되었습니다.





텀프로젝트가 원래 생각했던 것만큼의 결과가 나오지 않아 ESP32 를 구매하였고 프로젝트가 끝나더라도 제작중인 3D 프린터 모델을 사용해 원격으로 창문을 닫아볼 예정입니다.
3D 프린터는 웜기어 모델링을 진행 중입니다.

장치 등록 문제와 DB 트래픽 오버

현 프로젝트의 문제는 장치를 어떻게 등록할 것 인가가 문제입니다. 따라서 웹 서버에서 회원가입을 받고 ESP8266 WiFi 웹을 통해 로그인 해 장치 등록 문제를 해결할 수 있을 것으로 보입니다. 또한 현재 로직은 DB 에 과도한 트래픽이 몰리는 구조로 되어 있습니다. 이를 해결하기 위해 MQTT 를 사용하여 각기 다른 플랫폼에서 데이터를 공유할 수 있는 방법에 대해 조금 더 알아봐야 할 것 같습니다.

인공지능

DB 에 데이터가 축적되고 사용자가 언제 창문을 열고 닫는지를 학습시키면 현재 telegram bot 처럼 수동적이 아닌 능동적으로 자동화까지 가능할 것으로 보입니다. 과제 제출 후 계속 연구해 볼 생각입니다.

Ⅳ. 참고문헌

ArduinoJson
https://github.com/bblanchon/ArduinoJson

CopyThread
https://github.com/jensh/CopyThreads

MySQL_MariaDB_Generic
https://github.com/khoih-prog/MySQL_MariaDB_Generic

Task Scheduler
https://github.com/arkhipenko/TaskScheduler

감속기의 원리와 종류
https://careeman.tistory.com/100

웜기어
https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=motor_bank&logNo=220314967238

기어 공식
https://engineershelp.tistory.com/373

PHP MQTT
https://github.com/CloudMQTT/php-mqtt-example