

한 수 앞선 Project (Hansu Bot) (RaspberryPi)

※ 과제 목적

기술이 빠르게 고도화되면서 새로운 기술을 만들어내는 수준이 최고점에 도달했다. 이로 인해 신기술을 활용한 시장의 기술 독점은 어려워졌고, 기술이 곧 경쟁력이었던 과거의 사회에서 멀어지고 있다. 지금의 경쟁력은 친환경적이고 인간공학적 기술의 활용에 있다. 기술의 수준보다 얼마나 인간에게 도움을 주고 환경문제 개선에 참여하는지를 보기 시작한 것이다.

‘한 수 앞선’도 기술을 활용하여 환경문제 개선에 참여하고자 ‘라즈베리파이와 머신러닝을 활용한 수질 데이터 수집 및 분석’ 과제를 계획하였다. 이는 물속에 포함된 수질 데이터를 측정하여 수집하고, 수집된 데이터를 바탕으로 수질 문제의 정도를 파악하여 심각한 문제를 미리 방지하는 것에 목적을 두고 있다.

수질은 변화에 민감해서 꾸준히 측정되고 분석되어 현재 상태를 파악하고, 위험한 경우 조치를 해야 한다. 과제는 더 나아가 이전까지 기록된 수질 데이터를 바탕으로 날짜와 시간에 따라 변하는 수질 데이터의 흐름을 판단하고 각각의 수질 데이터가 서로 어떤 영향을 미치는지 상관관계를 분석한다. 또한, 다가오는 수질 정도를 예측할 수 있는 모델을 개발한다.

수질 등급을 평가하기 위해서는 물속에 포함된 다양한 이온들의 수치와 그들의 상호작용이 어떻게 이루어지는지 자세하게 분석할 필요가 있다. 과제를 통해 4가지 항목(pH, DO, 수온, 탁도)의 상호연관과 흐름을 분석하여 각각의 값의 변화가 물에 어떤 영향을 주는지 해석한다. 이를 통해 수질의 상태를 예측할 수 있으며 이는 이후에 발생할 위험을 미리 판단할 수 있게 하여 수질 악화의 정도를 줄일 수 있음을 의미한다.

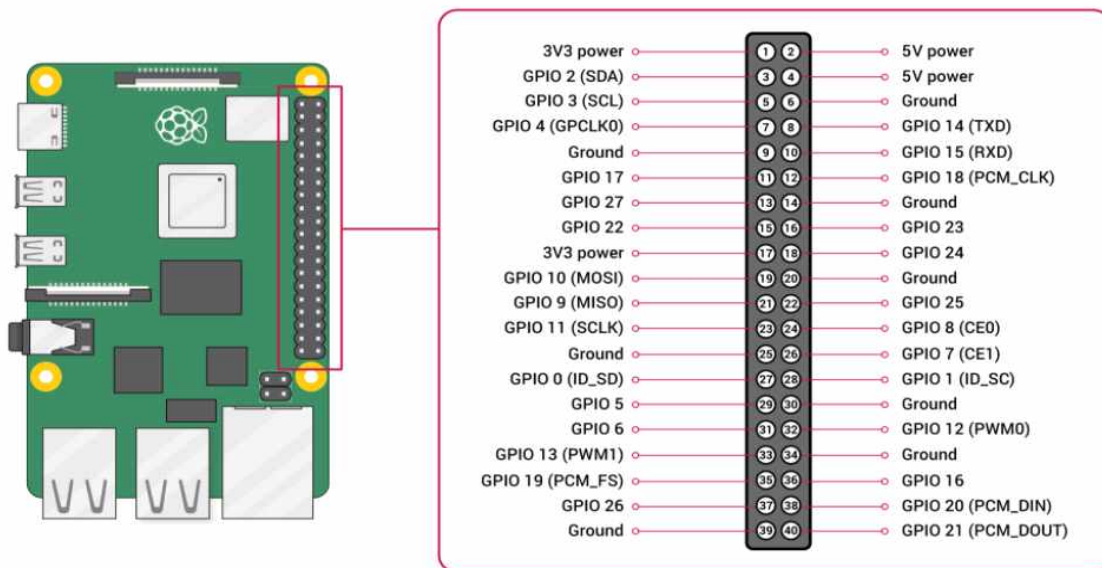
※ 과제 수행 내용

1. 프로젝트 하드웨어 사양

▶ RaspberryPi 4 Model B (8GB)

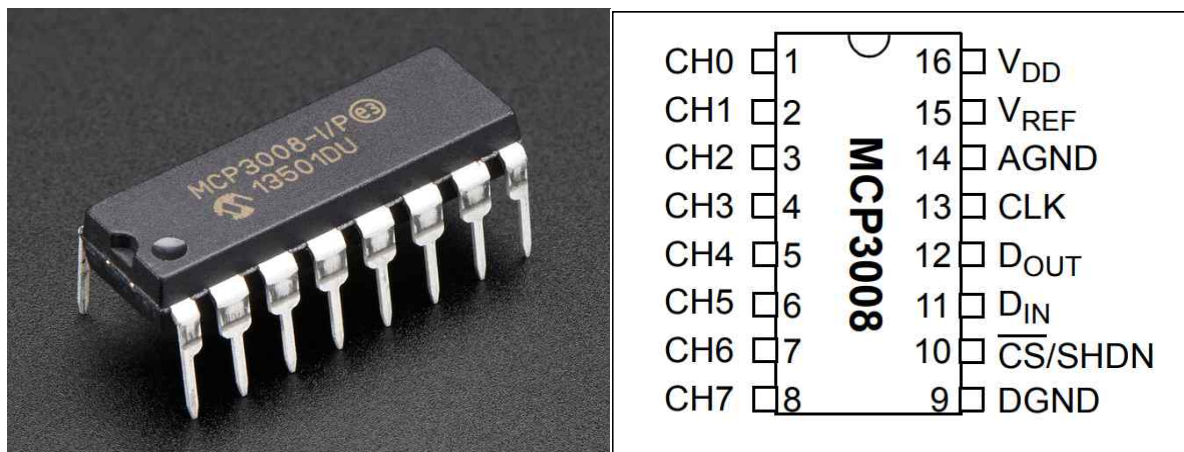


Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	8GB LPDDR4-3200 SDRAM
2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet	◎ 2 USB 3.0 ports ◎ 2 USB 2.0 ports.
Micro-SD card slot for loading operating system and data storage	◎ 5V DC via USB-C connector (minimum 3A*) ◎ 5V DC via GPIO header (minimum 3A*)
Operating temperature: 0 – 50 degrees C ambient	Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)



RaspberryPi 4 GPIO Header

► ADC (Analog Digital Converter)



MCP3008

10-bit resolution (해상도 10 bit)	8 (MCP3008) input channels
SPI serial interface (modes 0,0 and 1,1)	◎ 200 ksps max. sampling rate at VDD = 5V ◎ 75 ksps max. sampling rate at VDD = 2.7V
Single supply Voltage: 2.7V - 5.5V	Industrial temp range: -40°C to +85°C

VDD, VREF = 3.3V	AGND, DGND = GND
CLK = SPI SCLK	DOUT = SPI MISO
DIN = SPI MOSI	CS/SHDN = SPI CE0 or SPI CE1

※ 관련 문서 참조

There are only two states for digital signals: ON/HIGH or OFF/LOW (1 or 0). If one assumes a maximum digital input voltage of 5V, any value between 0V and 0.4V would be recognized as a LOW level and a value between 2.4V and 5V as a HIGH level. All values in between are undefined. If you apply such a voltage, the signal would be randomly recognized as 0 or 1. However, since this is not consistent, a digital signal should never be in this grey area.

In contrast to digital signals, an analog signal can also take intermediate values, so it is not clearly defined. For example, percentage values are given (value ÷ applied voltage), which many analog sensors use.

To read this voltage on the Raspberry Pi, an analog-digital converter like the MCP3008 must be used. However, this does not specify values in volts, but a number between 0 and 1023, which corresponds to 10 bits (2^{10}). The voltage can be determined as follows:

$$(\text{ADC Value} \div 1023) * \text{Voltage}$$

If the analog sensor is operated with a voltage of 3.3V and a value of 789 has been read out, the applied voltage corresponds to 2.54 volts.

▶ pH 측정 센서 Gravity: Analog pH Sensor / Meter Pro for Arduino (SEN0169)



Analog pH sensor SEN0169

https://wiki.dfrobot.com/Analog_pH_Meter_Pro_SKU_SEN0169

Supply Voltage: 5V	Module Size: 43mm x 32mm
Measuring Range: 0 - 14 PH	Temperature Range: 0 – 60 °C
Accuracy: ± 0.1 pH (25°C)	PH2.0 Interface, BNC Connector
Gain Adjustment Potentiometer	Response Time: <=1min

▶ **아날로그 산소 센서 Gravity: Analog Dissolved Oxygen Sensor / Meter Kit for Arduino**

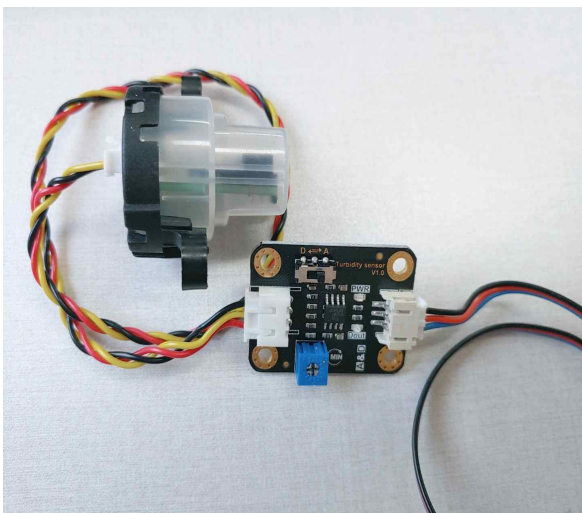


Analog Oxygen Sensor SEN0237

https://wiki.dfrobot.com/Gravity_-_Analog_Dissolved_Oxygen_Sensor_SKU_SEN0237

Supply Voltage: 3.3 - 5.5V	Module Size: 42mm x 32mm
Output Signal: 0 - 3.0V	Temperature Range: 0 - 40 °C
Measuring Range: 0 - 20 mg/L	Response Time: 98% within 90 seconds (25°C)

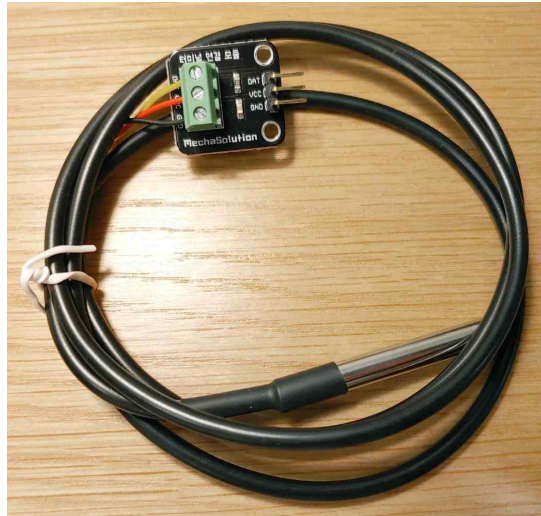
▶ **탁도 센서 (SEN0189)**



Supply Voltage: 5V	Module Size: 38mm x 28mm x 10mm
Analog Output: 0 - 4.5V	Digital Output: High/Low level signal

https://wiki.dfrobot.com/Turbidity_sensor_SKU_-_SEN0189

▶ 수온 센서 (DS18B20)



https://www.mechasolution.com/shop/goods/goods_view.php?goodsno=1864&category=

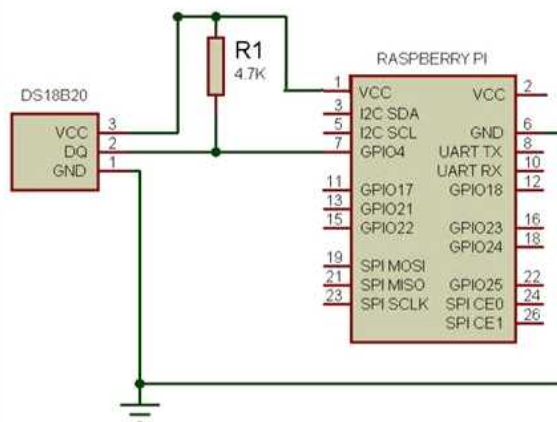
Supply Voltage: 3 – 5.5V	1 Wire Interface
Temperature Range: -55 – 125 °C	4.7K Ω Resistance

※ 관련 문서 참조

1-Wire protocol을 사용하는 DS18B20 sensor는 bus에 병렬로 설치된다. 모든 sensor는 동일 data pin을 사용한다. data line은 Raspberry Pi의 pin 7(BCM port 4)에 연결하도록 한다.

data line에 pull-up 저항을 사용하는데, 4.7K ~ 10K ohm 저항을 사용한다. 여러 개의 sensor가 동시에 설치되는 경우에도 pull-up 저항은 하나만 설치하면 된다.

다음은 Raspberry Pi의 pin 1에서 공급되는 3.3V전원으로 sensor에 별도의 외부 전원을 공급하는 방식으로 회로를 구성한 형태이다.



출처: http://www.realomega.com/publish/raspberry-pi_kor_25-8-2/

▶ GPS 센서 (NEO-6M)



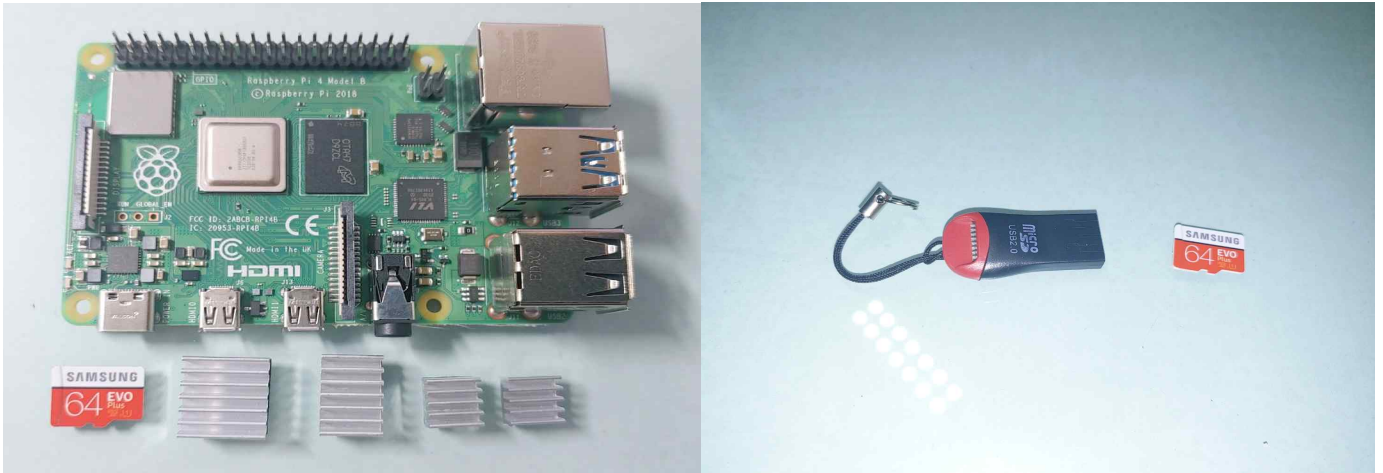
https://eduino.kr/product/detail.html?product_no=261

Supply Voltage: 3.3 – 5V	Module Size: 35 x 26 x 3mm Antena Size: 50 x 25mm
UART Interface	Baud Rate: 9600bps

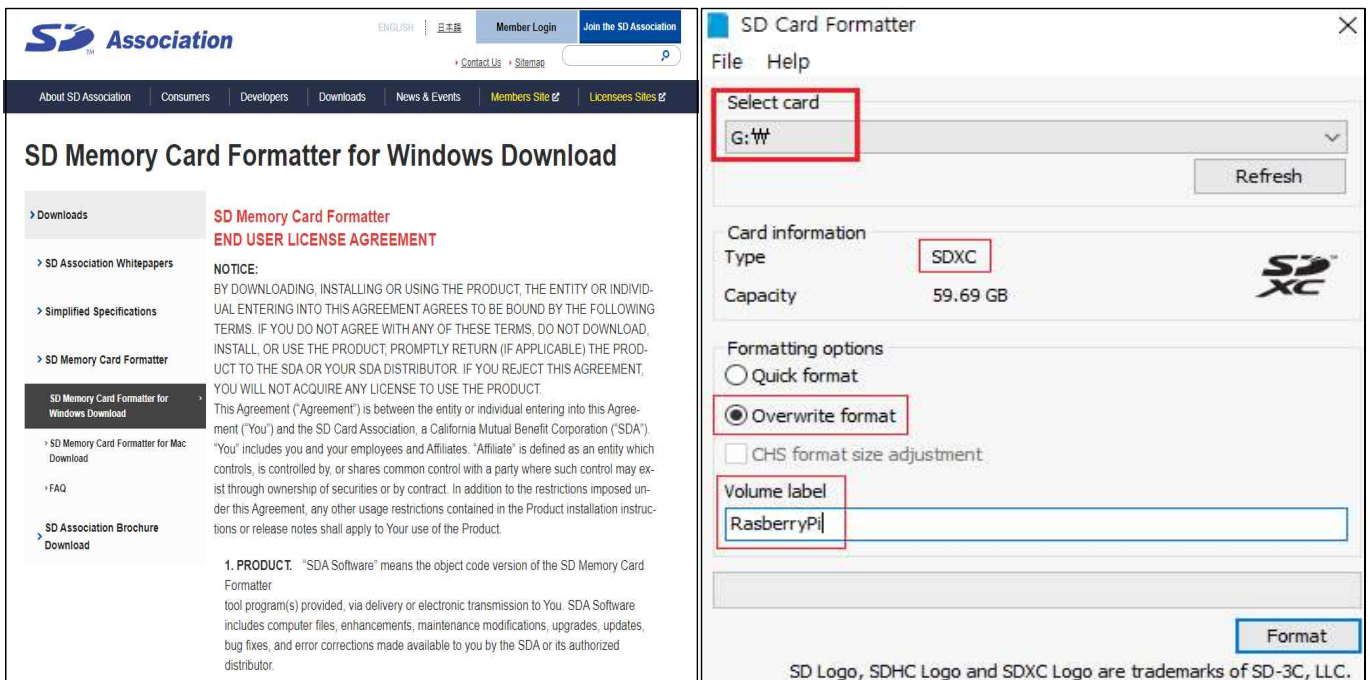
2. 프로젝트 과제 수행 과정

2-1. 라즈베리파이 수질 측정 시스템 개발

▶ 라즈베리파이 4 Model B 운영체제 설치



Raspberry Pi 4 Model B, SD card, SD card Reader 준비



SD Card Formatter 설치 및 Formatting

<https://www.sdcard.org/downloads/formatter/sd-memory-card-formatter-for-windows-download/>

SD Card Formatter 프로그램을 사용하여 SD card 포맷을 진행한다. 'Select card'에서 포맷한 SD card를 선택한다. 메모리의 용량이나 성능에 따라서 'Card information'의 'Type' 값이 'SDHC' 또는 'SDXC'로 표시된다. 해당 두 Type의 메모리를 사용하는 것을 권장한다. 확실한 포맷을 위해 'Formatting options'에서 'Overwrite format'을 선택한다. 'Volume label'은 포맷 후 메모리의 이름을 지정하는 것이며 라즈베리파이 운영체제로 사용할 것이기 때문에 'RaspberryPi'로 작명한다.

Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. [Watch our 45-second video](#) to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Windows](#)

[Download for macOS](#)

[Download for Ubuntu for x86](#)

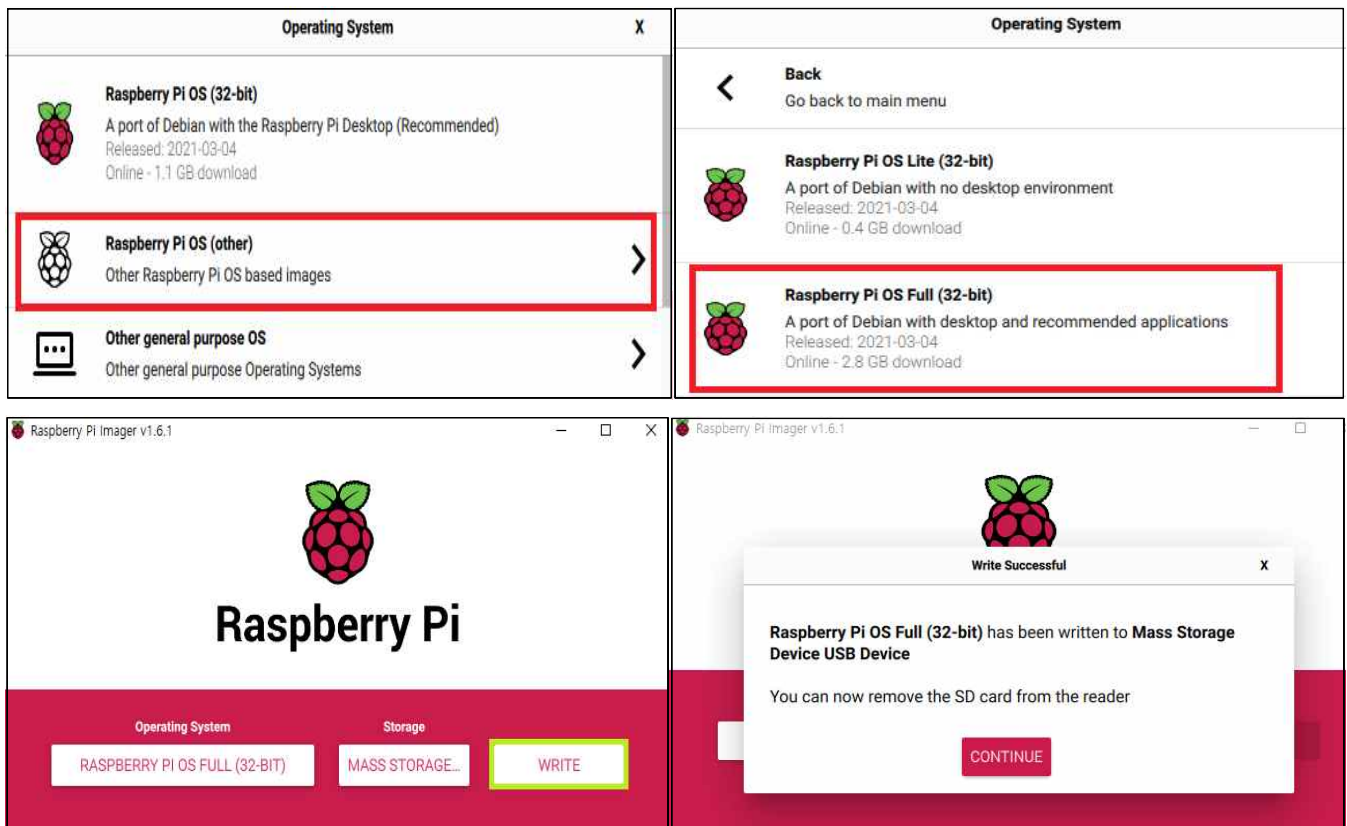
To install on **Raspberry Pi OS**, type `sudo apt install rpi-imager` in a Terminal window.



Raspberry Pi Imager Download

<https://www.raspberrypi.org/software/>

포맷한 SD card에 라즈베리파이 운영체제를 설치하기 위해 라즈베리파이 운영체제 설치 프로그램인 'Raspberry Pi Imager'를 설치한다.



'Operating System'에서 Raspberry Pi OS (other) -> Raspberry Pi OS Full (32-bit) 버전을 선택한다. 'Storage'에서 SD card를 선택한 다음 'WRITE'를 진행한다. 완료 후에는 'CONTINUE' 선택 후 SD card를 안전하게 제거한다. 라즈베리파이 운영체제 설치를 완료한다.

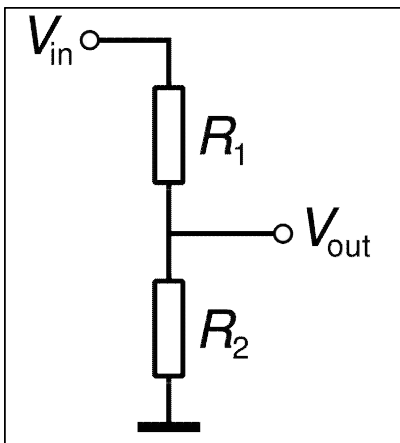
▶ 아날로그 센서 모듈 회로 구성

수질 데이터 수집에 사용할 수소이온농도(pH), 용존산소량(DO), 탁도 센서는 아날로그 데이터를 취급한다. 라즈베리파이는 아두이노와는 다르게 아날로그 센서를 직접 다룰 수 없다. 따라서 아날로그 데이터를 디지털 데이터로 변환시켜 줄 ADC(Analog-Digital Converter)인 '**MCP3008**'을 사용한다.

MCP3008은 'SPI (Serial Peripheral Interface)'로 통신해서 라즈베리파이의 MOSI, MISO, SCK, SS 핀을 사용한다.

MOSI (Master Out, Slave In), SDI, DI, SI : 마스터에서 데이터 출력, 슬레이브에서 데이터 입력
MISO (Master In, Slave Out), SDO, DO, SO : 마스터에서 데이터 입력, 슬레이브에서 데이터 출력
SCLK (Serial Clock), SCK, CLK : 동기화 신호 (CLOCK 신호 통신)
SS (Slave Select), **CE (Chip Enable)**, nCS, nSS, STE : 슬레이브 장치를 선택

아날로그 센서 모듈 중 pH와 탁도 센서는 데이터를 전달하는 출력 신호로 3.3V 이상의 전압을 사용한다. 라즈베리파이는 수용 가능한 최대 전압이 3.3V이기 때문에 이상의 전압이 취급되면 장비의 오작동과 손상이 발생할 수 있다. 따라서 pH와 탁도 센서를 연결하기 위해 '**전압 분배 법칙**'을 적용한다.



전압 분배 법칙

V_{in} = 입력으로 들어오는 전압 (센서 모듈이 데이터 전달로 사용)

V_{out} = 최종 출력으로 나가는 전압 (MCP3008, 라즈베리파이로 전달)

R_1 R_2 = 저항. R_2 는 R_1 의 2배에 해당하는 저항값을 가진다.

$$V_{out} = R_2 / (R_1 + R_2) \times V_{in}$$

센서 모듈에서 데이터 전달에 사용하는 최대 전압이 5V라고 했을 때, $R_1=1k\Omega$, $R_2=2k\Omega$ 로 저항을 두면 V_{out} 의 최종 전압은 $2/(1+2) \times 5 = 10/3 = 3.33\dots$ 이 된다.

수온 센서(DS18B20)는 라즈베리파이에서 '1-wire' 통신에 적합한 모듈이다. 1-wire 통신은 신호에 사용하는 선으로 전원 공급까지 가능하지만 풀업 저항을 요구한다. 보편적으로 사용하는 $4.7k\Omega$ 저항을 사용하여 연결한다.

GPS 센서(NEO-6M)는 시리얼 통신(UART)을 사용한다. 라즈베리파이에서 UART를 지원하는 핀인 GPIO14(TX), GPIO15(RX)를 사용하여 연결한다.

※ 라즈베리파이 센서 회로도

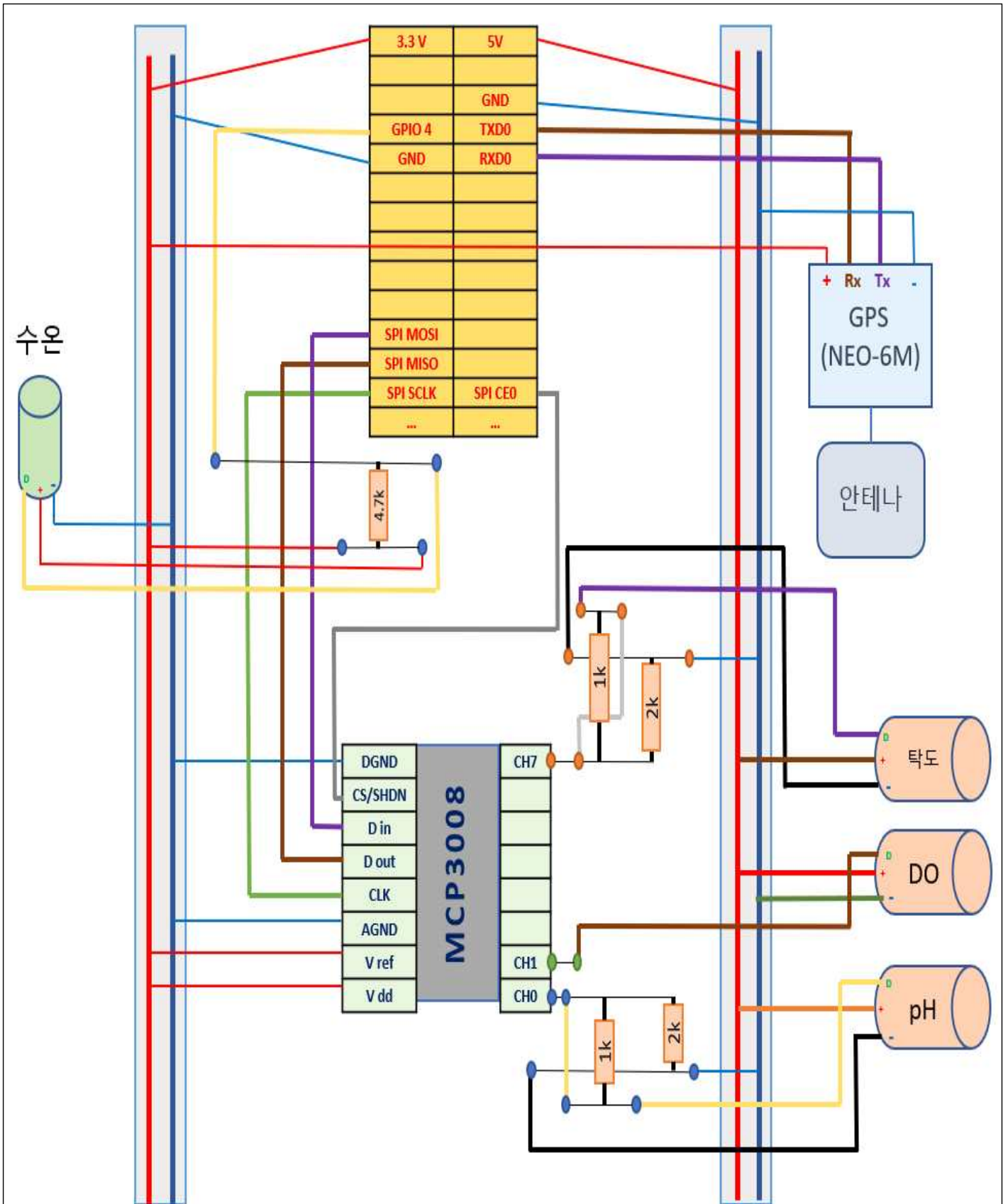


그림 20. 라즈베리파이 센서 구성 회로도

※ 라즈베리파이 센서 회로도 (구성 이미지)

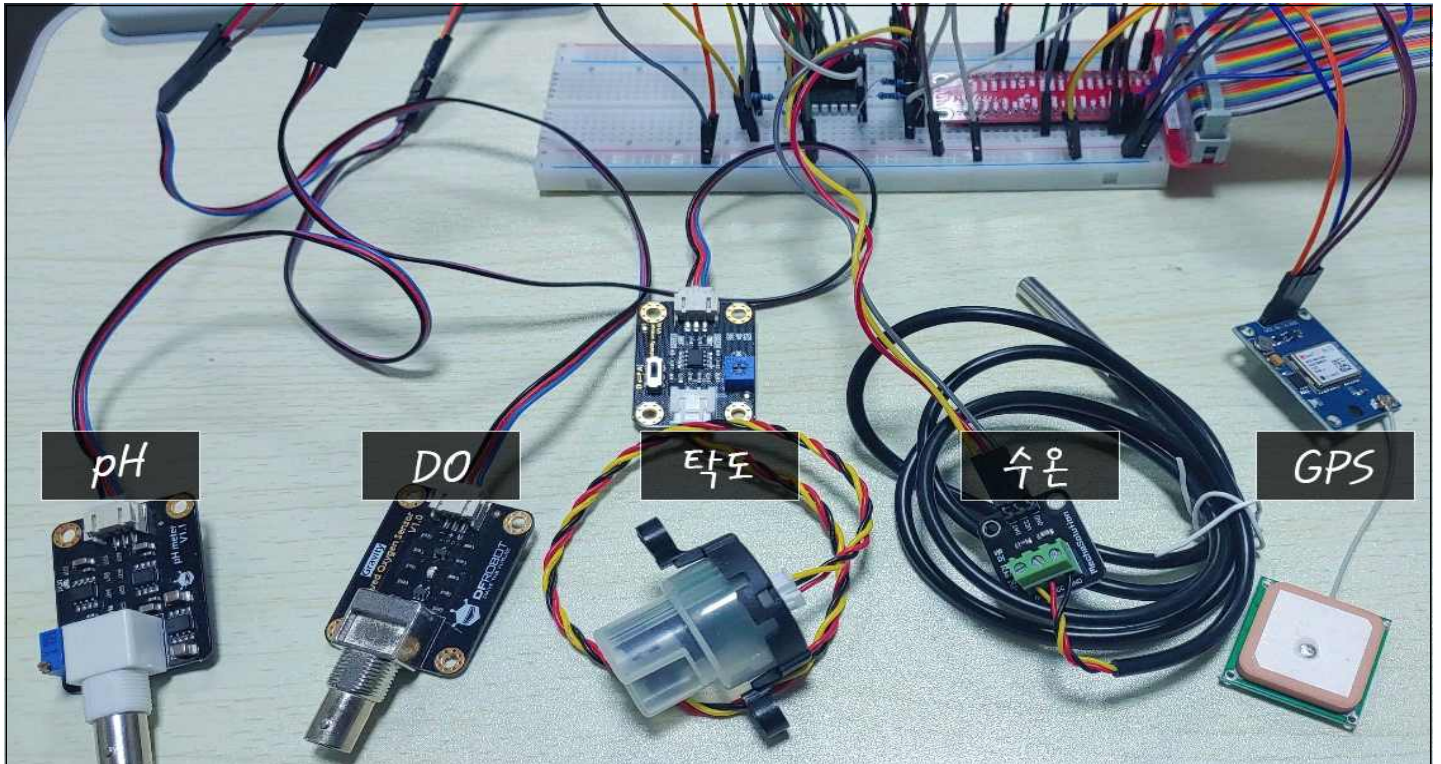


그림 21 데이터 수집 센서

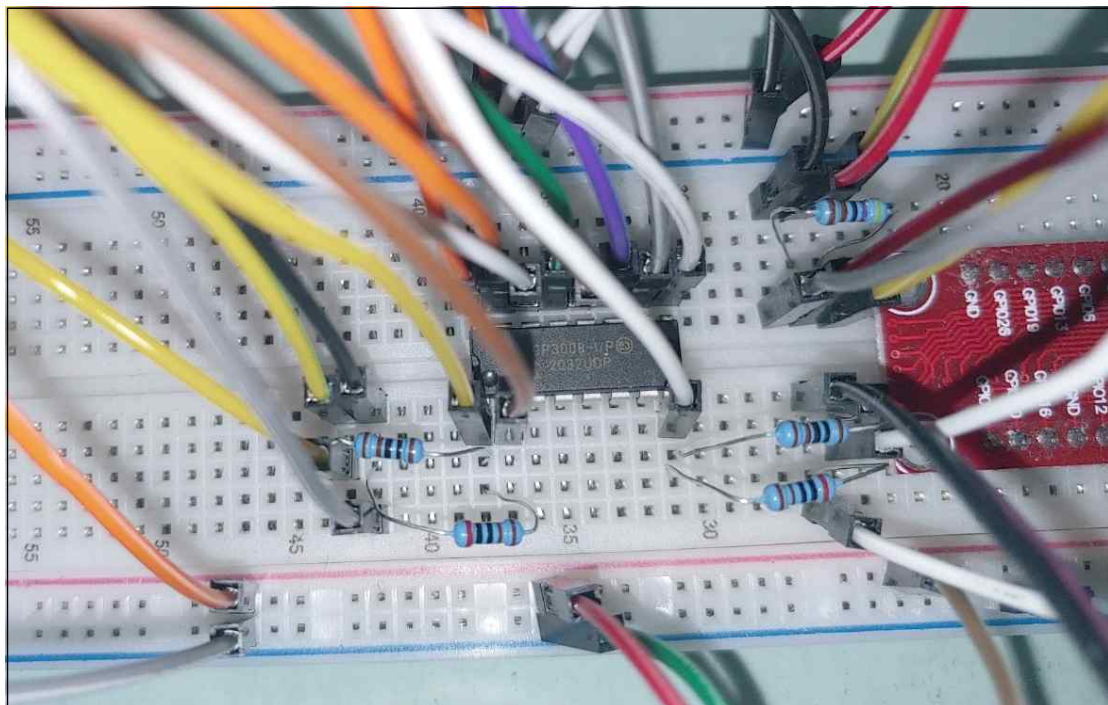


그림 22 MCP3008과 센서와 연결한 저항

▶ 수온 센서 연결 및 코드 구현

① 수온 센서 회로 연결

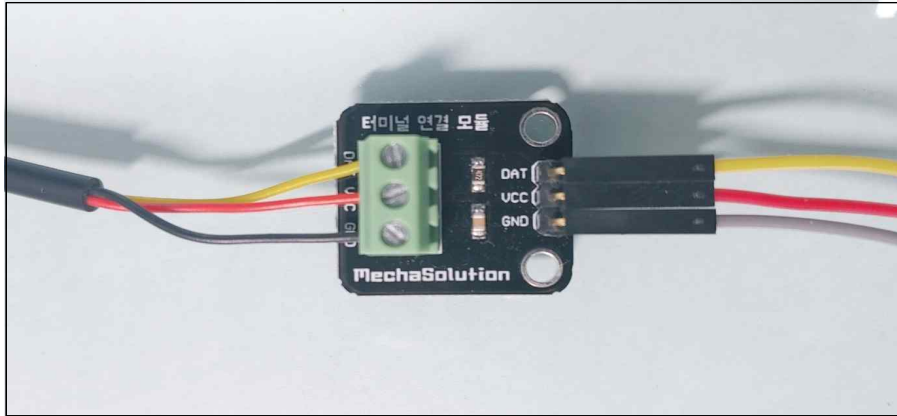


그림 23 수온 센서 연결 모듈

수온 센서는 3v 이상부터 사용할 수 있어서 VCC는 3.3v와 연결한다. GND는 동일한 GND와 연결하며 DAT는 GPIO4 핀과 연결한다. 수온 센서는 1-wire 통신을 사용하기 때문에 한 개의 핀으로 통신할 수 있다. DAT와 연결된 GPIO4를 통해서 데이터의 입/출력이 발생하기 때문에 신호의 변화를 주어야 하며 풀업 저항을 통해 가능하다. 4.7 ~ 10kΩ을 사용할 수 있으며 많은 예제에서 사용된 4.7Ω을 사용한다.

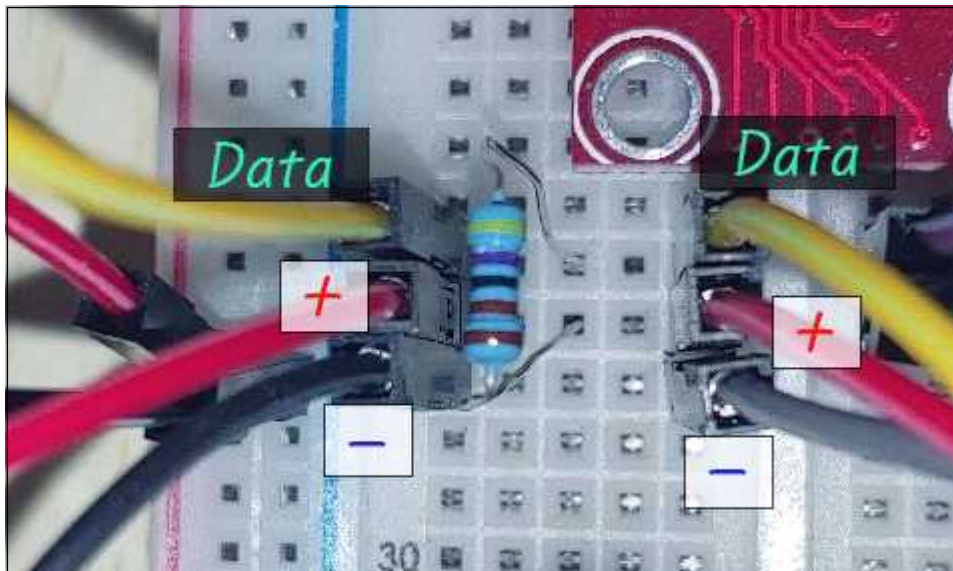


그림 24 센서 케이블(우측)과 라즈베리파이 핀(좌측) 사이에 저항(4.7k) 연결

회로 연결 후에는 라즈베리파이에 1-wire 통신을 가능하도록 설정하고, 코드를 작성하여 수온 센서로부터 데이터가 정상적으로 들어오는지 확인한다. 먼저 라즈베리파이의 'Raspberry Pi Configuration'에서 'Interfaces' 탭에 들어간 다음 1-Wire 설정을 Enable로 한다.

② 라즈베리파이 1-wire 인터페이스 연결 설정

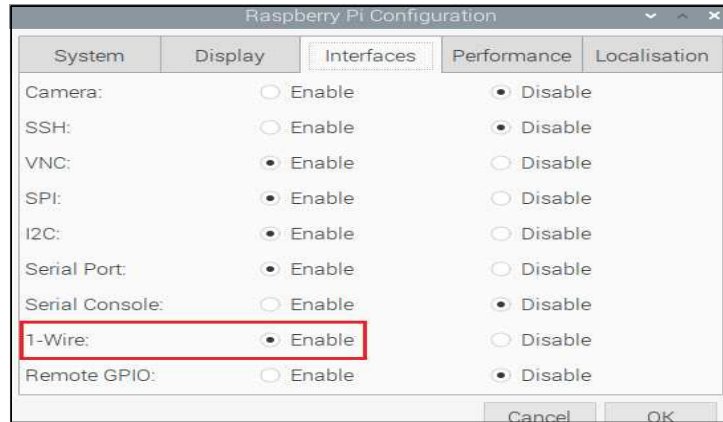


그림 25 1-Wire Enable 설정

센서에서 측정되는 데이터는 외부 인터페이스를 통해서 라즈베리파이로 들어오기 때문에 디바이스 트리 오버레이를 통해서 인터페이스 장치를 연결해야 한다. 하드웨어와 관련된 부분을 제어하는 파일은 **/boot/config.txt**에 존재하며 해당 파일을 수정하여 1-wire 인터페이스를 연결한다.

sudo nano /boot/config.txt
맨 아래의 [all] 항목에 dtoverlay=w1-gpio 작성 후 저장
cat /boot/config.txt 저장된 내용 확인

```
[pi4]
# Enable DRM VC4 V3D driver on top of the dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2

[all]
#dtoverlay=vc4-fkms-v3d
dtoverlay=w1-gpio
dtoverlay=uart5
```

그림 26 /boot/config.txt 파일에 1-wire 인터페이스 설정

설정 이후에는 재부팅(reboot)을 통해서 시스템에 반영될 수 있도록 한다. 정상적으로 반영되었다면 라즈베리파이의 **/sys/bus** 경로에 **w1** 디렉터리가 존재한다. 만약 존재하지 않는다면 모듈을 적재하는 명령인 **modprobe**를 사용하여 생성한다.

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

`/sys/bus/w1/devices` 경로에 1-wire 인터페이스 연결 장치가 있으며 수온 센서(DS18B20)는 "28-*" 형태의 이름으로 존재한다. 해당 이름으로 이동하면 장치의 여러 정보를 확인할 수 있다. temperature와 w1_slave에 기록되는 정보로 수온 데이터를 얻을 수 있으며 과제에서는 w1_slave의 내용을 바탕으로 수온 데이터를 수집한다.

```
pi@raspberrypi:~ $ ls /sys/bus/w1
devices drivers drivers_autoprobe drivers_probe uevent
pi@raspberrypi:~ $ ls /sys/bus/w1/devices
28-3c01d607f349 w1_bus_master1
```

그림 27 수온 센서 장치 목록 확인

```
pi@raspberrypi:/sys/bus/w1/devices/28-3c01d607f349 $ ls
alarms driver ext_power hwmon name resolution temperature w1_slave
conv_time eeprom features id power subsystem uevent
```

그림 28 수온 센서 장치의 정보 확인

```
pi@raspberrypi:~ $ cat /sys/bus/w1/devices//28-3c01d607f349/w1_slave
75 55 7f 66 91 : crc=91 YES
75 55 7f 66 91 t=23312
```

그림 29 w1_slave에 기록된 내용 확인

w1_slave 내용 중 앞부분에 출력되는 값은 장치의 고유 번호이며, 첫 줄 마지막의 YES는 측정이 정상 완료되었음을 의미한다. 두 번째 줄 "t=값"으로 표현되는 부분이 측정된 온도 데이터에 해당한다. 섭씨온도로 변환하기 위해서는 측정된 값에서 1000을 나눈다. (ex. $23312/1000 = 23.312$)

③ 라즈베리파이 수온 센서 데이터 수집 코드 (Python3) - `sensor_temp_DS18B20.py`

```
import os
import glob
import time

# os.system('modprobe w1-gpio')          <- ㉠
# os.system('modprobe w1-therm')

w1_device_dir = '/sys/bus/w1/devices/'   <- ㉡
w1_data_dir = glob.glob(w1_device_dir + '28*')[0]
w1_device_file = w1_data_dir + '/w1_slave'

def read_temp_raw():                      <- ㉢
    f = open(w1_device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():                          <- ㉤
    lines = read_temp_raw()

    while lines[0].strip()[-3:] != 'YES':
        time.sleep(1.0)
        lines = read_temp_raw()

    equals_pos = lines[1].find('t=')

    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        return temp_c

if __name__ == "__main__":               <- ㉥
    while True:
        print("CEL temperature =%.2f" %read_temp())
        time.sleep(20)
```

※ 코드 해석

<p>㉑ 1-wire 인터페이스 장치가 정상적으로 인식하지 않을 때 모듈을 적재. 인터페이스 연결이 정상적으로 된 상태이기 때문에 주석처리.</p>
<p>㉒ 측정 데이터를 획득할 경로를 지정. w1_device_dir = 1-wire 인터페이스 장치들이 기록된 디렉터리 지정. w1_data_dir = w1_device_dir에서 “28”로 시작하는 장치 중 첫 번째 원소의 경로 지정. w1_device_file = w1_data_dir에서 w1_slave의 경로 지정.</p>
<p>㉓ w1_device_file에 해당하는 경로에서 기록된 내용을 읽어서 반환한다. lines는 w1_device_file에 기록된 내용을 줄 띄움을 기준으로 리스트를 구성한다.</p>
<p>㉔ ㉓에서 반환받은 lines 리스트를 바탕으로 값을 해석한다. 첫 줄의 마지막 내용이 YES가 아니면 정상적인 측정이 아니므로 다시 측정한다. 정상적인 측정 결과이며 그 값이 -1이 아니면 “t=”문자열이 있는 인덱스를 찾는다. “t=”문자열을 찾은 인덱스의 2문자 뒤부터 끝까지의 문자열이 곧 수온 데이터이다. 해당 값을 수치 데이터로 변환한 다음 1000을 나누어 실수 형태의 섭씨온도를 획득한다.</p>
<p>㉕ 측정된 수온 데이터를 소수점 2자리까지 표시하여 출력한다. 출력한 다음에는 20초의 대기시간을 가지며 해당 작업을 반복한다.</p>

※ 실행 결과

```

Shell
Python 3.7.3 (/usr/bin/python3)
>>> %Run sensor_temP_DS18B20.py

CEL temperature =23.31
CEL temperature =23.38
CEL temperature =23.44
CEL temperature =23.44

```


▶ GPS 센서 연결 및 코드 구현

① GPS 센서 회로 연결

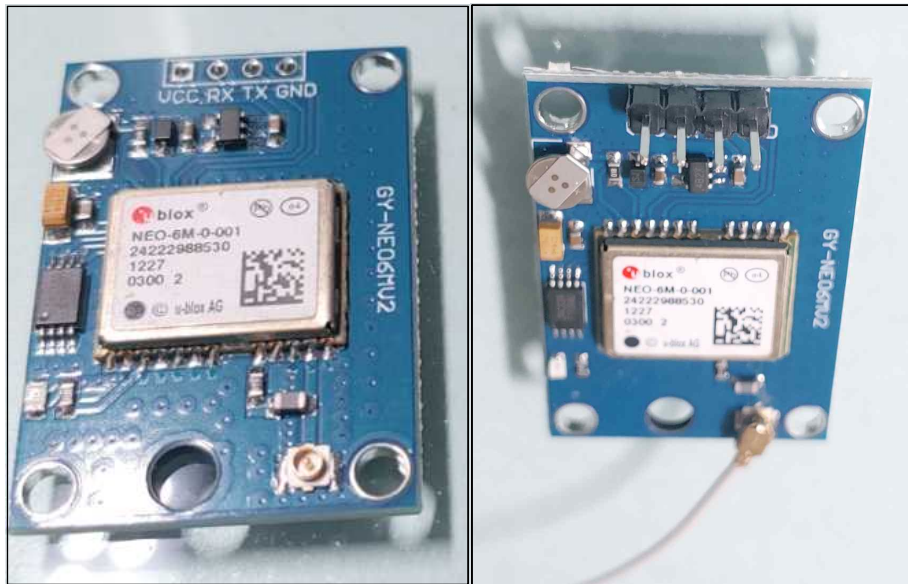


그림 31 핀 헤더를 GPS 모듈에 납땜

GPS 센서를 구매하면 구성품으로 센서 모듈과 안테나를 받는다. 하지만 핀을 연결할 수 있는 헤드가 없어서 핀 헤더를 별도로 구매하여 납땜 작업을 한다. GPS 센서 모듈인 NEO-6M은 시리얼 통신인 UART를 지원한다. 따라서 라즈베리파이에서 UART 통신을 지원하는 GPIO14(Tx), GPIO15(Rx)를 각각 교차하여 모듈의 Rx, Tx와 연결한다. VCC는 라즈베리파이에 무리가 가지 않도록 3.3v와 연결한다.

② 라즈베리파이 UART(Serial) 인터페이스 연결 설정

라즈베리파이에서 UART는 miniUART와 PL011을 지원하며 각각 리눅스 콘솔 연결이나 블루투스 모듈 활성화에 사용된다. GPS 센서 모듈을 사용하면서 의도하지 않은 오류를 방지하기 위해 설정을 통해서 다른 UART 연결을 해제한다. 라즈베리파이에서의 GPIO14, GPIO15는 miniUART에 해당하며 디바이스 트리 오버레이 설정을 통해 외부 장치와의 연결을 활성화하여 통신한다.

②-1 라즈베리파이 UART 기본 설정

라즈베리파이의 "Preferences -> Raspberry Pi Configuration -> Interfaces"에서 Serial Port를 Enable, Serial Console을 Disable 설정한다. 외부 장치를 연결하며 리눅스 콘솔 등의 UART 연결을 차단하기 위함이다.

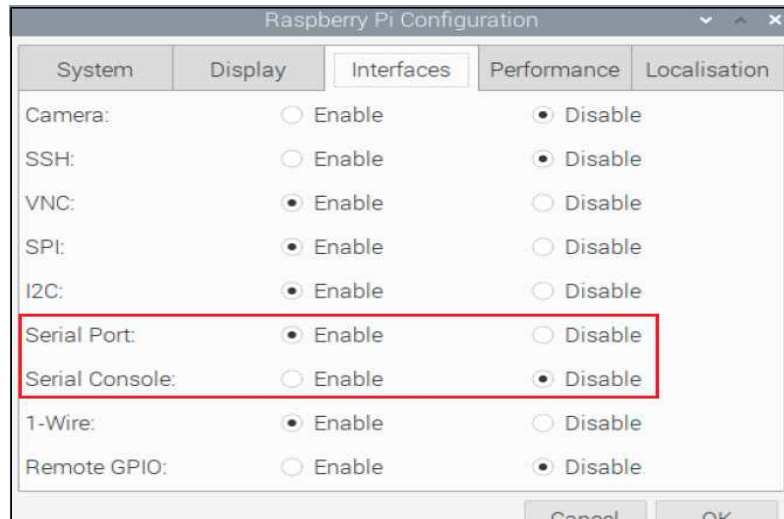


그림 32 Serial 설정

라즈베리파이 설정 툴(Raspberry Pi Configuration Tool) 명령인 "raspi-config"로 시리얼 통신을 설정한다.

```
sudo raspi-config
```

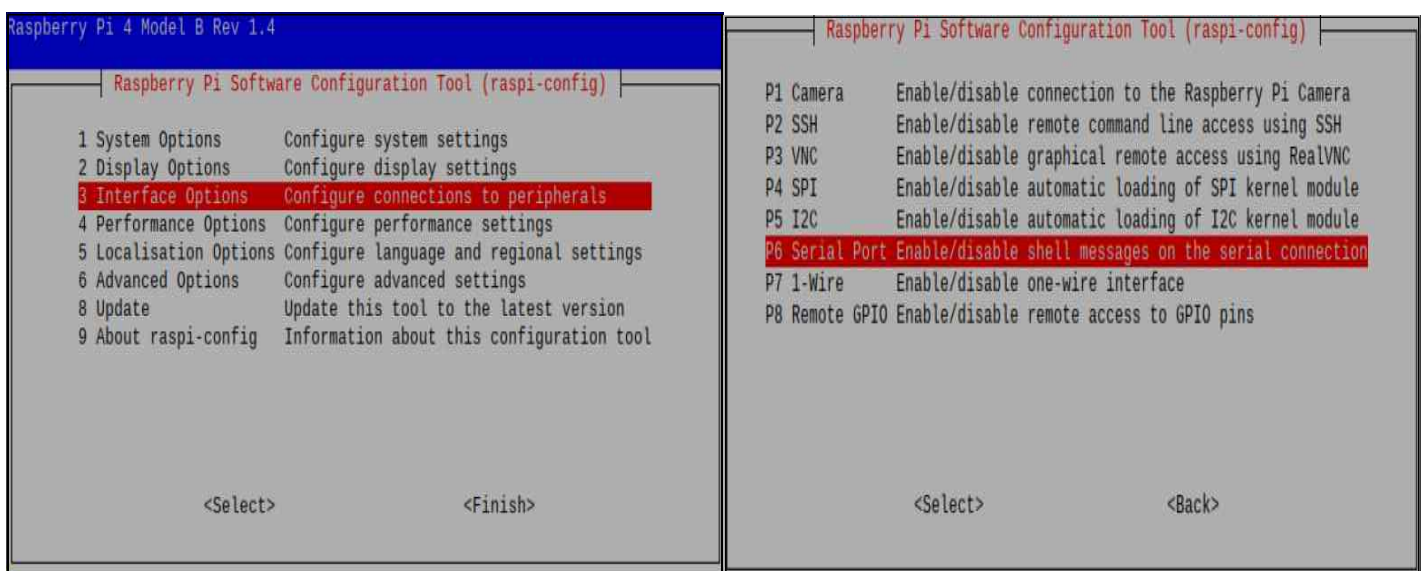


그림 33 Interface Options 선택

그림 34 Serial Port 선택

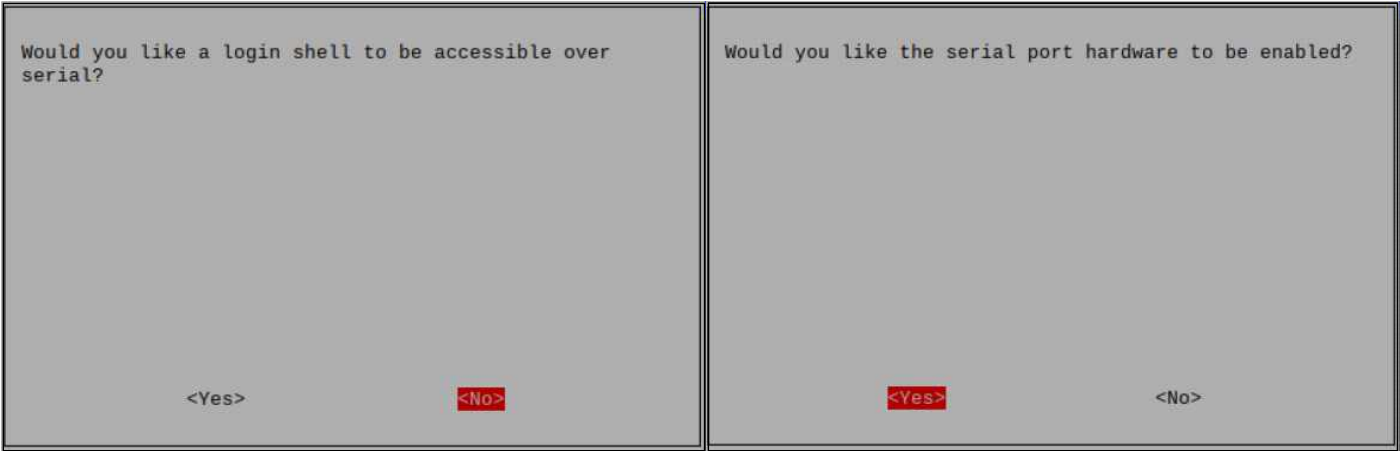


그림 35 Login Shell 관련 설정 No

그림 36 Serial Port 관련 설정 Yes

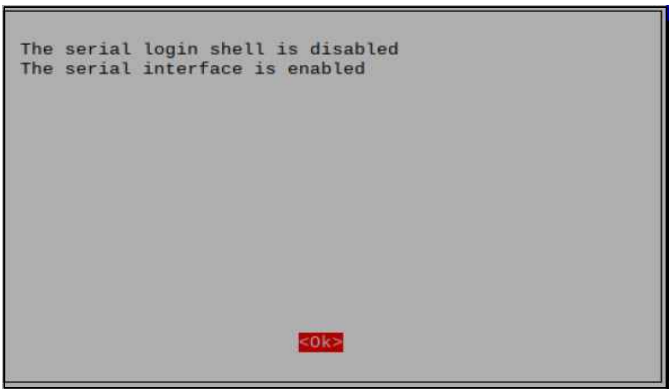


그림 37 설정 내용 최종 확인

라즈베리파이 터미널에서 GPS 센서 데이터 수집 및 측정을 도와주는 패키지를 설치한다. 패키지 설치 이전에 업데이트와 업그레이드를 진행하여 최신화를 유지한다.

sudo apt-get update	<- 사용 가능한 패키지 리스트의 버전 업데이트
sudo apt-get full-upgrade	<- 변경된 의존성을 포함한 패키지 버전 업그레이드
sudo apt-get install python3-pip	<- python3 버전 pip3 설치
sudo pip install pip --upgrade	<- pip 업그레이드 (최신 버전 확인 용도)
sudo pip3 install pyserial	<- 시리얼 통신 관련 라이브러리
sudo pip3 install pynmea2	<- GSP NMEA 프로토콜 관련 라이브러리
sudo apt-get install gpsd gpsd-clients	
GPS 데이터 수집 및 클라이언트 애플리케이션 프로그램	

모든 설치가 완료되었다면 reboot 명령을 통해 라즈베리파이의 설정을 완료한다.

②-2 라즈베리파이 UART 상세 설정

GPS 센서 모듈과 연결되는 라즈베리파이의 GPIO14, GPIO15 핀의 활성화를 위한 설정을 진행한다. dtoverlay로 시스템이 작동하는 동안 연결된 하드웨어나 오버레이 관련 정보나 설정을 할 수 있다. UART에 사용하는 GPIO14, GPIO15의 BCM번호를 확인한 다음, /boot/config.txt 파일을 수정하여 장치 인터페이스를 활성화 한다.

dtoverlay -a grep uart	<- uart에 해당하는 dtoverlay 정보 출력
dtoverlay -h uart5	<- uart5에 해당하는 정보 상세 출력

```
pi@raspberrypi:~ $ dtoverlay -a | grep uart
midi-uart0
midi-uart1
miniuart-bt
uart0
uart1
uart2
uart3
uart4
uart5
pi@raspberrypi:~ $ dtoverlay -h uart5
Name: uart5
Info: Enable uart 5 on GPIOs 12-15. BCM2711 only.
Usage: dtoverlay=uart5,<param>
Params: ctsrts Enable CTS/RTS on GPIOs 14-15 (default off)
```

그림 38 GPIO14, GPIO15에 해당하는 uart 정보 확인

sudo nano /boot/config.txt

[all] 항목에 dtoverlay=uart5 추가	<- uart5에 인터페이스 등록
[all] 항목에 enable_uart=1 추가	<- miniUART 활성화

```
[all]
#dtoverlay=vc4-fkms-v3d
dtoverlay=w1-gpio
dtoverlay=uart5
enable_uart=1
```

그림 39 /boot/config.txt 파일 내용 수정

설정이 완료되면 재부팅하여 반영될 수 있도록 한다.

터미널에서 “ls -l /dev” 명령을 통해 UART 장치가 정상적으로 연결되어 있는지 확인한다. miniUART에 해당하는 장치는 serial0이며 ttyS0도 같다. serial1 또는 ttyAMA0은 PL011이기 때문에 신경 쓰지 않는다.

```
brw-rw---- 1 root disk      1,  9 May 18 22:57 ram9
crw-rw-rw- 1 root root       1,  8 May 18 22:57 random
drwxr-xr-x 2 root root      60, 60 Jan  1 1970 raw
crw-rw-r-- 1 root netdev   10, 242 May 18 22:57 rfkill
crw-rw---- 1 root video  239,  0 May 18 22:57 rpivid-h264mem
crw-rw---- 1 root video  241,  0 May 18 22:57 rpivid-hevcmem
crw-rw---- 1 root video  240,  0 May 18 22:57 rpivid-intcmem
crw-rw---- 1 root video  238,  0 May 18 22:57 rpivid-vp9mem
lrwxrwxrwx 1 root root       5 May 18 22:57 serial0 -> ttyS0
lrwxrwxrwx 1 root root       7 May 18 22:57 serial1 -> ttyAMA0
drwxrwxrwt 2 root root     40, 40 Feb 14 2019 shm
drwxr-xr-x 3 root root    140, 140 May 18 22:57 snd
crw-rw---- 1 root spi    153,  0 May 18 22:57 spidev0.0
crw-rw---- 1 root spi    153,  1 May 18 22:57 spidev0.1
```

그림 40 serial0 -> ttyS0 확인

GPS 작동을 위한 GPS 데몬(gpsd) 설정을 진행한다. gpsd 관련 설정은 /etc/default/gpsd 파일에서 수행한다. 설정하는 항목은 크게 3가지이다. 시스템이 실행될 때 gpsd를 자동으로 실행하는 것, GPS가 연결된 장치 경로, gpsd 제어 소켓 경로이다.

sudo nano /etc/default/gpsd

<- gpsd 관련 설정 파일 수정

START_DAEMON = "true"

<- 시스템 작동 시 자동으로 gpsd 데몬 실행

DEVICES = "/dev/serial0"

<- GPS와 연결된 UART 인터페이스 지정

GPSPD_OPTIONS = "-F /var/run/gpsd.sock"

<- gpsd 제어 소켓 생성

```
pi@raspberrypi:~ $ cat /etc/default/gpsd
# Default settings for the gpsd init script and the hotplug wrapper.

# Start the gpsd daemon automatically at boot time
START_DAEMON="true"

# Use USB hotplugging to add new USB devices automatically to the daemon
USB_AUTO="true"

# Devices gpsd should collect to at boot time.
# They need to be read/writeable, either by user gpsd or the group dialout.
DEVICES="/dev/serial0"

# Other options you want to pass to gpsd
GPSPD_OPTIONS="-F /var/run/gpsd.sock"
```

그림 41 /etc/default/gpsd 내용 수정

설정이 완료되었다면 재부팅하여 적용한다.

③ GPS 센서 데이터 확인

GPS 안테나를 측정이 잘 되는 장소(야외)에 배치한 다음 GPS 모듈에서 빨간색(다른 색일 수도 있음) 점 등이 반복해서 일어나는지 확인한다. 점등을 확인했다면 센서로부터 데이터가 수집되었는지 확인한다. 확인하는 방법은 장치 경로의 내용을 확인하는 `cat /dev/serial0` (또는 `cat /dev/ttyS0`)과 `cgps -s` 명령을 통해 GPS 해석 프로그램을 사용하는 방법이 있다.

```
pi@raspberrypi:~ $ cat /dev/ttyS0
$GPVTG,,,,,,N*30
$GPGGA,150803.00,,,,,0,06,12.60,,,,,*6A
$GPGSA,A,1,20,18,05,24,23,14,,,,,25.98,12.60,22.72*3A
$GPGSV,3,1,11,05,38,111,16,10,07,305,,13,38,048,,14,08,054,10*7B
$GPGSV,3,2,11,15,66,017,08,18,54,274,27,20,69,106,13,23,37,313,20*70
$GPGSV,3,3,11,24,61,185,15,28,11,068,16,29,01,220,*49
$GPGLL,,,,,150803.00,V,N*45
$GPGST,150803.00,58,,,,,18,11,8.2*56
$GPZDA,150803.00,18,05,2021,00,00*64
$GPGBS,150803.00,,,,,,*60
$GPRMC,150804.00,V,,,,,,180521,,,N*7A
```

그림 42 `cat /dev/ttyS0`, GPS 데이터 확인

Time:	2021-05-09T04:58:42.000Z	PRN:	Elev:	Azim:	SNR:	Used:
Latitude:	35.90977007 N	3	20	156	24	Y
Longitude:	128.81633319 E	7	33	215	31	Y
Altitude:	71.788 m	8	40	062	15	Y
Speed:	0.17 kph	14	37	314	14	Y
Heading:	104.3 deg (true)	17	19	277	29	Y
Climb:	-8.22 m/min	21	67	044	15	Y
Status:	3D FIX (44 secs)	22	37	129	26	Y
Longitude Err:	+/- 7 m	27	10	077	13	Y
Latitude Err:	+/- 10 m	30	39	266	28	Y
Altitude Err:	+/- 37 m	1	85	199	21	N
Course Err:	n/a					
Speed Err:	+/- 1 kph					
Time offset:	0.144					
Grid Square:	PM45jv					

그림 43 `cgps -s`, GPS 데이터 해석 프로그램

④ 위도·경도 데이터 수집 코드 작성 (python3) - gps_test.py

과제에 필요한 GPS 데이터는 위치를 계산하기 위한 위도와 경도이다. python3로 위도와 경도 데이터만 가져와서 활용할 수 있도록 한다.

```
import serial
import pynmea2
import time

GPS_PORT = "/dev/ttyS0"                                <- ㉓
BAUDRATE = 9600

while True:
    serial_gps = serial.Serial(GPS_PORT, BAUDRATE)      <- ㉔
    if serial_gps.readable():
        data = serial_gps.readline()
        data_text = data.decode()
    else:
        print("error")

    if data_text[0:6] == "$GPRMC":                      <- ㉕
        location_data = pynmea2.parse(data_text)
        latitude_data = location_data.latitude
        longitude_data = location_data.longitude

        location = [latitude_data, longitude_data]
        print(f"Latitude = {location[0]}, Longitude = {location[1]}")

    time.sleep(20)
```

※ 코드 해석

㉓ GPS 데이터가 수집되는 장치 경로와 GPS 보드레이트(9600) 지정.

㉔ 지정한 장치 경로와 보드레이트를 바탕으로 시리얼 통신 결과 serial_gps에 저장.
데이터가 존재 한다면("readable()") 해당 데이터를 data에 저장하고, 데이터를 디코딩하여 "\$GP*" 형태의 문자열을 갖는 위치 관련 정보를 data_text에 저장.

㉕ 디코딩 결과의 정보가 "\$GPRMC"로 시작한다면 해당 정보를 NMEA 규격에 맞게 파싱하여 location_data에 저장.
location_data에는 위도(latitude)와 경도(longitude) 데이터가 존재하며 각각을 추출하여 리스트에 저장 후 출력.

※ 실행 결과

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run gps_test.py

Latitude = 35.909808166666664, Longitude = 128.81625866666667
Latitude = 35.9098445, Longitude = 128.81625316666666
```

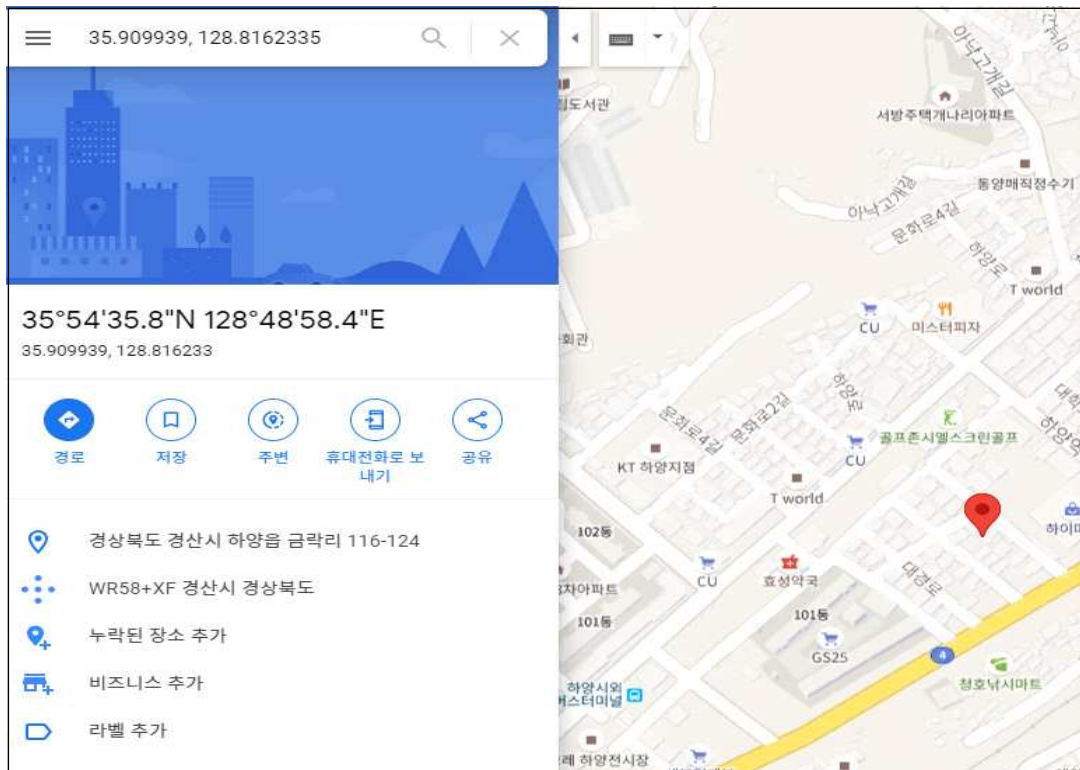


그림 45 측정된 위도와 경도를 구글 맵스로 검색

▶ ADC(Analog-to-Digital Converter) MCP3008 연결 및 샘플 코드

① MCP3008 회로 연결 및 라즈베리파이 설정



그림 46 MCP3008

과제 수행에서 사용되는 수소이온농도(pH), 용존산소량(DO), 탁도 센서는 아날로그 데이터로 수집한다. 라즈베리파이는 디지털 데이터만 수용할 수 있어서 아날로그 데이터를 디지털 데이터로 변환해주는 ADC 장치가 필요하며 MCP3008을 사용한다.

MCP3008은 라즈베리파이와 SPI 통신을 한다. 회로도 SPI에 맞추어 제작되었으며 다음과 같이 연결한다.

VDD, VREF = 3.3V	AGND, DGND = GND
CLK = SPI SCLK	DOUT = SPI MISO
DIN = SPI MOSI	CS/SHDN = SPI CE0 or SPI CE1

라즈베리파이에서는 Raspberry Pi Configuration에서 SPI 항목을 Enable 선택한다.

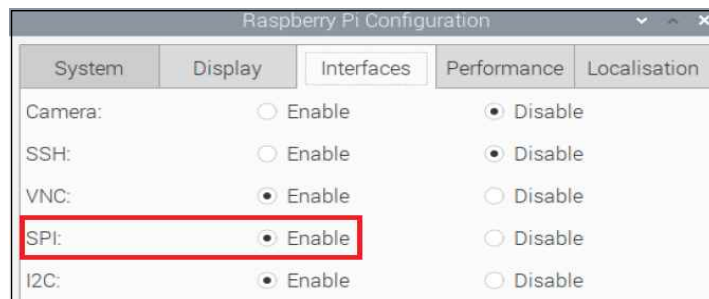


그림 47 SPI Enable

3개의 아날로그 센서를 SPI 버스에서 제어하기 위해 spidev 파이썬 패키지를 설치한다. 설치가 완료되었으면 "pip3 show spidev" 명령으로 확인할 수 있다.

```
pip3 install spidev
```

```
pi@raspberrypi:~ $ pip3 show spidev
Name: spidev
Version: 3.4
Summary: Python bindings for Linux SPI access through spidev
Home-page: http://github.com/doceme/py-spidev
Author: Stephen Caudle
Author-email: scaudle@doceme.com
License: GPLv2
Location: /usr/lib/python3/dist-packages
Requires:
Required-by: unicornhathd
```

그림 48 spidev 설치 확인

② MCP3008 회로 연결 상태 확인 샘플 코드 작성 (python3) - mcp3008.py

```
import spidev, time

spi = spidev.SpiDev()                                <- ㉠
spi.open(0,0)
spi.max_speed_hz = 1350000

def analog_read(channel):                             <- ㉡
    r = spi.xfer2([1, (8 + channel) << 4, 0])
    adc_out = ((r[1]&3) << 8) + r[2]
    return adc_out

while True:                                           <- ㉢
    reading = analog_read(0)
    voltage = reading * 3.3 / 1024
    print("Reading = %d \ tVoltage = %f" %(reading, voltage))
    time.sleep(3)
```

※ 코드 해석

㉠ SPI 객체를 생성한 다음 객체를 통해 SPI 버스를 제어.

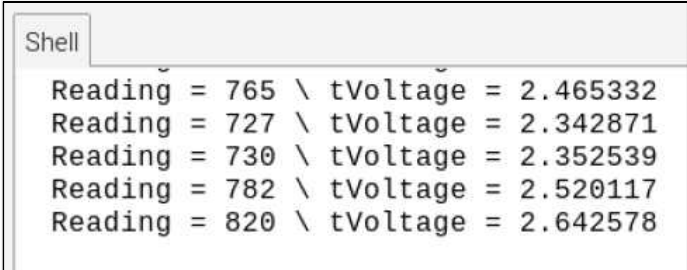
open()은 버스 번호, CS(Chip Select) 번호를 인자로 받는다. 라즈베리파이는 1개의 버스(0)와 2개의 CS(CE0, CE1)를 제공한다. 과제에서는 CE0과 연결했기 때문에 open(0,0)으로 버스를 연다.

max_speed_hz는 Clock Frequency의 최댓값이다. MCP3008 데이터시트에 기반하여 3.3v로 연결했을 때 1.35MHz(1350000Hz)이다.

㉡ MCP3008의 채널(0~7) 중 하나의 채널의 데이터를 송수신하는 함수.

㉢ 0번 채널에 연결된 SPI 통신 데이터를 목적에 맞게 변환하여 사용.

※ 실행 결과



```
Shell
Reading = 765 \ tVoltage = 2.465332
Reading = 727 \ tVoltage = 2.342871
Reading = 730 \ tVoltage = 2.352539
Reading = 782 \ tVoltage = 2.520117
Reading = 820 \ tVoltage = 2.642578
```

그림 49 MCP3008 모듈 값 출력

▶ pH 센서(SEN0169) 연결 및 코드

pH 센서 모듈은 5V 지원 모델이다. 3.3V를 초과하는 전압은 라즈베리파이에 고장을 발생시키기 때문에 pH 센서를 사용하기 위해서는 전압 분배 법칙을 적용한다. 저항을 사용한 전압 분배는 위의 회로 구성을 참고한다.

아날로그 센서 데이터를 생산하는 모듈이기 때문에 라즈베리파이로 수집하기 위해서는 디지털 데이터로 변환하는 과정이 필요하다. 이를 위해 ADC 모듈을 사용하며 MCP3008의 0번 채널에 pH 센서를 연결한다. 데이터가 정상적으로 들어오는지 ADC 모듈을 통해 확인할 수 있으며 pH 센서 모듈의 샘플 코드를 바탕으로 전압을 계산하여 값을 구한다.

값의 교정을 위해 pH 7.0 고정액을 사용한다. 값의 차이가 존재한다면 오프셋(offset) 상수를 통해 추가 교정을 수행한다. pH 값을 측정하는 코드는 아래와 같다.

※ Python3 – `sensor_pH.py`

```
import spidev                # SPI 객체 관련
import time                   # 시간 처리 관련
import sensor_temP_DS18B20 as temp    # 수온 측정 관련

spi = spidev.SpiDev()        # SPI Device Instance(SPI 객체 생성)
spi.open(0, 0)                # open(spi_bus, device_channel). CS=0
spi.max_speed_hz=1350000     # SPI speed (1350000hz) = 1.35MHz

pH_channel = 0                # MCP3008 CH0 연결
OFFSET = 0.47                 # 교정 상수

def readadc(adc_channel):    # read out the ADC
    if ((adc_channel > 7) or (adc_channel < 0)): # Out of range MCP3008 port
        return -1
    r = spi.xfer2([1, (0x08 + adc_channel) << 4, 0])
    adc_out = ((r[1] & 0x03) << 8) + r[2]
    return adc_out

if __name__ == "__main__":
    while True:
        temperature = round(temp.read_temp())    # 수온 측정
        Value = readadc(pH_channel)              # read ADC channel 0
        voltage = Value*3.3/1024                  # pH 센서에서 측정한 전압 계산
        phvalue = voltage*3.5 + OFFSET            # pH 값 계산

        print("Temperature : %d" %temperature)
        print("MCP3008 read value = %d " % Value)
        print("change to V = %.2lf " % voltage)
        print (phvalue, "PH")
        time.sleep(5)
```

※ 실행 결과

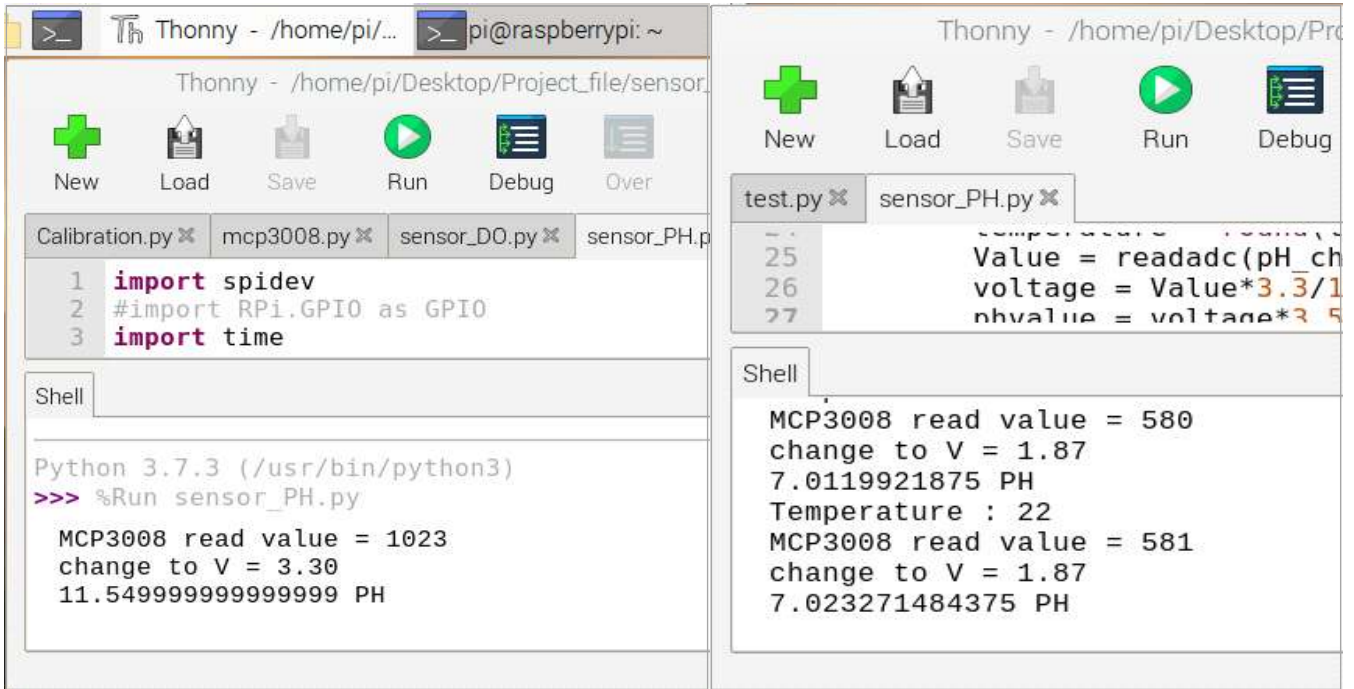


그림 50 교정 전 pH 센서 측정 결과(좌측)와 교정 후 측정 결과(우측)

▶ 탁도 센서(SEN0189) 연결 및 코드

탁도 센서도 pH 센서와 마찬가지로 5V 지원 모델이기 때문에 전압 분배 법칙을 적용한다. 탁도 단위를 적용하지 않고 측정된 전압을 바탕으로 혼탁 정도를 판단한다. 0V의 값에 가까울수록 탁한 수치이며 3.3V에 가까울수록 맑은 수치이다. 최대 3.3V인 이유는 전압 분배 법칙과 ADC에 의해서이다. MCP3008의 7번 채널을 사용했으며 데이터를 측정하는 코드 외에는 pH 측정 코드와 동일하다.

※ Python3 – `Turbidity.py`

```
import spidev
import time

TURB_CHANNEL = 7          # MCP3008 CH7 연결

spi = spidev.SpiDev()      # SPI 객체 생성
spi.open(0,0)              # open(spi_bus, device_channel) for the MCP3008
spi.max_speed_hz = 1350000 # SPI speed (1350000hz) = 1.35MHz

def readadc(adc_channel):  # read out the ADC
    if ((adc_channel > 7) or (adc_channel < 0)): # Out of range MCP3008 port
        return -1
    r = spi.xfer2([1, (0x08 + adc_channel) << 4, 0])
    adc_out = ((r[1] & 0x03) << 8) + r[2]
    return adc_out
```

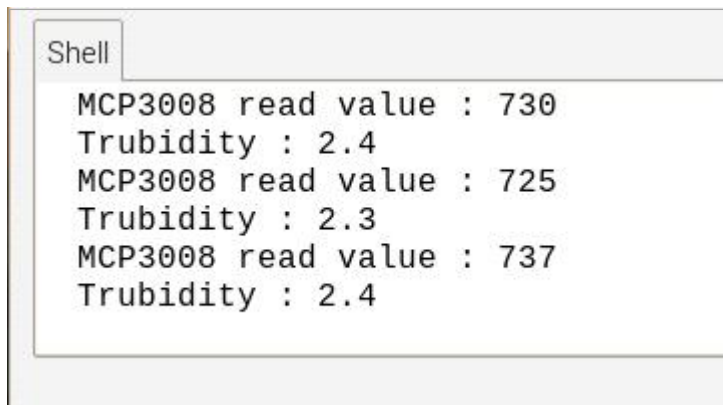
```

if __name__ == "__main__":
    while True:
        Turb_value = readadc(TURB_CHANNEL)          # read ADC channel 7
        voltage = Turb_value * (3.3 / 1023)         # 탁도 센서에서 측정한 전압 계산
        print("MCP3008 read value : %d" %Turb_value)
        print("Trubidity : %.1lf" %voltage)

        time.sleep(20)

```

※ 실행 결과



```

Shell
MCP3008 read value : 730
Trubidity : 2.4
MCP3008 read value : 725
Trubidity : 2.3
MCP3008 read value : 737
Trubidity : 2.4

```

그림 51 탁도 센서 측정 결과

▶ 용존산소 센서(SEN0237) 연결 및 코드

용존산소 센서 모듈은 3.3V~5V 지원이어서 3.3V에 직접 연결하여 사용한다. 용존산소 센서의 측정 샘플 코드를 바탕으로 Python3 코드를 구현한다. 수온과 측정된 전압을 바탕으로 용존산소의 정도를 판단하며 코드는 아래와 같다.

※ Python3 – `sensor_DO.py`

```

import spidev
import time
import sensor_temP_DS18B20 as temp

spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 1350000

DO_Channel = 1          # MCP3008 CH1 연결
TWO_CALIBRATION = 1     # 교정법 상수

# value (mg/L * 1000), refer to the DO_table in the DFRobot.com

```



```

DO_Table = [14600, 14220, 13800, 13440, 13080, 12760, 12440, 12110, 11830, 11560,
11290, 11040, 10760, 10540, 10310, 10060, 9860, 9640, 9470, 9270,
9090, 8910, 8740, 8570, 8410, 8250, 8110, 7960, 7830, 7680,
7560, 7430, 7300, 7170, 7060, 6940, 6840, 6720, 6600, 6520,
6400, 6330, 6230, 6130, 6060, 5970, 5880, 5790]

def readadc(adc_channel):
    if ((adc_channel > 7) or (adc_channel < 0)):
        return -1
    r = spi.xfer2([1, (0x08 + adc_channel) << 4, 0])
    adc_out = ((r[1] & 0x03) << 8) + r[2]
    return adc_out

def readDO(mv_value, temp_c):
    CAL1_V = 1600
    CAL1_T = 25
    CAL2_V = 1300
    CAL2_T = 15
    if TWO_CALIBRATION == 0:
        V_saturation = CAL1_V + 35 * temp_c - CAL1_T * 35
    else:
        V_saturation = (temp_c - CAL2_T) * (CAL1_V - CAL2_V) / (CAL1_T - CAL2_T) + CAL2_V

    return mv_value * DO_Table[temp_c] / V_saturation

if __name__ == "__main__":
    while True:

        temperature = round(temp.read_temp())
        ADC_value = readadc(DO_Channel)
        ADC_voltage = ADC_value * 3.3 / 1024

        print("Temperature : %d" %temperature)
        print("MCP3008 read value : %d" %ADC_value)
        print("ADC_voltage = %.2lf" %ADC_voltage)
        print(readDO(ADC_voltage, temperature), " mg/L")

        time.sleep(20)

```

readDO() 함수는 ADC로 전달한 용존산소 센서의 전압과 측정된 수온을 바탕으로 용존산소 값(mg/L)을 계산한다. 계산에 활용되는 DO_Table과 계산식은 용존산소 센서 개발 문서의 샘플 코드를 바탕으로 작성한다.

※ 실행 결과

```
Shell
python 3.7.5 (/usr/bin/python3)
>>> %Run sensor_D0.py

Temperature : 24
MCP3008 read value : 9
ADC_voltage = 0.03
0.1555167454283953 mg/L
```

그림 52 용존산소 센서 측정 결과

2-2. 메인 서버 및 데이터베이스 구축

▶ 메인 서버(Ubuntu 20.04) 및 MariaDB 구축

메인 서버는 라즈베리파이에서 측정된 수질 데이터를 받아와서 데이터베이스에 저장하고, 저장된 수질 데이터를 처리한다. 데이터베이스는 MariaDB 기반으로 구축하며 4가지의 수질 데이터(pH, DO, 탁도, 수온)와 측정 시간, 측정 장비 번호 값을 저장한다. 또한, 측정 장비 번호를 기본키로 두는 테이블을 만들고, 설치된 위치를 위도, 경도 값으로 저장한다. 이는 설치된 장소의 장비를 판단하고, 추후 관리를 용이하게 한다.

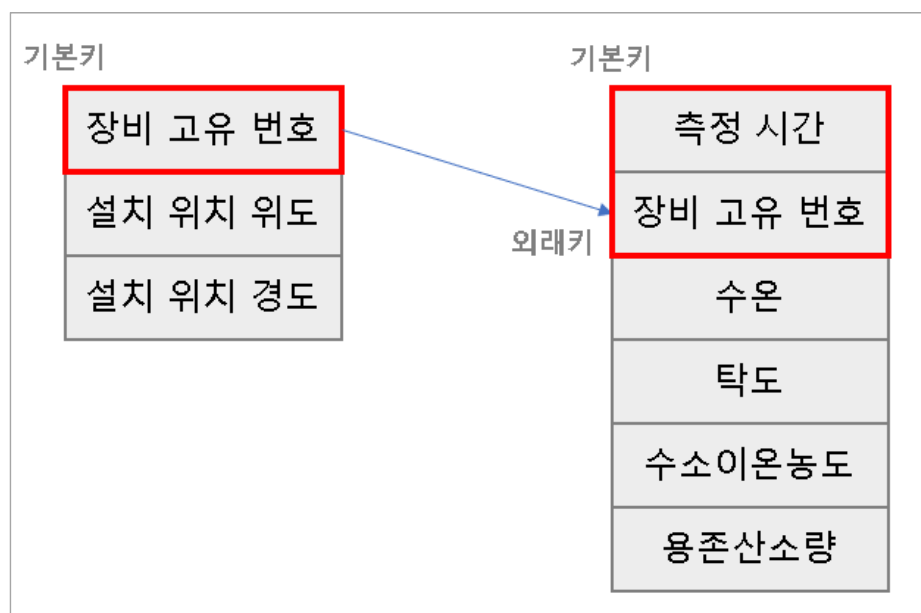


그림 53 MariaDB 수질 측정 데이터 저장 테이블 구조

① MariaDB 설치

메인 서버에 MariaDB 설치를 진행한다. 설치 전에 Ubuntu에서 패키지 업데이트와 업그레이드를 진행한다. 이후에 mariadb-server 패키지를 설치한 다음, 기본 보안 설정을 마무리하고서 준비를 완료한다. MariaDB를 쉽고 효과적으로 관리하기 위해 HeidiSQL(하이지드SQL) 프로그램을 설치하여 GUI 환경에서 작업한다. CLI 환경이 익숙하다면 CMD에서 진행해도 무관하다.

```
sudo apt-get update && sudo apt-get full-upgrade  
sudo apt install mariadb-server
```

```
pcb@ubuntu:~$ sudo apt-get update && sudo apt-get full-upgrade  
[sudo] password for pcb:  
Get:1 http://packages.microsoft.com/repos/code stable InRelease [10.4 kB]  
Get:2 http://dl.google.com/linux/chrome/deb stable InRelease [1,811 B]  
Get:3 http://packages.microsoft.com/repos/code stable/main arm64 Packages [34.9 kB]  
Get:4 http://packages.microsoft.com/repos/code stable/main amd64 Packages [34.0 kB]  
Get:5 http://packages.microsoft.com/repos/code stable/main armhf Packages [34.8 kB]
```

```
pcb@ubuntu:~$ sudo apt install mariadb-server  
[sudo] password for pcb:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  galera-3 gawk libaio1 libcgi-fast-perl libcgi-pm-perl  
  libconfig-inifiles-perl libdbd-mysql-perl libdbi-perl libfcgi-perl  
  libhtml-template-perl libreadline5 libsigsegv2 libterm-readkey-perl  
  mariadb-client-10.3 mariadb-client-core-10.3 mariadb-common  
  mariadb-server-10.3 mariadb-server-core-10.3 socat  
Suggested packages:
```

그림 54 패키지 업데이트&업그레이드 및 mariadb-server 패키지 설치

패키지 설치를 완료했다면 MariaDB의 접속 환경(보안 등)을 설정한다. “mysql_secure_installation”에서 설정할 수 있으며 MariaDB는 MySQL 기반이기 때문에 mysql의 이름을 가지고 있다. 5가지의 설정 항목이 존재한다.

```
sudo mysql_secure_installation
```

- Ⓐ Change the root password?: root 계정의 접속 비밀번호를 변경한다. [y] = 비밀번호 설정
- Ⓑ Remove anonymous users?: 익명의 사용자 접속을 제한(제거)한다. [y] = 제거
- Ⓒ Disallow root login remotely?: root 계정을 원격으로 접속하는 것을 거부한다. [y/n] = 필요에 따라 선택
- Ⓓ Remove test database and access to it?: 테스트용 데이터베이스를 제거하고 접속한다. [y/n]
- Ⓔ Reload privilege tables now?: 권한에 해당하는 테이블을 재부팅한다. [y] = 최신 설정 반영

```
pcb@ubuntu:~/Downloads$ sudo mysql_secure_installation
[sudo] password for pcb:

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user.  If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorisation.

You already have a root password set, so you can safely answer 'n'.

Change the root password? [Y/n] y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
... Success!

By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y
... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] n
... skipping.

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] n
... skipping.

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

그림 55 mysql_secure_installation 설정

설정까지 모두 완료하면 MariaDB에 접속해본다. 다음은 root 사용자로 MariaDB에 접속한 것이다.

```
sudo mysql -u root -p
-u : 접속할 계정, -p : 패스워드 입력 (패스워드 없다면 생략 가능)
```

```
pcb@ubuntu:~$ sudo mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 50
Server version: 10.3.29-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

그림 56 root 계정으로 MariaDB 접속

데이터베이스에 접속하면 SQL문을 통해서 명령을 처리한다. 간단하게 데이터베이스를 확인하는 명령과 접속할 수 있는 사용자를 확인하고, 생성하는 명령을 사용했다.

show databases;	<- 저장된 데이터베이스 목록을 표시
use mysql;	<- mysql의 SQL로 활용
select host, user from user;	<- user 목록에서 host와 user 리스트 검색
create user 'testuser'@'localhost' identified by 'dfg09787';	
localhost에서 접속 가능한 testuser 사용자를 생성하고, 비밀번호를 dfg09787로 설정	

```
MariaDB [(none)]> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [mysql]> select host, user from user;
+-----+-----+
| host      | user      |
+-----+-----+
| localhost | root      |
+-----+-----+
1 row in set (0.000 sec)

MariaDB [mysql]> create user 'testuser'@'localhost' identified by 'dfg09787';
Query OK, 0 rows affected (0.001 sec)

MariaDB [mysql]> select host, user from user;
+-----+-----+
| host      | user      |
+-----+-----+
| localhost | root      |
| localhost | testuser   |
+-----+-----+
2 rows in set (0.000 sec)
```

```
MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql         |
| performance_schema |
+-----+
3 rows in set (0.000 sec)

MariaDB [(none)]>
```

그림 57 MariaDB에 저장된 데이터베이스 정보 및 사용자 정보 확인, 새로운 사용자 생성

② HeidiSQL 설치

HeidiSQL(하이지드SQL)은 데이터베이스를 쉽게 확인하고 관리할 수 있는 프로그램이다. 이전에 “MySQL Front”였던 만큼 MariaDB에 접근이 가능하며 추후 라즈베리파이에서 수집되는 데이터가 정상적으로 저장되었는지 실시간으로 확인하기 쉽다. HeidiSQL은 설치파일로 exe 확장자를 제공하기 때문에 우분투와 호환되지 않는다. 따라서 exe 확장자를 설치할 수 있게 도와주는 wine 패키지를 설치한다.

```
sudo apt install wine
```

```
pcb@ubuntu:~/Downloads$ sudo apt install wine
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  fonts-wine gcc-10-base:i386 glib-networking:i386
  gstreamer1.0-plugins-base:i386 gstreamer1.0-plugins-good:i386
  gstreamer1.0-x:i386 i965-va-driver:i386 intel-media-va-driver:i386
  libaa1:i386 libaom0:i386 libapparmor1:i386 libasn1-8-heimdal:i386
  libasound2:i386 libasound2-plugins:i386 libasyncns0:i386 libatomic1:i386
  libavahi-client3:i386 libavahi-common-data:i386 libavahi-common3:i386
  libavc1394-0:i386 libavcodec58:i386 libavutil56:i386 libblkid1:i386
  libbrotli1:i386 libbsd0:i386 libbz2-1.0:i386 libc6:i386 libcaca0:i386
```

그림 58 wine 패키지 설치

“<https://www.heidisql.com>”에서 HeidiSQL Installer를 다운로드 받는다. “Downloads” -> “Installer”를 선택하면 자동으로 exe 파일을 받아온다. 우분투의 Downloads에서 확인할 수 있다.



그림 59 HeidiSQL 설치 파일 다운로드

exe 파일이 존재하는 디렉터리 경로로 이동한 다음, wine 명령을 통해 설치를 진행한다. 설정은 모두 기본값으로 진행한다. 설치가 완료되면 프로그램을 실행시켜 본다.

```
wine [HeidiSQL.exe 파일이름]
```

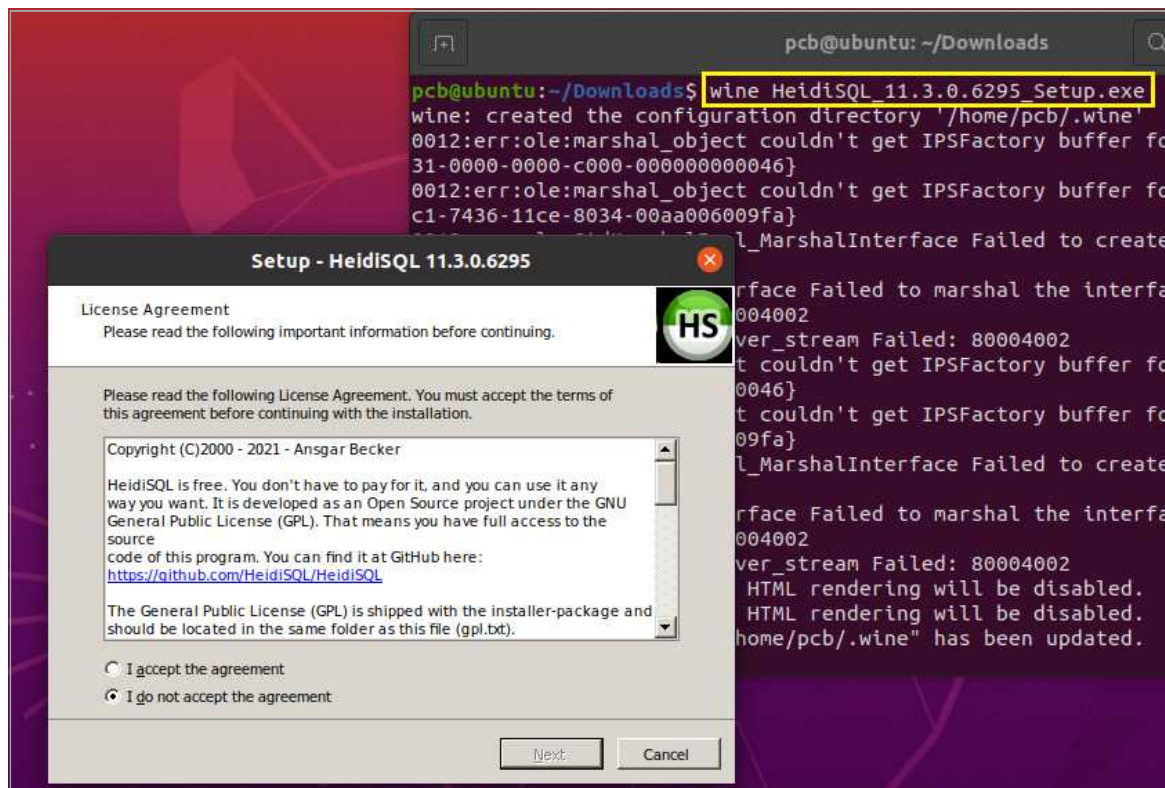


그림 60 HeidiSQL 설치 진행

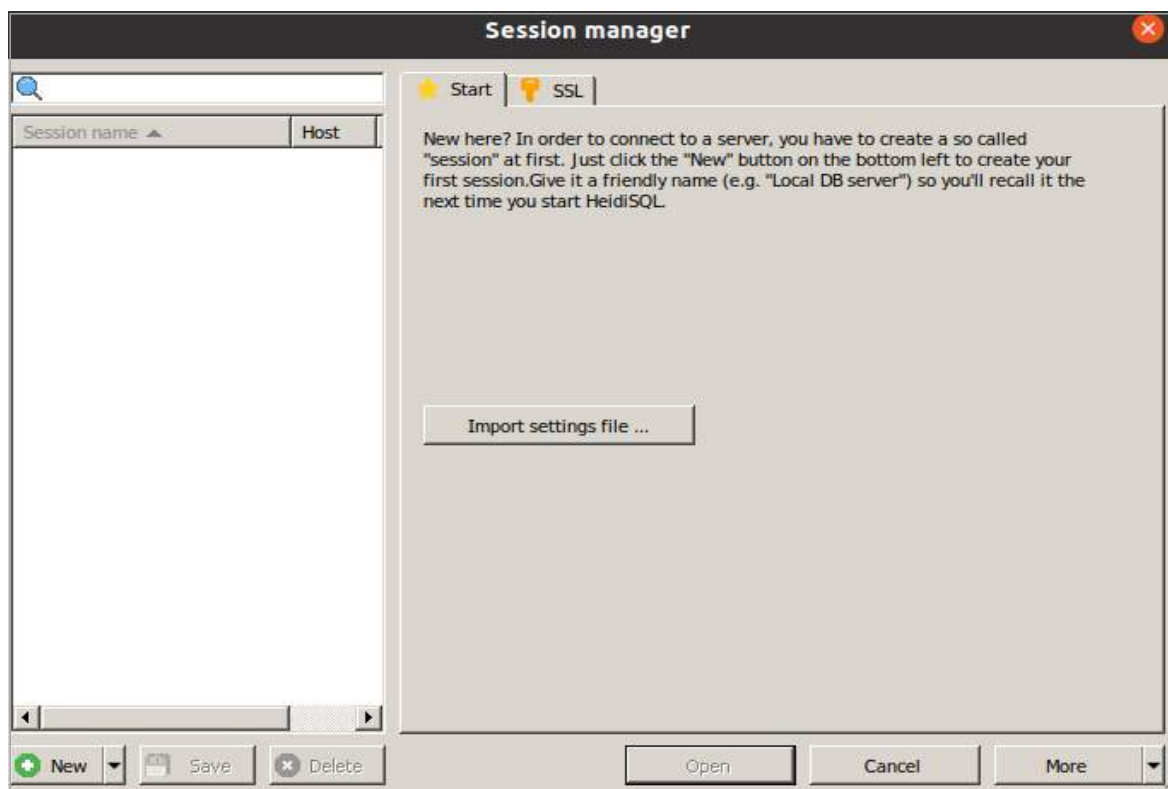


그림 61 HeidiSQL 첫 실행 화면

③ HeidiSQL에서 MariaDB 접속 및 데이터베이스&테이블 생성

모든 설치가 완료되었으며 마지막으로 MariaDB에 접속한다. 예시로 접속하기 위한 사용자 계정으로 는 위에서 테스트용으로 만들었던 "testuser"로 진행한다(그림 62). 본 프로젝트에서는 "hansuBot"@"localhost" 계정으로 수질 데이터를 저장하는 데이터베이스를 관리한다. 수질 데이터를 저장하는 데이터베이스는 "hansu", 테이블은 각각 "Device", "Sensor_Data"이다.

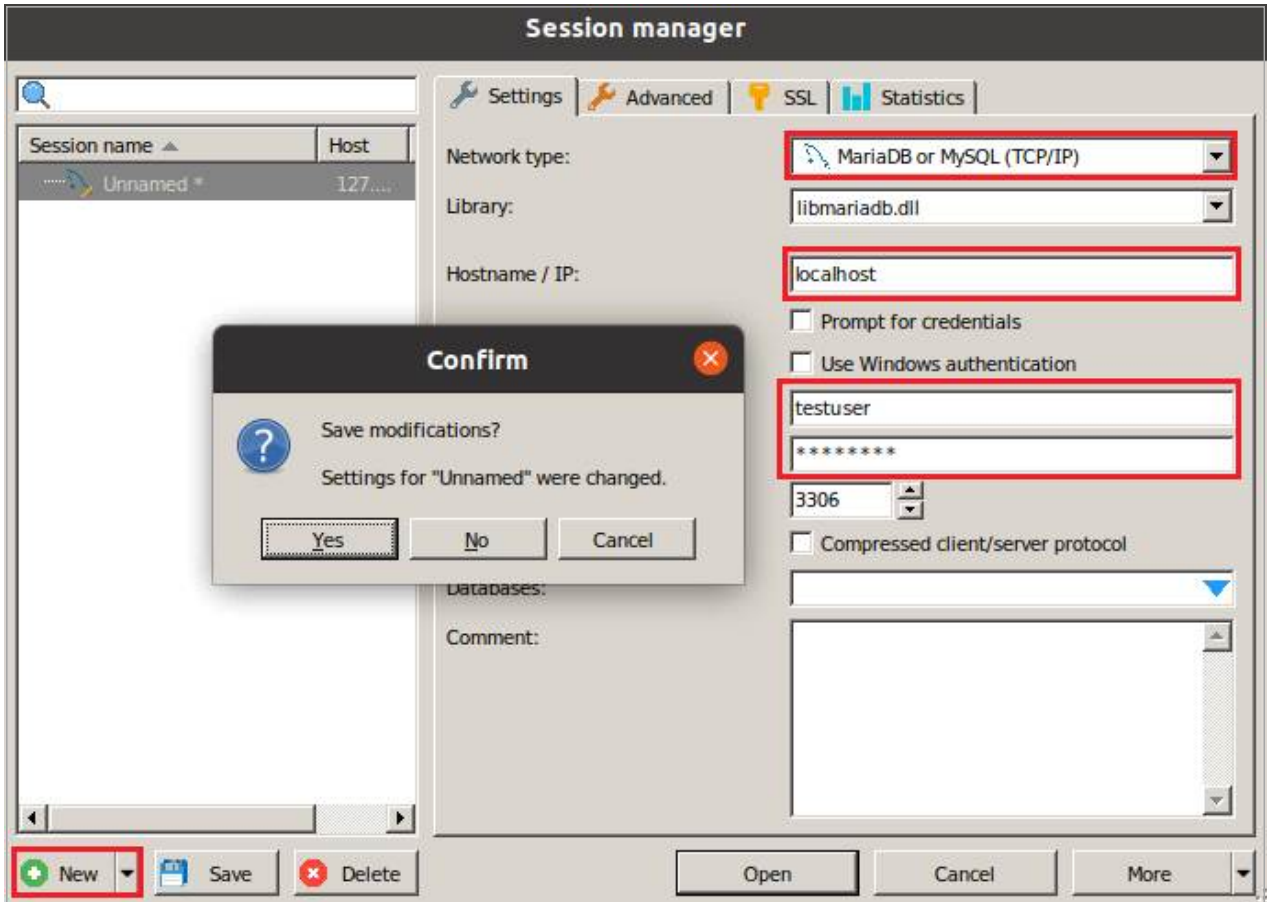


그림 62 MariaDB 접속 환경 설정

"New" 또는 "신규"를 선택하면 Session이 생성되며 우측에 접속할 데이터베이스와 계정을 입력할 수 있다. Network type을 MariaDB로 선택, Hostname/IP는 "testuser"@"localhost"에서 "localhost"부분에 해당한다. localhost는 동일 네트워크 내부에서만 접속 가능하며 127.0.0.1과 같다. 접속할 계정과 비밀번호를 입력하고 "Open"을 선택하여 접속을 시도한다. 그림 62에서는 변경 사항 수정에 대한 안내가 나오는데, "Yes"를 선택하여 진행한다.

"데이터베이스 생성 -> 테이블 생성 -> 데이터 열 구성" 순서로 진행하며 최종적으로 완성되는 수질 데이터 저장 데이터베이스는 그림 64와 같다. 테이블 구성과 함께 기본키 외래키 설정도 같이 진행한다.

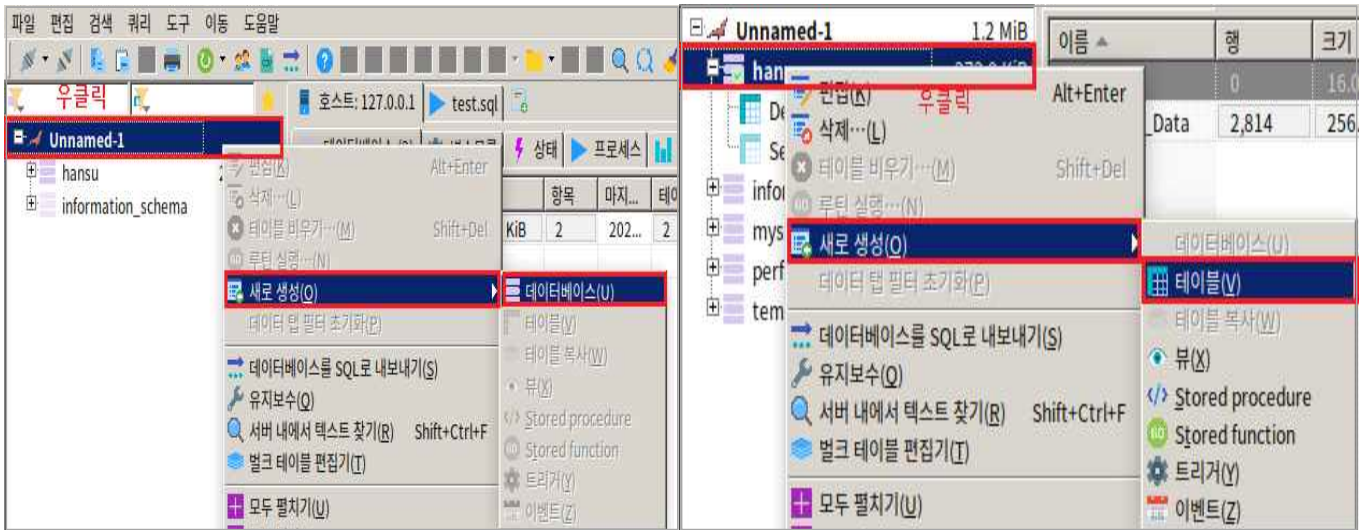


그림 63 데이터베이스 생성(좌측), 테이블 생성(우측)

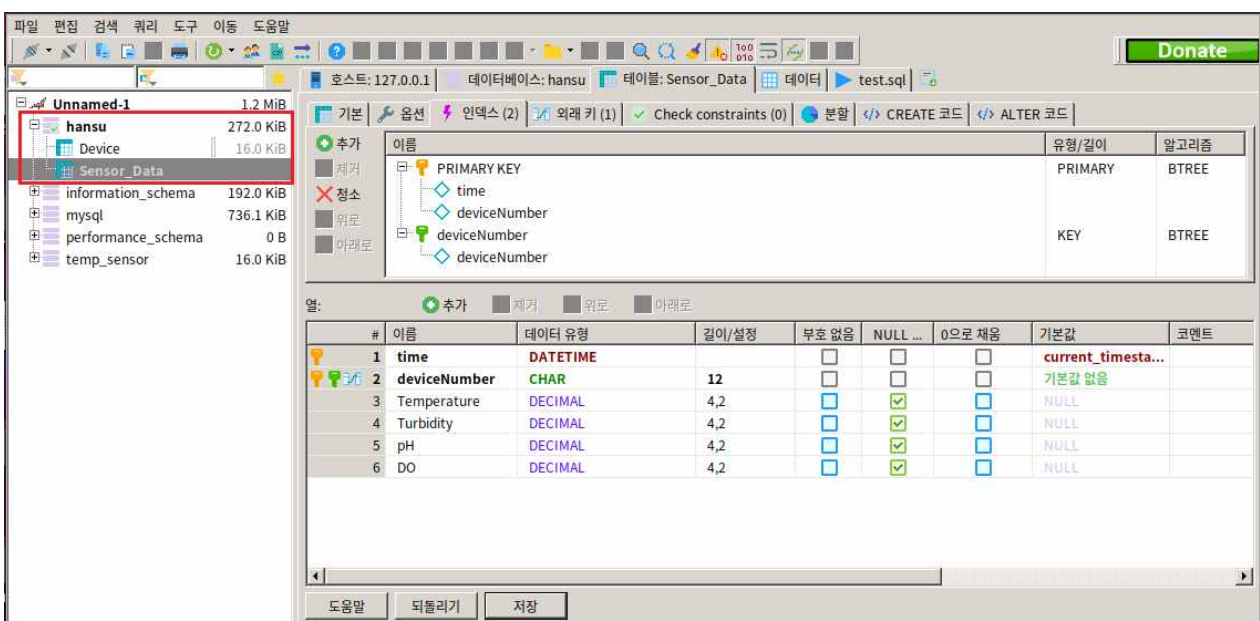
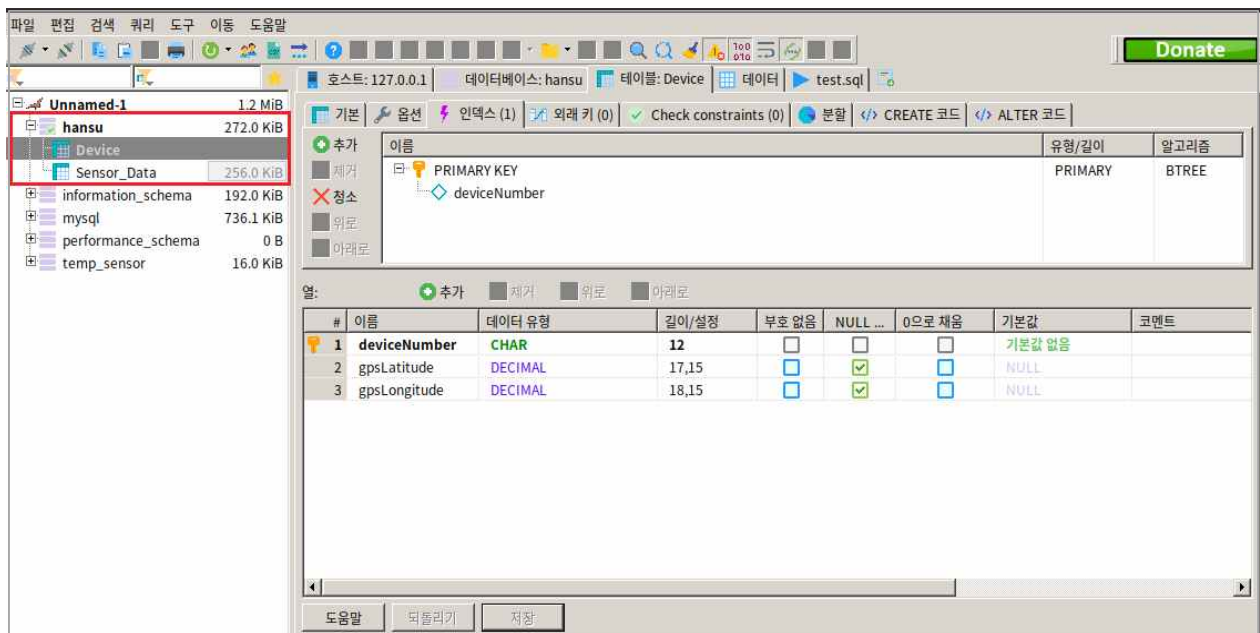


그림 64 수질 데이터 저장 테이블(Device, Sensor_Data) 구성

2-3. 라즈베리파이와 메인 서버 데이터베이스 연결

라즈베리파이와 메인 서버 2개의 단말을 연결하기 위해서는 소켓(Socket) 통신을 수행해야 한다. 소켓은 프로토콜(TCP), IP 주소, 포트(Port) 번호로 정의되며 3가지의 항목이 모두 일치하는 호스트와 연결할 수 있다. 소켓 통신 방식은 그림 65와 같다. Python에서는 socket 패키지를 통해 소켓 구현이 가능하다.

메인 서버의 경우에는 라즈베리파이와의 소켓 통신과 동시에 MariaDB와 연결되어야 한다. Python에서 MariaDB 연결에 사용되는 패키지로 pymysql이 있으며 pip install을 통해 설치 후 진행한다.

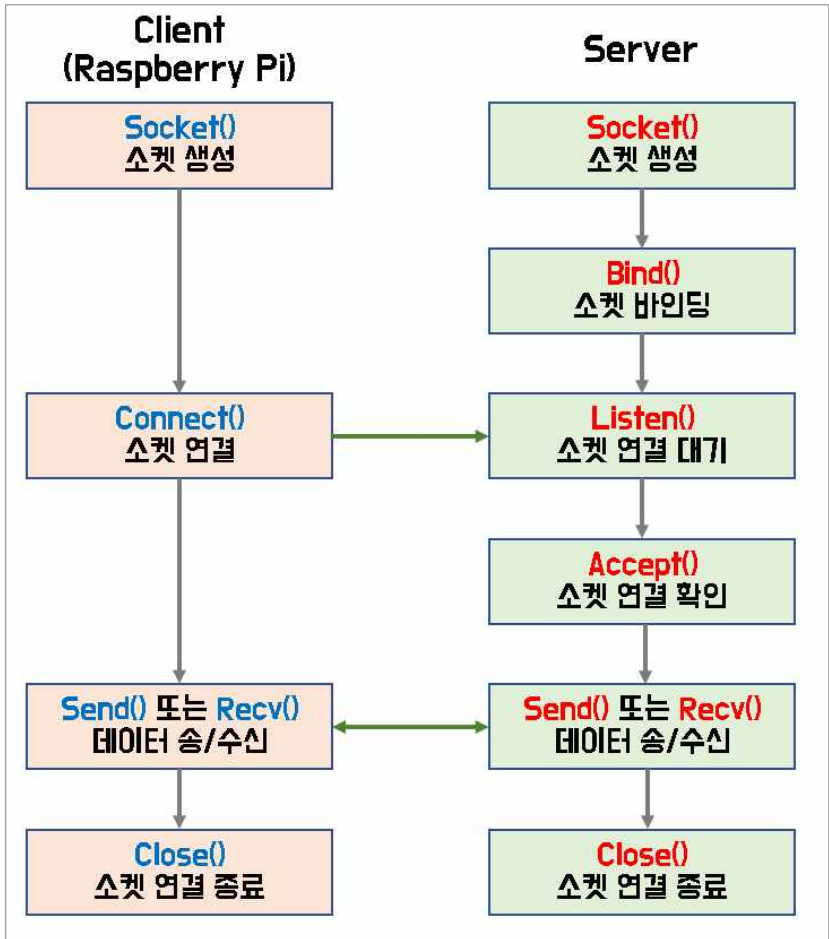


그림 65 소켓(Socket) 통신

```
pcb@ubuntu:~$ pip install pymysql
Defaulting to user installation because normal site-packages is not writeable
Collecting pymysql
  Downloading PyMySQL-1.0.2-py3-none-any.whl (43 kB)
    | 43 kB 321 kB/s
Installing collected packages: pymysql
Successfully installed pymysql-1.0.2
```

그림 66 메인 서버에 pymysql 패키지 설치

※ 라즈베리파이(Client) 소켓 통신 python3 - `client.py`

소켓 통신을 위한 코드와 수질 데이터를 메인 서버에 전달하는 코드를 작성한다. 수질 데이터 수집 코드는 “2-1 라즈베리파이 수질 측정 시스템 개발”에서 작성했던 코드를 활용한다.

```
import socket                                # socket Connect
import spidev                                # mcp3008 SPI
import time                                  # 시간 제어 관련
import sensor_temP_DS18B20 as temp          # 수온 측정
import sensor_DO as DO                      # 용존 산소량 측정
import serial                                # GPS 시리얼 통신 관련
import pynmea2                               # GPS NMEA 프로토콜 관련

spi = spidev.SpiDev()                        # SPI Device Instance(SPI)
spi.open(0, 0)                              # open(spi_bus, device_channel). CS=0
spi.max_speed_hz= 1350000                   # SPI speed (1350000hz) = 1.35MHz

PH_CHANNEL = 0                             # MCP3008 CH0
DO_CHANNEL = 1                             # MCP3008 CH1
TURB_CHANNEL = 7                           # MCP3008 CH7
OFFSET = 0.47                             # pH 교정 상수
TWO_CALIBRATION = 1                        # DO 측정 방식 상수
GPS_PORT = "/dev/ttyS0"                    # GPS 장치 파일 경로
BAUDRATE = 9600                            # GPS baud rate 상수

HOST = '192.168.3.4'                       # (Ubuntu) 메인 서버 IP
PORT = 5538                                # (Ubuntu) 메인 서버 개방 포트

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 소켓 객체 생성(ipv4, Byte Stream)
server.connect((HOST,PORT))                 # 서버에 연결 시도

DO_Table = [14600, 14220, 13800, 13440, 13080, 12760, 12440, 12110, 11830, 11560,
            11290, 11040, 10760, 10540, 10310, 10060, 9860, 9640, 9470, 9270,
            9090, 8910, 8740, 8570, 8410, 8250, 8110, 7960, 7830, 7680,
            7560, 7430, 7300, 7170, 7060, 6940, 6840, 6720, 6600, 6520,
            6400, 6330, 6230, 6130, 6060, 5970, 5880, 5790]

# MCP3008
def readadc(adc_channel):
    if ((adc_channel > 7) or (adc_channel < 0)):
        return -1
    r = spi.xfer2([1, (0x08 + adc_channel) << 4, 0])
    adc_out = ((r[1] & 0x03) << 8) + r[2]
    return adc_out
```

```

if __name__ == "__main__":
    while True:
        # 수온 데이터 측정
        temperature = temp.read_temp()
        print("CEL temperature =%.2f" %temperature)
        time.sleep(2)

        # 탁도 데이터 측정
        mcp_Turb = readadc(TURB_CHANNEL)
        Turb_volt = mcp_Turb * (3.3 / 1024)
        print("MCP3008 read value(Turbidity) : %d" %mcp_Turb)
        print("Turbidity : %.1lf" %Turb_volt)
        time.sleep(2)

        # pH 데이터 측정
        mcp_pH = readadc(PH_CHANNEL)
        pH_volt = mcp_pH * (3.3 / 1024)
        pH_value = pH_volt * 3.5 + OFFSET
        print("MCP3008 read value(pH) = %d " %mcp_pH)
        print("pH Voltage = %.2lf " %pH_volt)
        print (pH_value, "PH")
        time.sleep(2)

        # DO 데이터 측정
        mcp_DO = readadc(DO_CHANNEL)
        DO_volt = mcp_DO * (3.3 / 1024)
        print("MCP3008 read value(DO) : %d" %mcp_DO)
        print("DO voltage = %.2lf" %DO_volt)
        DO_value = DO.readDO(DO_volt, int(temperature))
        print(DO_value, " mg/L")

        # 측정한 4개 항목 데이터를 쉼표(,)로 구분하는 문자열 데이터로 저장
        data_table = str(temperature) + ',' + str(Turb_volt) + ',' + \
            str(pH_value) + ',' + str(DO_value)

        server.send(data_table.encode()) # 인코딩(encode) 작업 후 서버로 전달
        reply = server.recv(1024)       # 서버로부터 전달받은 내용 저장

        if reply.decode() == 'received': # 서버로부터 전달받은 내용이 "received"라면
            print("data sending success!")

        time.sleep(600)

```

※ 메인 서버(Server) 소켓 통신 python3 - `server.py`

소켓 통신을 위한 코드와 라즈베리파이로부터 전달받은 수질 데이터를 처리하는 코드, 처리한 데이터를 MariaDB의 데이터베이스 테이블에 저장하는 코드를 작성한다.

```
import socket
import pymysql    # MariaDB
import time

DEVICE = '202105191705'    # 수질 측정 장비 고유 번호
HOST = '192.168.3.4'        # 메인 서버(Server) IP 주소
PORT = 5538                # 메인 서버에서 개방할 포트 번호

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    # 소켓 객체 생성(ipv4, Byte Stream)
    # 일반 소켓 레벨 설정, 이미 사용 중인 주소&포트에 대해 바인드 허용
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print("Socket creation complete!")

    # 서버 IP와 PORT 결합 확인 (예외처리 적용)
try:
    server.bind((HOST, PORT))    # 소켓 바인드
except socket.error:
    print("Bind failed")
    exit()

server.listen()                # 클라이언트 접속 허용
print("Socket waiting for client messages")
(client_socket, addr) = server.accept()    # 클라이언트가 접속하면 새로운 소켓 반환
print(f"client_socket connected = {addr}")

    # MariaDB에 연결하기 위한 주소 및 사용자 계정, 연결할 데이터베이스, 문자 포매팅 값
connect_maria = pymysql.connect(host='127.0.0.1', user='hansuBot',\
    password='dfg09787', db='hansu', charset='utf8' )

cur = connect_maria.cursor()    # MariaDB 커서 (SQL문 적용에 사용)

    # 데이터베이스에 Sensor_Data 테이블이 존재하지 않는다면 sql을 통해 생성. (저장된 이후에는 적용 안 됨)
sql = 'CREATE TABLE IF NOT EXISTS Sensor_Data (time DATETIME, deviceNumber CHAR(12),
    Temperature DECIMAL(4, 2), Turbidity DECIMAL(4, 2), pH DECIMAL(4, 2), DO DECIMAL(4, 2))'

cur.execute(sql)                # 커서를 통한 sql 실행

connect_maria.commit()    # MariaDB에서 수행한 작업이나 SQL 반영
```

```

# 클라이언트(라즈베리파이)와 연결 되었을 때 수행하는 과정
try:
    while True:
        data = client_socket.recv(1024)          # 클라이언트로부터 데이터 수신
        data = data.decode()                     # 수신한 데이터 디코딩(decode)
        data_table = data.split(',')             # 쉼표(.)를 기준으로 구분하는 리스트 생성
        print("Sensor Data Received " + data)     # 예비 출력
        reply = 'received'
        client_socket.send(reply.encode())        # 클라이언트에 수신 완료 전달

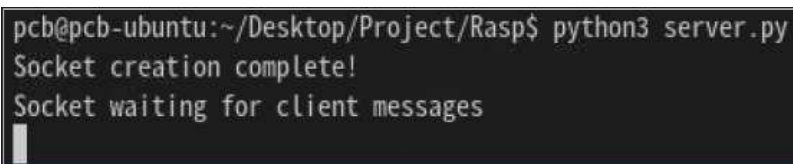
        print("All Data Signal GOOD! Save DB...")
        # 수집 시간 포맷(MariaDB의 Sensor_Data 테이블에 저장할 목적)
        local_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        # Sensor_Data에 수집 데이터를 저장하는 SQL
        sql = "INSERT INTO Sensor_Data VALUES('" + local_time + "','" + DEVICE + "','" +
data_table[0] + "','" + data_table[1] + "','" + data_table[2] + "','" + data_table[3] + "')"
        print(sql)
        cur.execute(sql)                        # sql 문 실행
        connect_maria.commit()                  # 데이터베이스에 반영
        print("Send Success")
except KeyboardInterrupt:                      # 키보드 강제 종료 입력(ctrl+c, ctrl+z) 처리
    print("DB Server Stop to save")
    exit()
finally:                                       # 서버 연결 종료 시
    connect_maria.close()
    client_socket.close()
    server.close()

```

※ 실행 결과

프로그램을 실행하기 전에 메인 서버에서 연결을 위한 포트를 개방해야 한다. 본 코드에서는 5538번을 포트 번호로 지정했다. 다음 명령을 통해 포트를 개방한다. 이후에 프로그램을 실행한다.

```
sudo iptables -I INPUT 1 -p tcp --dport 5538 -j ACCEPT
```



```

pcb@pcb-ubuntu:~/Desktop/Project/Rasp$ python3 server.py
Socket creation complete!
Socket waiting for client messages

```

그림 67 메인 서버 연결 대기

```

pi@raspberrypi:~/Desktop/Project_file $ python3 client.py
CEL temperature =26.75
MCP3008 read value(Turbidity) : 876
Turbidity : 2.8
MCP3008 read value(pH) = 507
pH Voltage = 1.63
6.188603515624999 PH
MCP3008 read value(DO) : 162
DO voltage = 0.52
2.597540021088957 mg/L
data sending success!

```

그림 68 라즈베리파이 수질 데이터 전달 (data sending success!)

```

pcb@pcb-ubuntu:~/Desktop/Project/Rasp$ python3 server.py
Socket creation complete!
Socket waiting for client messages
client_socket connected = ('192.168.3.19', 51246)
Sensor Data Received 26.75,2.8230468749999997,6.188603515624999,2.597540021088957
All Data Signal GOOD! Save DB...
INSERT INTO Sensor_Data VALUES('2021-07-01 09:22:33','202105191705','26.75','2.8230468749999997','6.188603515624999','2.597540021088957')
Send Success

```

그림 69 클라이언트 연결 확인 및 수신된 수질 데이터를 데이터베이스에 저장

Unnamed-1

hansu

288.0 KiB

Device

16.0 KiB

Sensor_Data

information_schema

mysql

performance_schema

temp_sensor

hansu.Sensor_Data: 3,167 행 (총) (대략적)

time

deviceNumber

Temperature

Turbidity

pH

DO

2021-06-16 09:56:02	202105191705	24.44	2.82	6.22	3.25
2021-06-16 10:01:09	202105191705	24.44	2.80	6.26	2.92
2021-06-16 10:06:15	202105191705	24.38	2.83	6.22	2.97
2021-06-16 10:11:22	202105191705	24.44	2.81	6.24	2.80
2021-06-16 10:12:04	202105191705	24.50	2.77	6.22	2.90
2021-06-16 10:19:21	202105191705	24.38	2.83	6.19	3.30
2021-06-16 10:24:28	202105191705	24.38	2.81	6.24	3.14
2021-06-16 10:29:35	202105191705	24.38	2.83	6.21	3.23
2021-06-16 10:34:42	202105191705	24.38	2.83	6.31	3.06
2021-06-16 10:39:49	202105191705	24.38	2.85	6.24	2.87
2021-06-16 10:44:56	202105191705	24.38	2.82	6.27	2.95
2021-06-16 10:50:03	202105191705	24.44	2.83	6.24	2.99
2021-06-16 10:55:10	202105191705	24.44	2.81	6.28	3.28
2021-06-16 11:00:17	202105191705	24.31	2.82	6.26	3.16
2021-06-16 11:05:24	202105191705	24.31	2.82	6.34	2.80
2021-06-16 11:10:31	202105191705	24.38	2.81	6.28	3.16
2021-07-01 09:01:47	202105191705	26.56	2.88	6.26	2.63
2021-07-01 09:06:53	202105191705	26.75	2.86	6.22	2.57
2021-07-01 09:12:00	202105191705	26.81	2.87	6.31	2.65
2021-07-01 09:17:07	202105191705	26.75	2.86	6.22	2.63
2021-07-01 09:22:33	202105191705	26.75	2.82	6.19	2.60

그림 70 HeidiSQL로 저장된 수질 데이터 확인

2-4. 수질 데이터 측정 웹 서비스 구현

측정된 수질 데이터를 웹 서비스를 통해 시각화된 자료로 보여준다. 구현을 위해서 Node.js를 사용한다. 웹 페이지 제작에는 데이터베이스의 데이터를 활용하기 수월하도록 자바스크립트 코드를 사용할 수 있는 포맷인 ejs(Embedded JavaScript)를 사용한다. 서비스 항목은 총 4가지이며 실시간 수질 데이터 측정 차트 테이블, 누적된 수질 데이터의 통계 차트 테이블, 수질 측정 장비가 설치된 장소, 해당 장소의 날씨이다. 설치된 장소를 표현하기 위해서 카카오맵 API를 사용하고, 날씨 정보를 표현하기 위해서 OpenWeather API를 사용한다. 그림 71은 페이지를 간단하게 구상한 것이다.



그림 71 웹 페이지 구상도

① Node.js 설치 및 환경 설정

Node.js는 패키지 매니저로 npm을 사용한다. 따라서 서버 구현을 위한 Node.js와 패키지 매니저인 npm을 설치한다. 설치 완료 후에는 버전을 확인하여 정상적으로 설치가 되었는지 판단한다.

```
sudo apt install nodejs && sudo apt install npm  
npm -v && nodejs -v
```

```
pcb@ubuntu:~/Desktop/hansu$ npm -v && nodejs -v  
6.14.4  
v10.19.0
```

그림 72 Node.js, npm 패키지 설치 확인

Node.js 프로젝트를 수행할 디렉터리를 만들고 npm init 명령으로 서버 설정을 진행한다. 최종적으로 "package.json"파일이 생성된다. 명령 수행 시 몇 가지 질의가 나타나며 이후에 package.json 파일에서 수정할 수 있어서 기본값으로 진행한다.

```
pcb@ubuntu:~/Desktop/hansu$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (hansu)
version: (1.0.0)
description: Water Project
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/pcb/Desktop/hansu/package.json:

{
  "name": "hansu",
  "version": "1.0.0",
  "description": "Water Project",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)

New major version of npm available! 6.14.4 -> 7.19.0
Changelog: https://github.com/npm/cli/releases/tag/v7.19.0
Run npm install -g npm to update!
```

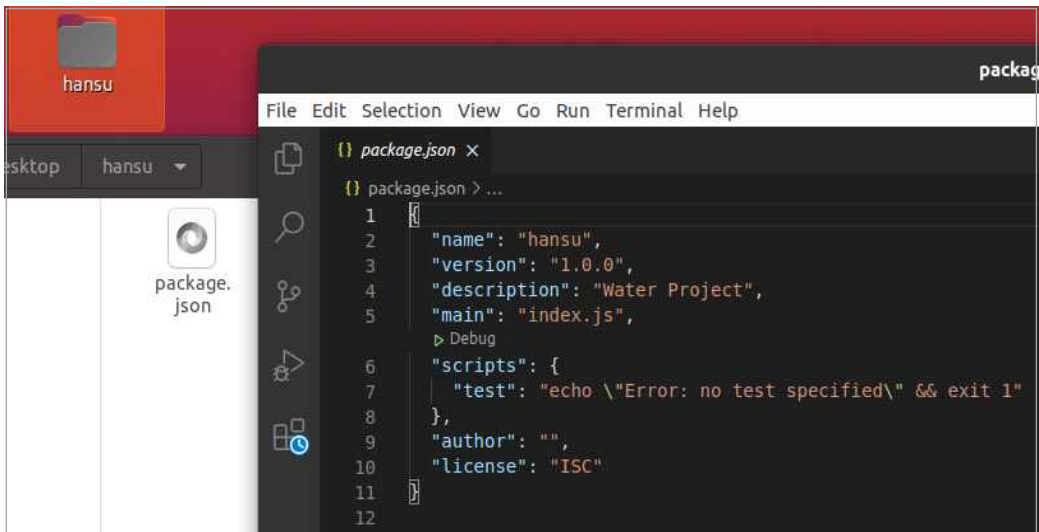


그림 73 npm init 수행 결과

이후 작업 디렉터리 “name”은 “hansu_server”, “main”은 “main.js”로 수정한다. Node.js에서 필요한 외부 라이브러리 모듈들은 npm install 명령을 통해서 받아올 수 있다. 다른 디렉터리에서도 호환 가능한 전역으로 받아오는 방법과 작업 디렉터리에서만 가능한 의존성으로 받아오는 방법이 있다. 본 프로젝트에서는 의존성으로 받아오는 방법을 사용한다.

Node.js에서 웹 서버를 구축하는데 반드시 필요한 모듈은 “express.js”가 있다. 웹 프레임워크이며 웹 애플리케이션 및 API 개발을 목적으로 설계되었다. 웹 애플리케이션을 만들었다면 외부 사용자로부터 호출되어야 하고, 페이지를 보여줄 수 있어야 한다. 즉, 웹 소켓을 통한 연결이 이루어져야 한다. 이는 라즈베리파이-서버 연결에서 사용했던 방법이며 코드만 python3에서 JavaScript로 변경된 것이다. 이를 위해서 “socket.io” 패키지가 필요하다. 웹 페이지 제작에는 ejs 포맷을 사용할 것이기 때문에 “ejs” 패키지를 받아온다. 또한, 웹 페이지에 사용될 데이터베이스 정보를 가져오기 위해 MariaDB에 연결할 수 있도록 “mysql” 패키지를 받아온다. “nodemon” 패키지는 서버 구축에 사용되는 파일 (main.js)이 수정되었을 때 서버를 자동으로 재부팅하여 수정 내용을 반영해주는 역할을 한다. 수정하고서 수동으로 일일이 restart하는 번거로움을 없애준다. 이렇게 총 5가지의 패키지 모듈들을 받는다. 개발 과정에서 필요한 패키지들은 추후에 다운로드 받으면서 진행한다.

작업 디렉터리에서 “package-lock.json”파일과 “node_modules” 디렉터리가 생성되는 것을 확인할 수 있다. 또한 package.json의 “dependencies”에서 패키지가 추가된 것을 확인할 수 있다.

```
npm install --save socket.io
npm install --save nodemon
npm install --save express
npm install --save ejs
npm install --save mysql
```

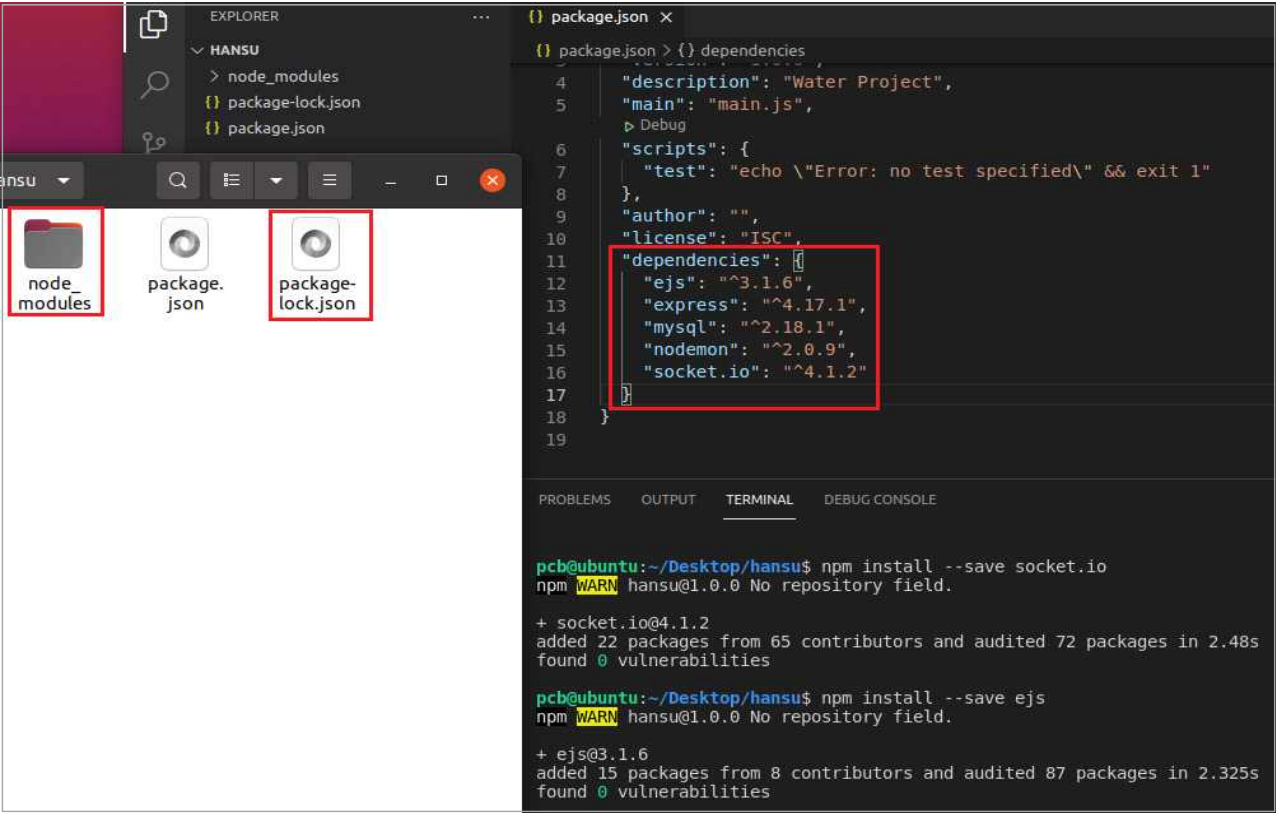


그림 74 Node.js 관련 패키지 다운로드

② main.js 서버 실행 파일 코드

main.js 파일의 역할은 웹 소켓을 통해 클라이언트 호출(웹 페이지를 보여달라는 신호)을 받으며 해당 클라이언트에게 웹 페이지를 보여준다. 또한, MariaDB에 접속하여 웹 페이지에 사용될 데이터를 SQL로 추출하여 가져오고, 해당 데이터를 목적에 맞게 웹 페이지에 전달한다.

```
const express = require('express');    // express 웹 프레임워크
const socket = require('socket.io');    // 웹 소켓
const w_request = require('request')    // 날씨 API 호출
const http = require('http');           // http 모듈
const fs = require('fs');               // file system 모듈
const app = express();                 // express 객체
const mysql = require('mysql');         // mysql 모듈, mariadb 접근 가능
const ejs = require("ejs");             // html + javascript 호환 ejs 포맷

    // ejs 경로 및 엔진 설정
app.set('views', './views');
app.set('view engine', 'ejs');

    // 정적 선언해서 서버에 연결 (css, image)
app.use(express.static('./static'));

const hostname = '192.168.3.4';         // (Ubuntu) 메인 서버 IP
const port = 9787;                     // 메인 서버에서 개방할 포트 번호
const server = http.createServer(app);  // 서버 객체 생성
const io = socket(server)               // 소켓 바인딩

    // OpenWeather API JSON 추출 경로
const w_addr =
"http://api.openweathermap.org/data/2.5/weather?lat=35.911111&lon=128.81074&appid=5ed1005c16c91ae6270fb1d93a1e14a6&units=metric"

    // MariaDB connection
var conn = mysql.createConnection({
  host: '127.0.0.1',
  user: 'hansuBot',
  password: 'dfg09787',
  database: 'hansu',
  multipleStatements: true              // 다중 쿼리
});
conn.connect();
```

```

var data;          // sql 결과 저장
var avgData;       // avgSQL 결과 저장
var timeAvgData;   // timeAvgSQL 결과 저장
    //실시간 최근 6개의 데이터 추출
var sql = 'select * from Sensor_Data ORDER BY time DESC LIMIT 6;';
    // 일 평균 데이터 추출
var avgSQL = 'SELECT DATE_FORMAT(TIME,"%Y-%m-%d") AS daytime, round(avg(Temperature), 2) as
avg_temp, round(avg(pH), 2) as avg_ph, round(avg(DO), 2) as avg_do, round(avg(Turbidity), 2) as
avg_turbidity FROM Sensor_Data GROUP BY daytime ORDER BY daytime DESC LIMIT 11;';
    // 시간 평균 데이터 추출
var timeAvgSQL = 'SELECT DATE_FORMAT(TIME,"%Y-%m-%d") AS daytime, DATE_FORMAT(TIME, "%H")
AS dayhour, round(avg(Temperature), 2) as avg_temp, round(avg(pH), 2) as avg_ph, round(avg(DO), 2)
as avg_do, round(avg(Turbidity), 2) as avg_turbidity FROM Sensor_Data GROUP BY daytime, dayhour
ORDER BY daytime DESC, dayhour DESC LIMIT 12;';

    // 주기적 반복을 통한 데이터 최신화
setInterval(function() {
    conn.query(sql + avgSQL + timeAvgSQL, function(err, results, fields){
        if(err){
            console.log(err);
        } else {
            // sql 결과를 data, avgData, timeAvgData에 저장
            data = results[0];
            avgData = results[1];
            timeAvgData = results[2];
        }
    });
}, 10000);          // 밀리세컨. 1초 = 1000

// OpenWeather API Request
var w_condition;   // 날씨 상태
var w_data;        // 날씨 데이터(온도 습도 등)
setInterval(function() {
    w_request(w_addr, function(error, res, body){
        if(error) {
            console.log(error);
        } else {
            // API에서 받아온 json 형식 파일을 파싱하여 목적에 맞는 데이터를 추출
            var w_obj = JSON.parse(body);
            w_condition = w_obj["weather"][0];
            w_data = w_obj["main"];
            w_data.location = w_obj["name"];
            w_data.wind = w_obj["wind"]["speed"];
        }
    });
}, 10000);          // 밀리세컨. 1초 = 1000

```



```

#####
//      페이지 호출 시 실행.
//      저장한 데이터들을 담아 index.ejs에서 사용할 수 있도록 전달
#####
app.get('/', function(require, response){
    // console.log(data)
    response.render('index', {data_obj: data, avgData_obj: avgData, timeAvgData_obj: timeAvgData,
w_condition_obj: w_condition, w_data_obj: w_data});
});

// 클라이언트 연결 대기
server.listen(9787, hostname, function(){
    console.log("ejs server ON");
    console.log(`http://${hostname}:${port}`);
});

```

코드에서 ejs 포맷을 적용하기 위해서 ejs 엔진을 적용했다. “app.set('views', './views');”, “app.set('view engine', 'ejs');” Node.js 페이지로 사용할 ejs 파일의 경로로 ./views로 한다는 의미이다. 또한, “app.use(express.static('./static'));”를 통해서 ejs 파일에 적용할 정적 자료들을 ./static에 저장해서 사용한다. 따라서, 최종적인 파일 구조는 다음과 같다.

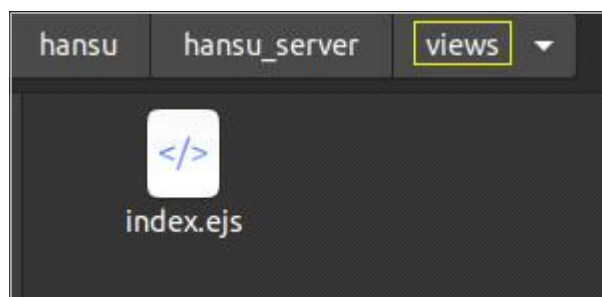
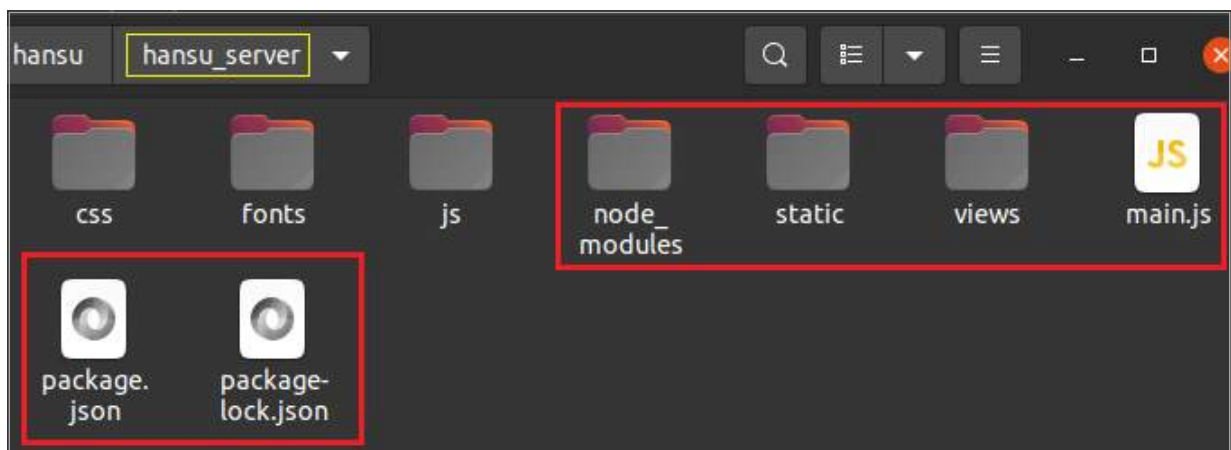


그림 75 Node.js 서버 파일 구조

③ index.ejs 페이지 구현 코드

그림 75에서 ./views 디렉터리에 위치한 index.ejs 파일은 웹 페이지를 구현한다. ejs는 HTML 코드에서 JavaScript를 호환하여 적용하기 수월하다. CSS로 Bootstrap을 적용하여 페이지 디자인을 간편하게 하고 코드의 가독성을 높인다. 수질 데이터 시각화에 사용되는 차트 테이블은 Chart.js를 사용한다. 각각의 라이브러리는 오픈소스로 제공하는 CDN(Content Delivery Network)을 활용한다. 수질 측정 장비의 설치 위치를 표현하는 지도는 카카오맵 API를 사용하고, 설치 위치의 날씨를 OpenWeather API를 사용한다.

API를 사용하기 위해서는 인증 키를 발급받아야 한다. 카카오 디벨로퍼(Kakao Developer)와 OpenWeather에 접속하여 인증 키를 발급받고 코드에 적용한다. 카카오맵의 경우 플랫폼 등록까지 완료해야 지도가 정상적으로 나타난다. 본 프로젝트는 내부 네트워크에서 [메인 서버 IP:포트번호]의 경로로 접속 가능한 상태에서 진행했다.

Chart.js CDN: <https://chartjs-plugin-datalabels.netlify.app/guide/getting-started.html#integration>

Bootstrap CDN: <https://getbootstrap.com/docs/4.6/getting-started/introduction/>

Kakao Developer: <https://developers.kakao.com/>

OpenWeather API: <https://openweathermap.org/current>

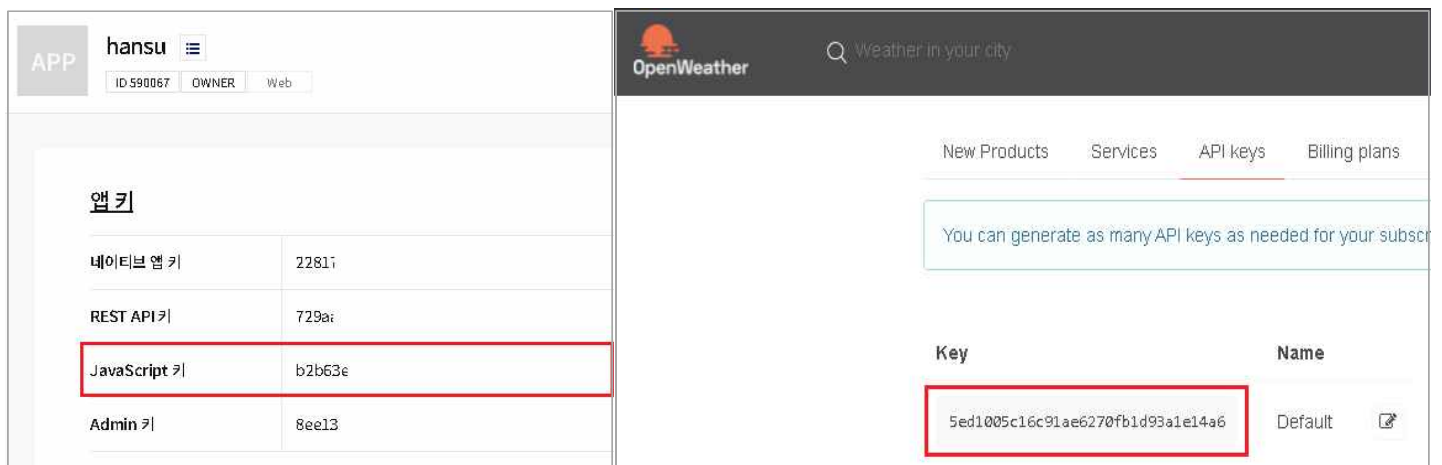


그림 76 Open API 인증 키 발급

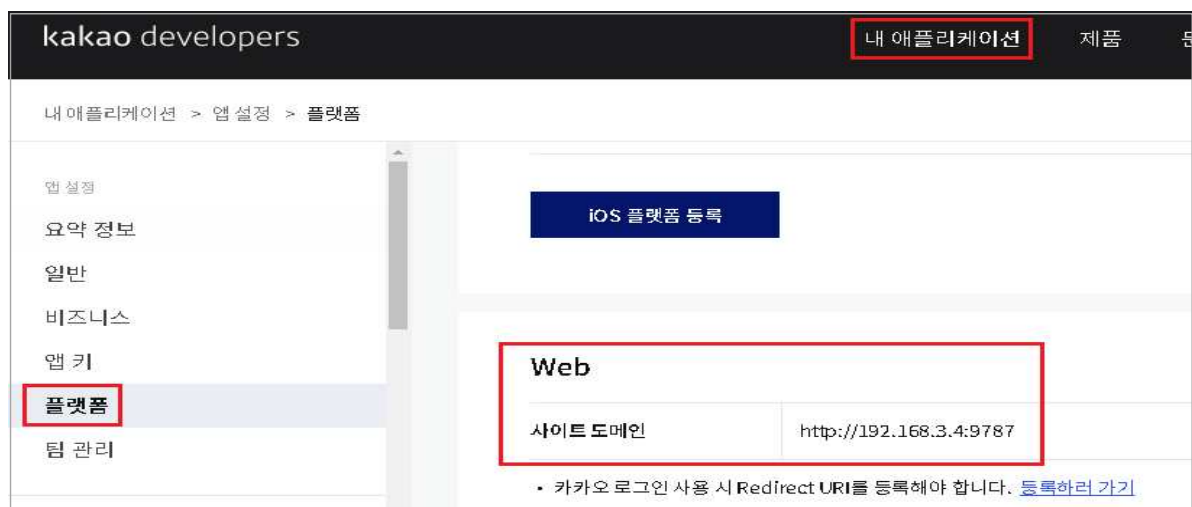


그림 77 카카오맵 플랫폼 등록

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset = "UTF-8">
  <title>한수앞선 Project</title>
  <!--#####-->
  <!-- bootstrap, JQuery, Web Css CDN -->
  <!--#####-->
  <link rel="stylesheet" href="/site.css">
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@2.9.4/dist/Chart.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-datalabels@0.7.0"></script>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"
    integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+I"
    crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/js/bootstrap.min.js"
    integrity="sha384-YQ4JLhgyBLPDQt//I+STsc9iw4uQqACwlvpslubQzn4u2UU2UFM80nGisd026JF"
    crossorigin="anonymous">
  </script>
</head>
<body>
  <div class="container">
    <!-- #####-->
    <!-- Page Header Part-->
    <!-- #####-->
    <div class="row header">
      <div class="col-sm-5 col-lg-5 logopart">
        
        <h5 class="text-darkblue text-font">컴퓨터소프트웨어학부 한수앞선</h5>
      </div>
      <div class="col-sm-7 col-lg-7 header_name">
        <h1 class="text-dark">Water Monitoring System</h1></br>
      </div>
    </div>
    <!-- #####-->
    <!-- Last measured time Part -->
    <!-- #####-->
    <div class="row justify-content-end">
      <h6>마지막 측정 시간 : <span id="lastTime"></span> </h6>
      <script>
        var timeStamp = "<%=data_obj[0].time%>";
        console.log(timeStamp);
        var endPoint = timeStamp.indexOf("G");
        var lastTime = timeStamp.substring(0,endPoint);
        document.getElementById("lastTime").innerHTML=lastTime;
      </script>
    </div>
  </div>
</body>
</html>
```

```

</div>
<!-- #####-->
<!--           Live Data Part           -->
<!-- #####-->
</br>
<div class="row mb-3">
  <div class="col-lg-3 col-sm-6 text-center">
    <h4>온
도</h4>

    <hr>
    <h4><%= data_obj[0].Temperature %> °C</h4>
  </div>
  <div class="col-lg-3 col-sm-6 text-center">
    <h4>수소이온농도
</h4>

    <hr>
    <h4><%= data_obj[0].pH %> pH</h4>
  </div>
  <div class="col-lg-3 col-sm-6 text-center">
    <h4>용존산소량
(DO)</h4>

    <hr>
    <h4><%= data_obj[0].DO %> mg/L</h4>
  </div>
  <div class="col-lg-3 col-sm-6 text-center">
    <h4>탁도
</h4>

    <hr>
    <h4> <%= data_obj[0].Turbidity %> </h4>
    <p> 탁함(0) ~ 맑음(3.3)</p>
  </div>
</div>
<!-- #####-->
<!--           센서 데이터 배열 수집 부분           -->
<!-- #####-->
<script>
  var cnt = 6; // 6개 JSON 객체
  var timeArray = [];
  var tempArray = [];
  var phArray = [];
  var turbiArray = [];
  var doArray = [];
  var dateArray = [];
  var tempAvgArray = [];
  var turbiAvgArray = [];
  var phAvgArray = [];

```

```

var doAvgArray = [];
var hourArray = [];
var timeTempAvgArray = [];
var timeTurbiAvgArray = [];
var timePhAvgArray = [];
var timeDoAvgArray = [];

//실시간 데이터 저장
"<% for(var data of data_obj) { %>"
var timeStamp = "<%=data.time%>";
// console.log(timeStamp);
var startTime = timeStamp.indexOf(":")-2;
var endTime = timeStamp.indexOf("G", startTime+6);
timeArray.push(timeStamp.substring(startTime, endTime));
tempArray.push("<%= data.Temperature%>");
phArray.push("<%= data.pH%>");
turbiArray.push("<%= data.Turbidity%>");
doArray.push("<%= data.DO%>");
"<% } %>"

// 일 평균 데이터 배열 저장
"<% for(var avgData of avgData_obj) { %>"
dateArray.push("<%= avgData.daytime%>");
tempAvgArray.push("<%= avgData.avg_temp%>");
turbiAvgArray.push("<%= avgData.avg_turbidity%>");
phAvgArray.push("<%= avgData.avg_ph%>");
doAvgArray.push("<%= avgData.avg_do%>");
"<% } %>"

// 시간 평균 데이터 배열 저장
"<% for(var timeAvgData of timeAvgData_obj) { %>"
timeTempAvgArray.push("<%= timeAvgData.avg_temp%>");
timeTurbiAvgArray.push("<%= timeAvgData.avg_turbidity%>");
timePhAvgArray.push("<%= timeAvgData.avg_ph%>");
timeDoAvgArray.push("<%= timeAvgData.avg_do%>");
hourArray.push("<%= timeAvgData.dayhour%>");
"<% } %>"

timeArray.reverse();
tempArray.reverse();
phArray.reverse();
turbiArray.reverse();
doArray.reverse();
dateArray.reverse();
tempAvgArray.reverse();
turbiAvgArray.reverse();

```



```

    phAvgArray.reverse();
    doAvgArray.reverse();
    timeTempAvgArray.reverse();
    timeTurbiAvgArray.reverse();
    timePhAvgArray.reverse();
    timeDoAvgArray.reverse();
    hourArray.reverse();
</script>
<div class="row">
    <div class="col-lg-3 col-sm-6">
        <canvas id="myChart1" height="220px"></canvas>
    </div>
<script>
    var ctx = document.getElementById('myChart1').getContext('2d');
    var chart = new Chart(ctx, {
        type: 'line',
        data: { labels: timeArray,
        datasets: [{ label: '온도 (Temperature)', backgroundColor: 'transparent',
        borderColor: 'green', data: tempArray}] },
        // 옵션
        options: {
            scales: {
                yAxes: [{
                    ticks: {
                        reverse: false,
                        stepSize: 0.08
                    },
                ]
            },
            plugins: {
                datalabels: {
                    display: false
                },
            }
        }
    } });
</script>
<div class="col-lg-3 col-sm-6">
    <canvas id="myChart2" height="220px"></canvas>
</div>
<script>
    var ctx = document.getElementById('myChart2').getContext('2d');
    var chart = new Chart(ctx, {
        type: 'line',
        data: { labels: timeArray,
        datasets: [{ label: '수소이온농도 (pH)', backgroundColor: 'transparent',
        borderColor: 'blue', data: phArray }] },
        options: {

```

```

        scales: {
            yAxes: [{
                ticks: {
                    reverse: false,
                    stepSize: 0.1
                },
            }]
        },
        plugins: {
            datalabels: {
                display: false
            }
        }
    } }));
</script>
<div class="col-lg-3 col-sm-6">
    <canvas id="myChart3" height="220px"></canvas>
</div>
<script>
    var ctx = document.getElementById('myChart3').getContext('2d');
    var chart = new Chart(ctx, {
        type: 'line',
        data: { labels: timeArray,
        datasets: [{ label: '용존산소량 (DO)', backgroundColor: 'transparent',
        borderColor: 'orange', data: doArray }] },
        options: {
            scales: {
                yAxes: [{
                    ticks: {
                        reverse: false,
                        stepSize: 0.2
                    },
                }]
            },
            plugins: {
                datalabels: {
                    display: false
                }
            }
        }
    } }));
</script>
<div class="col-lg-3 col-sm-6">
    <canvas id="myChart4" height="220px"></canvas>
</div>
<script>
    var ctx = document.getElementById('myChart4').getContext('2d');
    var chart = new Chart(ctx, {

```

```

        type: 'line',
        data: { labels: timeArray,
        datasets: [{ label: '탁도 (Turbidity)', backgroundColor: 'transparent',
        borderColor: 'purple', data: turbiArray }] },
        options: {
            scales: {
                yAxes: [{
                    ticks: {
                        reverse: false,
                        stepSize: 0.2
                    },
                }]
            },
            plugins: {
                datalabels: {
                    display: false
                }
            }
        }
    } });
</script>
</div>
<!-- #####-->
<!--     하루 누적 데이터 평균 차트         -->
<!-- #####-->
<div class="row mt-5">
    <div>
        </br><h4> 누
적 데이터</h4>
        </br>
        <form>
            Day      <input      type="radio"      name="chartselect"      id="dayButton"
onclick="showDayButton();" value="day" checked>
            Hour     <input      type="radio"      name="chartselect"      id="timeButton"
onclick="showTimeButton();" value="time">
        </form>
    </div>
    <div class="col-12">
        <canvas id="myChart5"></canvas>
        <script>
            var ctx = document.getElementById('myChart5').getContext('2d');
            function showDayButton() {
                var chart = new Chart(ctx, {
                    type: 'line',
                    data: { labels: dateArray,
                    datasets: [
                        { label: '온도 (Temperature)', backgroundColor: 'transparent',
                        borderColor: 'green', data: tempAvgArray, datalabels: {labels: { value:

```

```

{color: 'green'} } } },

        { label: '수소이온농도 (pH)', backgroundColor: 'transparent',
          borderColor: 'blue', data: phAvgArray, datalabels: {labels: { value: {color:
'blue'} } } },

        { label: '용존산소량 (DO)', backgroundColor: 'transparent',
          borderColor: 'orange', data: doAvgArray, datalabels: {labels: { value: {color:
'orange'} } } },

        { label: '탁도 (Turbidity)', backgroundColor: 'transparent',
          borderColor: 'purple', data: turbiAvgArray, datalabels: {labels: { value:
{color: 'purple'} } } }

            ] },
options: {
  plugins: {
    datalabels: {
      align: function(context){
        if (context.dataIndex == 0) return '-45'
        else if (context.dataIndex == tempAvgArray.length - 1) return
'240'

        else return 'end'
      },
      anchor: 'start',
      backgroundColor: null,
      font: {
        size:12,
        weight: 600
      },
    },
  }
});
}

function showTimeButton() {
  var chart = new Chart(ctx, {
    type: 'line',
    data: { labels: hourArray,
    datasets: [
      { label: '온도 (Temperature)', backgroundColor: 'transparent',
        borderColor: 'green', data: timeTempAvgArray, datalabels: {labels: { value:
{color: 'green'} } } },

      { label: '수소이온농도 (pH)', backgroundColor: 'transparent',
        borderColor: 'blue', data: timePhAvgArray, datalabels: {labels: { value:
{color: 'blue'} } } },

      { label: '용존산소량 (DO)', backgroundColor: 'transparent',
        borderColor: 'orange', data: timeDoAvgArray, datalabels: {labels: { value:
{color: 'orange'} } } },

      { label: '탁도 (Turbidity)', backgroundColor: 'transparent',
        borderColor: 'purple', data: timeTurbiAvgArray, datalabels: {labels: { value:
{color: 'purple'} } } }

```

```

        ] },
        options: {
            plugins: {
                datalabels: {
                    align: function(context){
                        if (context.dataIndex == 0) return '-45'
                        else if (context.dataIndex == timeTempAvgArray.length - 1)
return '240'

                        else return 'end'
                    },
                    anchor: 'start',
                    backgroundColor: null,
                    font: {
                        size: 12,
                        weight: 600
                    },
                },
            }
        }
    });
}
showDayButton()
</script>
</div>
</div>
</br>
<!-- #####-->
<!-- 카카오택 API -->
<!-- #####-->
<div class="row">
    <div class="col-sm-6 col-lg-8 col-md-5">
        </br><h4> 설치 위치
</h4></br>
        <div id = "map" style="width:100%; height:410px;"></div>
        <script
                                                                    type="text/javascript"
src="//dapi.kakao.com/v2/maps/sdk.js?appkey=b2b63e3bdf61bac7e7ebcf827c4fcc6"></script>
        <script>
            var container = document.getElementById('map');
            var options = {
                center: new kakao.maps.LatLng(35.91111,128.81074),
                level: 3
            };
            var map = new kakao.maps.Map(container, options);
            var markerPosition = new kakao.maps.LatLng(35.91111, 128.81074); // Marker
            var marker = new kakao.maps.Marker({
                position: markerPosition
            });
            marker.setMap(map);

```



```

        var iwContent = '<div style="padding:10px; width:300px; height:80px;"><h6>대구가톨릭
대학교 강당</h6><p>경북 경산시 하양읍 하양로 13-13</p></div>',
        iwRemoveable = true;
        var infowindow = new kakao.maps.InfoWindow({
            content : iwContent,
            removable : iwRemoveable
        });
        kakao.maps.event.addListener(marker, 'click', function() {
            infowindow.open(map,
marker);
        }
    );

</script>
</div>
</br>
<!-- #####-->
<!--      OpenWeather API      -->
<!-- #####-->
<div class="col-sm-6 col-lg-4 col-md-7">
<br><h4>날씨
</h4></br>
<div class="row" style="background-color:#F4FFFF;">
    <div class="col-6">
        .svg" style="width:150px;
height:150px;"/>
    </div>
    <div class="col-6" style="margin-top:20px; text-align:right;">
        <h2><%= w_data_obj["temp"]%></h3>
        <h3><%= w_condition_obj["description"] %></h4>
        <h4 style="font-style:italic;"><%= w_data_obj["location"]%></h5>
    </div>
</div>
<div class="row" style="background-color:#EBF5FF; padding-top:10px;">
    <div class="col-6" style="text-align: center;">
        <span></img>최저온도
</span>
        <h5 style="padding-left:3rem;"><%= w_data_obj["temp_min"] %></h5>
    </div>
    <div class="col-6" style="text-align: center;">
        <span></img>최
고온도</span>
        <h5 style="padding-left:3rem;"><%= w_data_obj["temp_max"] %></h5>
    </div>

```

```

        </div>
        <div class="row" style="background-color:#EBF5FF; padding-top:10px;">
            <div class="col-6" style="text-align: center;">
                <span></img>풍속
</span>

                <h5 style="padding-left:2rem;"><%= w_data_obj["wind"] %> m/s</h5>
            </div>
            <div class="col-6" style="text-align: center;">
                <span></img>습도
</span>

                <h5 style="padding-left:2rem;"><%= w_data_obj["humidity"] %> %</h5>
            </div>
        </div>
    </div>
</div>
</br>
</br>
<!-- #####-->
<!-- Footer -->
<!-- #####-->
<div class="row">
    <div class="col-12 footer-design">
        <h6>Copyright by 한수앞선, 2021 All Rights Reserved</h6>
        <pre>|(38430) 경상북도 경산시 하양읍 하양로 13-13| Daegu Catholic Univ|</pre>
        <pre>|대표자명: 박천복| 번호: 010-5538-0978|</pre>
    </div>
</div>
</div>

<!-- #####-->
<!-- 페이지 새로고침 (DB 최신화 반영) -->
<!-- #####-->
<script>
    setTimeout(function(){
        location.reload();
    }, 300000); // (600000 = 10분)
</script>
</body>
</html>

```

④ 웹 서비스 실행 결과



컴퓨터소프트웨어학부 한수앞선

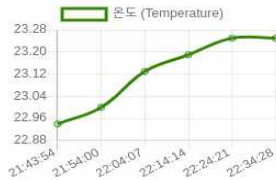
Water Monitoring System

마지막 측정 시간 : Tue Jun 01 2021 22:34:28



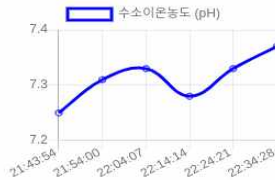
온도

23.25 °C



수소이온농도

7.37 pH



용존산소량(DO)

3.46 mg/L



탁도

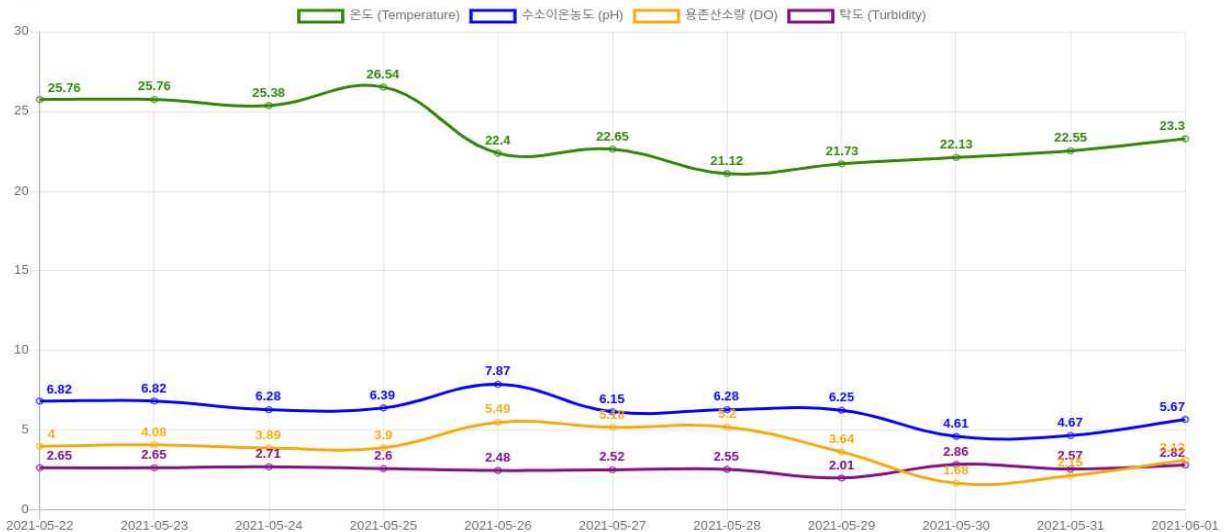
2.97

탁함(0) ~ 맑음(3.3)



누적 데이터

Day ☒ Hour ☐



설치 위치



날씨



23.77 °C

clear sky
Hayang



최저온도

21.67 °C



최고온도

23.77 °C



풍속

0.73 m/s



습도

57 %

Copyright by 한수앞선, 2021 All Rights Reserved

[(38430) 경상북도 경산시 하양읍 하양로 13-13] Daegu Catholic Univ

[대표자명: 박천복] 번호: 010-5538-0978

그림 78 웹 서비스 실행 결과

OpenWeather API에서 제공하는 날씨 이미지는 기본으로 제공하는 이미지가 마음에 들지 않아 교체한다. 무료 소스로 제공하는 날씨 이미지를 다운로드 후에 API에 호환되도록 이미지 파일명을 변경하여 적용한다. API에서는 날씨 이미지를 01d, 01n, 02d, 02n, ...형식으로 정의했다.

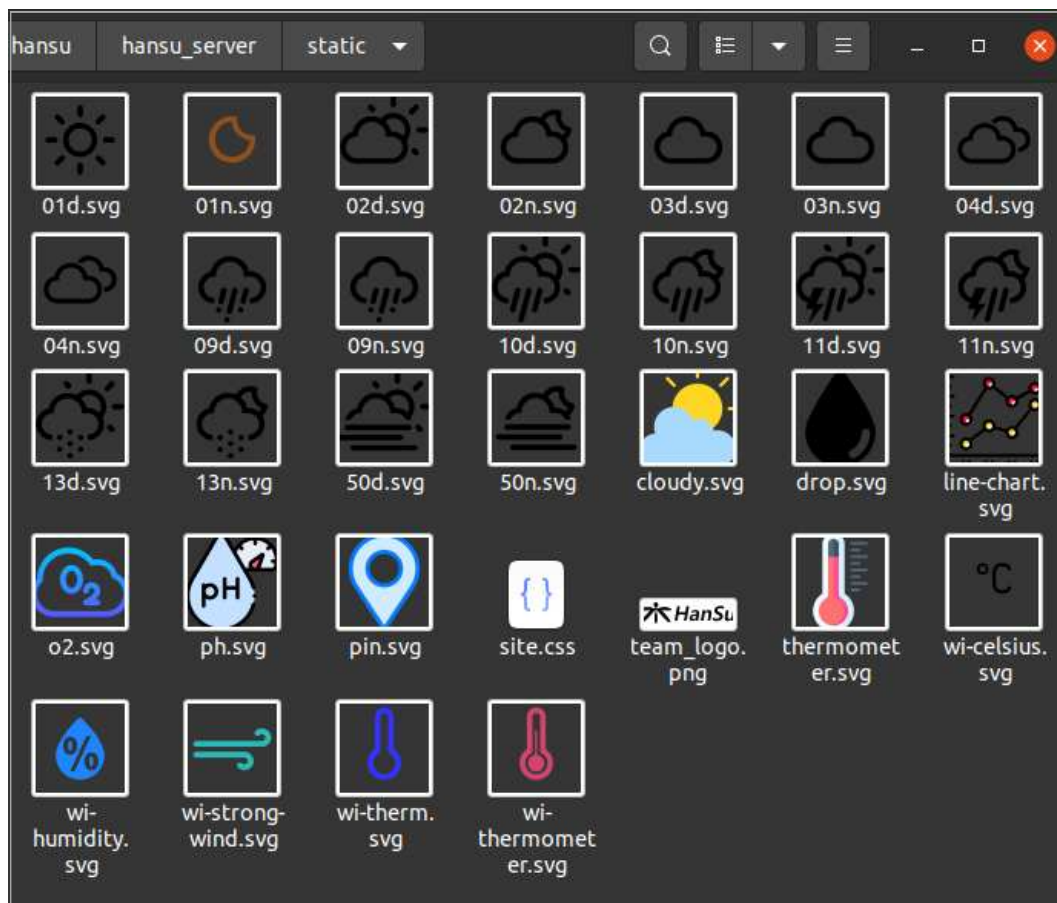


그림 79 ./static 디렉터리의 이미지 및 css 파일

2-5. 수질 측정 장비 케이스 제작

라즈베리파이와 센서 모듈의 보관과 이동이 불편한 문제가 있다. 회로와 모듈이 방치되어 있어서 물에 의한 고장의 문제도 있다. 따라서, 수질 측정 장비를 보관할 케이스 제작이 불가피하여 3D 프린팅을 통해서 케이스를 제작한다. 3D 모델링 프로그램으로 Fusion360을 사용했으며 3D 프린터로 큐비콘 장비를 사용한다. 3D 모델은 대구가톨릭대학교 중앙도서관의 DMZ(Digital Maker Zone)에서 제작한다.

최종적으로 만들어지는 케이스는 큐비콘의 최대 규격에서 벗어나기 때문에 분할하여 디자인했고, 총 8분할로 조립식 케이스를 제작한다.

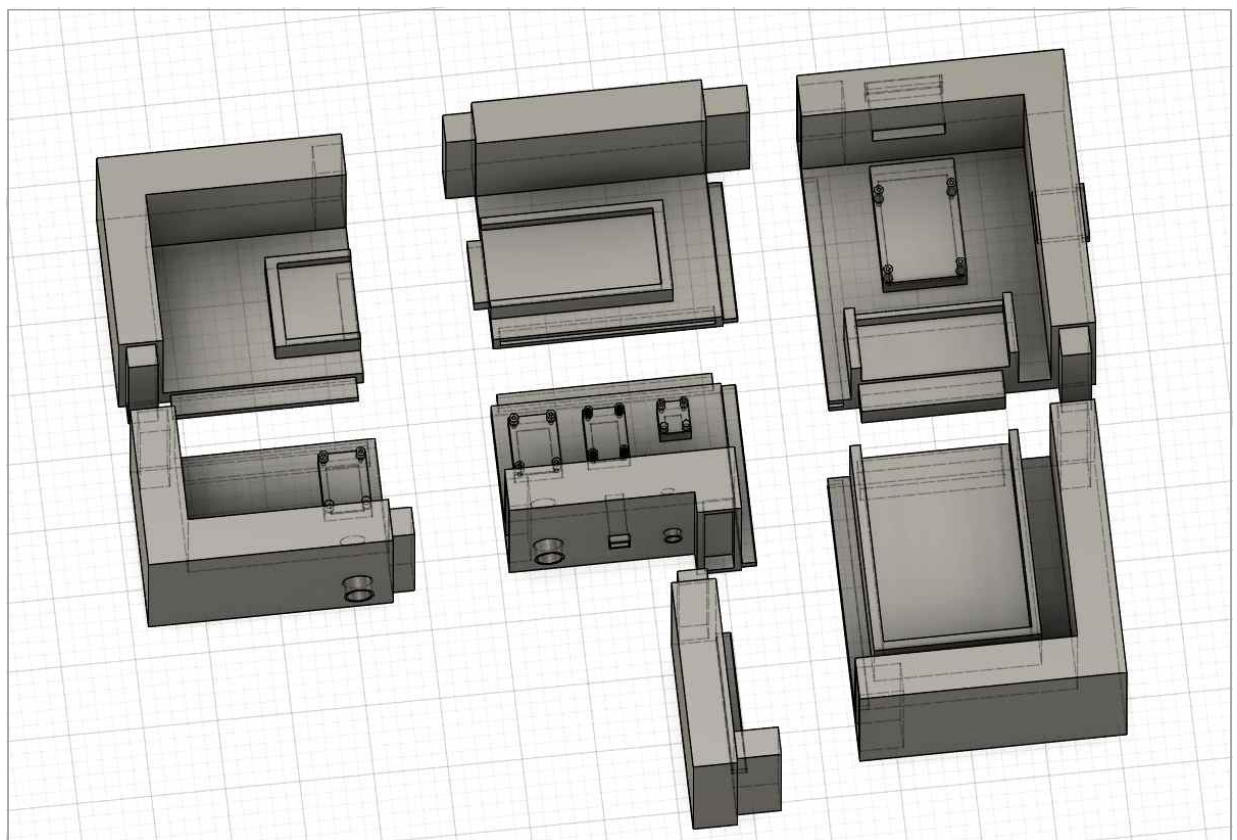


그림 80 수질 측정 장비 케이스 도면

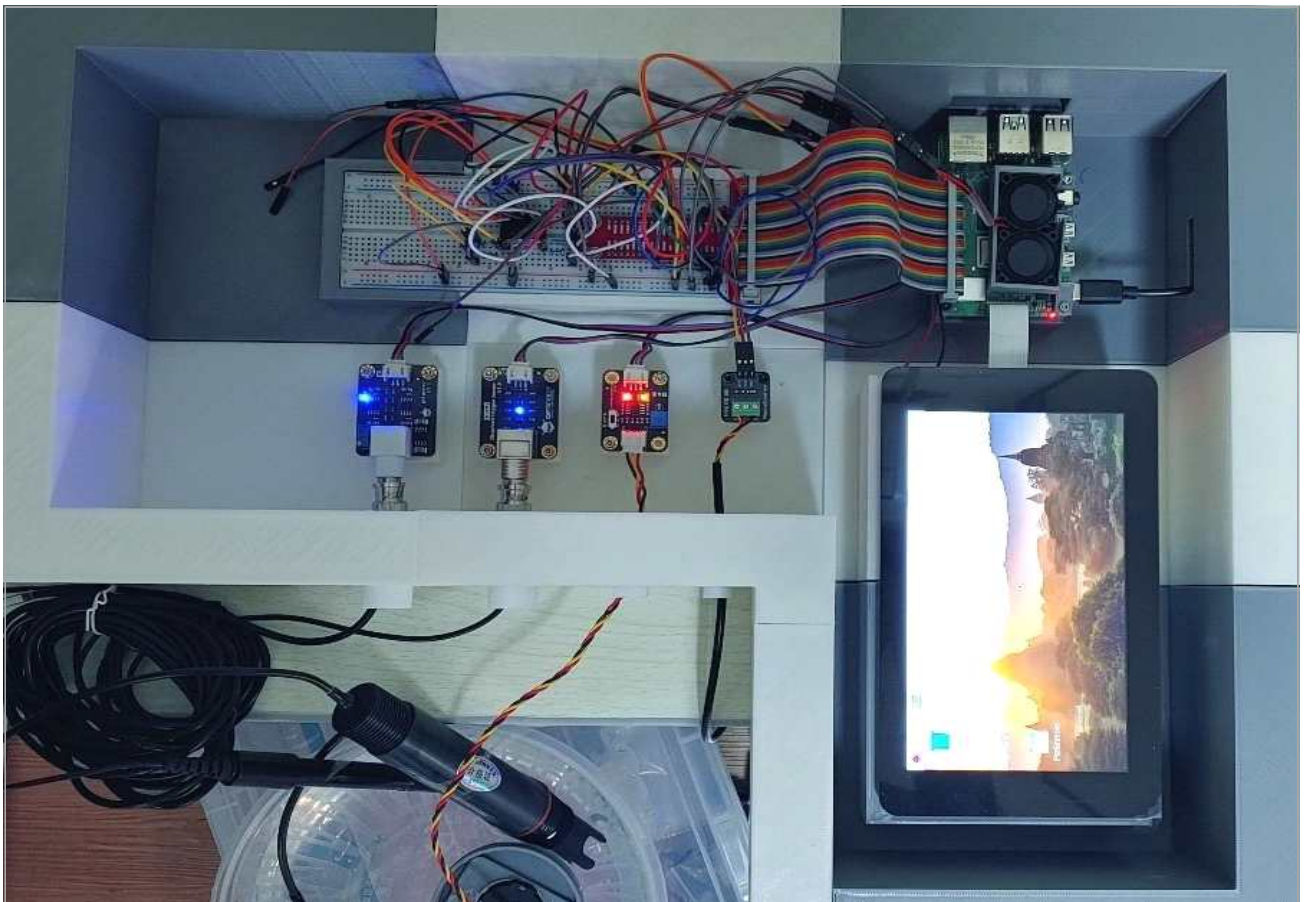


그림 81 케이스 제작 및 설치