|  |
| --- |
| **IoT 시스템 설계 [설계과제] 2.**  **동아리방 IOT 시스템** |

|  |  |
| --- | --- |
| **과목명(분반)** | IoT 시스템 설계(01) |
| **담당교수** | 조윤석 교수님 |
| **이름** | 정지우 |
| **학번** | 21800677 |
| **이름** | 김가륜 |
| **학번** | 21900059 |
| **이름** | 박석환 |
| **학번** | 22001020 |

**목차**

**1.서론**

1.1 설계 과제 내용

1.2 Requirements

1.3 Specification

1.4. 설계 과제 목적 및 의미

**2.   본론**

2.1. Function Flow Chart

2.1.1. project.ino 의 Function Flow Chart

2.1.2. Bus Information Function Flow Chart

2.1.3. 화재 경보 Function Flow Chart

2.1.4. 침입 경보 Function Flow Chart

2.2. UI 구성

2.3. 소스코드 및 알고리즘 설명

2.3.1. Arduino Source Code

2.3.2. Home Assistant Source Code

2.4. 최종 구현 및 데모 결과

2.4.1. HASS UI 및 Cam

2.4.2 Alert

**3.   결론**

3.1. 작업 중 발생한 문제점

3.2. 팀워크

3.3. 결과 및 고찰

**4. 참고문헌**

**1. 서론**

**1.1. 설계 과제 내용**

본 설계 과제의 목표는 NodeMCU, Raspberry Pi, HomeAssistant를 활용하여 실제로 활용 가능한 IoT 시스템을 구축하는 것이다. 우선 IoT란, 물리적인 장치들이 네트워크를 통해 연결되어 데이터를 주고받고 서로 상호작용하는 시스템을 말한다. 본 과제는 실생활에서 유용하게 사용 가능한 시스템을 구축하고자 한다.

NodeMCU는 다양한 센서와 LED를 포함하여 IoT 기기 역할을 한다. 센서를 통해 수집한 데이터를 JSON 형식의 메시지로 Raspberry Pi에 전송한다. Raspberry Pi는 이 데이터를 받아들여 처리하고 가공하는 역할을 수행한다. 본 과제에서는 Flask가 아닌 HomeAssistant가 중앙 제어 서버로 사용된다. HomeAssistant(약칭 HASS)는 IoT 기기들을 통합하고 제어하는 오픈소스 플랫폼으로, 다양한 제조사의 장치들을 하나의 통합된 시스템으로 운영할 수 있다.

**1.2. Requirements**

* NodeMCU(ESP8266)
* RaspberryPi 4
* ESP32-CAM-MB
* DHT22
* Adafruit\_SSD1306
* Home Assistant
* PIR Sensor(HC-SR501)
* 국토교통부\_(TAGO)\_버스도착정보 API
* Gas Sensor(MQ-7)

**1.3. Specification**

본 조의 주제는 동아리방 관리 시스템이다. 동아리 방의 각종 환경 데이터를 원격으로 측정하고 부가적인 정보를 제공하는 시스템을 구축하였다.

* dht\_22 센서를 이용하여 주기적으로 온도와 습도를 측정한다.
* MQ-7 센서로 LPG, 부탄가스, 프로판가스, 메탄가스, 알코올, 수소가스, 연기 등의 가연성 가스 농도를 측정한다. 가연성 가스의 농도가 특정 수치 이상이면 모바일 HASS로 알림을 보낸다.
* ESP32Cam으로 동아리 방 내부를 모바일 및 PC에서 확인할 수 있도록 한다. 문 근처에 인체 감지 센서를 설치하여 인원 파악을 가능케 한다. HASS에 등록된 사용자가 동아리 방 외부에 있으나, 내부에 인체 감지 센서로 사람이 감지되면 침입자로 간주하고 모바일 HASS로 알림이 전송된다.
* 공공 데이터 포털의 국토 교통부 버스 도착 정보 API를 활용하여 한동 대학교에서 탑승 가능한 302번 버스와 302(기본) 버스의 도착 정보를 표시한다. 또한, 버스 대기 모드를 만들어 이 상태에서 302번 버스 또는 302(기본) 버스의 한동 대학교 도착 예상 시간이 10분 이하일 경우 모바일 HASS로 알림이 전송된다.

**1.4. 설계 과제 목적 및 의미**

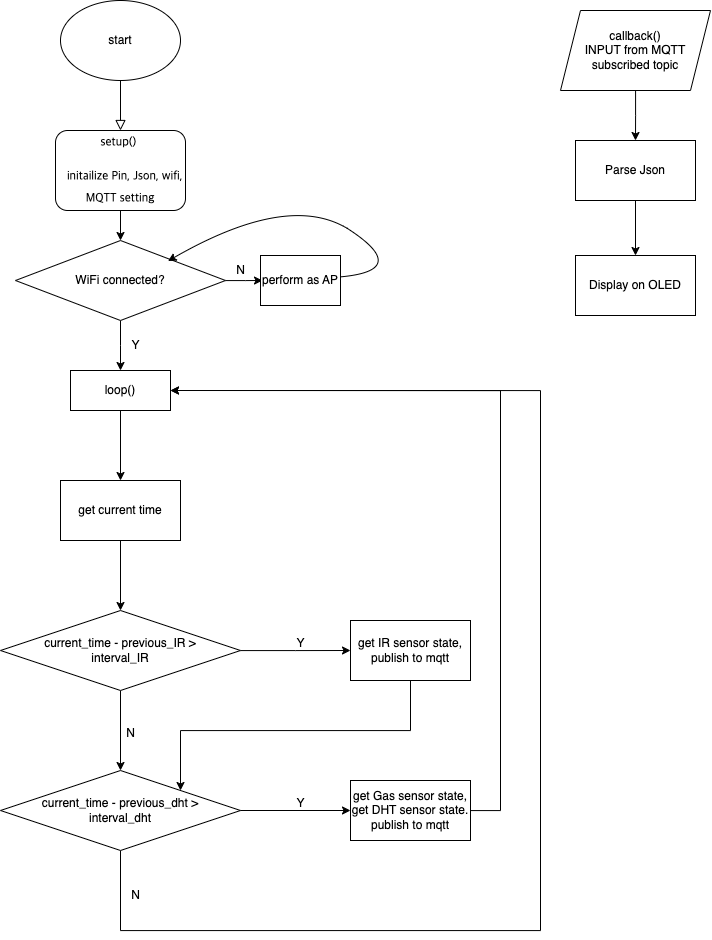
본 설계 과제의 목적은 원격으로 동아리방을 관리하고 다양한 환경 데이터를 측정하는 것이다. 기존에 본 수업에서 배웠던 내용을 활용하고, 추가적으로 각종 문서 및 매체를 찾아보며 실생활에서 불편함을 느낄만한 부분을 NodeMCU와 각종 센서, HASS를 활용하여 개선하는 것에 의미를 두었다.

특히 한동 대학교의 화재 경보 시스템은 오작동이 많기에 제대로 된 화재 알림 시스템의 필요성을 느꼈고, 교통편도 버스 2대와 택시로 불편하기에 이러한 점들을 개선하기 위한 시스템을 설계하였다. 또한, 정지우 학부생은 두나미스, 김가륜 학부생은 Ghost, 박석환 학부생은 슬기짜기에 속해있기에 각자의 동아리 방에서 느꼈던 동아리 방의 불편한 점들을 서로 공유하고, 이를 IOT 시스템을 활용하여 해결하는 것에 큰 의미를 두었다.

**2. 본론**

**2.1. Function Flow Chart**

**2.1.1. project.ino 의 Function Flow Chart**

****

**2.1.2. Bus Information Function Flow Chart**

**텍스트, 스크린샷, 폰트, 그래픽 디자인이(가) 표시된 사진

자동 생성된 설명**

**2.1.3. 화재 경보 Function Flow Chart**

**텍스트, 도표, 스케치, 영수증이(가) 표시된 사진

자동 생성된 설명**

**2.1.4. 침입 경보 Function Flow Chart**

**텍스트, 스크린샷, 폰트, 그래픽 디자인이(가) 표시된 사진

자동 생성된 설명**

**2.2. UI 구성**

UI는 Home Assistant를 사용하여 구성하였다. 대시보드 상에는 NodeMCU에서 MQTT 통신을 통하여 받아오는 온도, 습도, 가스 데이터 및 동작 감지 센서를 포함하는 카드, Bus Wait Mode 스위치를 포함하는 카드, 버스 노선 별 도착 예정 시간들을 포함하는 카드, 사용자의 위치를 출력하는 카드, 마지막으로 사용자의 위치를 지도 상에서 보여주는 카드가 있고, 우측의 목록에는 ESP-32 CAM을 iframe-panel을 활용해 추가하여 카메라의 상태를 확인할 수 있도록 하였다.

**2.3. 소스코드 및 알고리즘 설명**

**2.3.1. Arduino Source Code**

* Project.ino

|  |
| --- |
| #include <FS.h>  #include <PubSubClient.h>  #include "DHTesp.h"  #include <ArduinoJson.h>  #include <SPI.h>  #include <Wire.h>  #include <Adafruit\_GFX.h>  #include <Adafruit\_SSD1306.h>  #include <WiFiManager.h>  #include <DNSServer.h>  #include <ESP8266WebServer.h>  // Pin match  #define D0 16  #define D3 0  #define D4 2  #define D5 14  #define D6 12  #define LED\_PIN D0  #define GAS\_PIN A0  #define DHT\_PIN D3  #define IR\_O\_PIN D5  // #define IR\_I\_PIN D6  #define DHTTYPE DHT22  //OLED  #define SCREEN\_WIDTH 128 // OLED display width, in pixels  #define SCREEN\_HEIGHT 32 // OLED display height, in pixels  #define OLED\_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)  #define SCREEN\_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64, 0x3C for 128x32  Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET);  //sensors  unsigned long previousMillis\_DHT = 0;  const long interval\_DHT = 3000;  unsigned long previousMillis\_IR = 0;  const long interval\_IR = 1000;  float temperature = 0, humidity = 0;  int people\_count = 0;  int iSensorState;  int oSensorState;  int gas\_data = 0;  //json  DHTesp dht;  StaticJsonDocument<256> dhtJson;  JsonObject data = dhtJson.createNestedObject("data");  StaticJsonDocument<256> gasJson;  JsonObject data2 = gasJson.createNestedObject("data");  StaticJsonDocument<256> peopleJson;  JsonObject data3 = peopleJson.createNestedObject("data");  // Publish  const char \*pub\_dht = "iot/21900059/dht22";  const char \*pub\_gas = "iot/21900059/gas";  const char \*pub\_people = "iot/21900059/people";  // Subscribe  const char \*sub\_topic = "iot/21900059/bus\_info";  struct WifiSettings {  char apName[20] = "espAP";  char apPassword[20] = "esp12345";  };  // Here you can pre-set the settings for the MQTT connection. The settings can later be changed via Wifi Manager.  struct MqttSettings {  char clientId[20] = "ryun MCU";  char hostname[40] = "sweetdream.iptime.org";  char port[6] = "1883";  char user[20] = "iot";  char password[20] = "csee1414";  char wm\_client\_id\_identifier[15] = "client\_id";  char wm\_mqtt\_hostname\_identifier[14] = "mqtt\_hostname";  char wm\_mqtt\_port\_identifier[10] = "mqtt\_port";  char wm\_mqtt\_user\_identifier[10] = "mqtt\_user";  char wm\_mqtt\_password\_identifier[14] = "mqtt\_password";  };  // save config to file  bool shouldSaveConfig = false;  WiFiClient espClient;  WifiSettings wifiSettings;  PubSubClient client(espClient);  MqttSettings mqttSettings;  void readSettingsFromConfig() {  //read configuration from FS json  Serial.println("mounting FS...");  if (SPIFFS.begin()) {  Serial.println("mounted file system");  if (SPIFFS.exists("/config.json")) {  //file exists, reading and loading  Serial.println("reading config file");  File configFile = SPIFFS.open("/config.json", "r");  if (configFile) {  Serial.println("opened config file");  size\_t size = configFile.size();  // Allocate a buffer to store contents of the file.  std::unique\_ptr<char[]> buf(new char[size]);  configFile.readBytes(buf.get(), size);  // Use arduinojson.org/v6/assistant to compute the capacity.  StaticJsonDocument<1024> doc;  DeserializationError error = deserializeJson(doc, configFile);  if (error) {  Serial.println(F("Failed to read file, using default configuration"));  } else {  Serial.println("\nparsed json");  strcpy(mqttSettings.clientId, doc[mqttSettings.wm\_client\_id\_identifier]);  strcpy(mqttSettings.hostname, doc[mqttSettings.wm\_mqtt\_hostname\_identifier]);  strcpy(mqttSettings.port, doc[mqttSettings.wm\_mqtt\_port\_identifier]);  strcpy(mqttSettings.user, doc[mqttSettings.wm\_mqtt\_user\_identifier]);  strcpy(mqttSettings.password, doc[mqttSettings.wm\_mqtt\_password\_identifier]);  }  }  }  } else {  Serial.println("failed to mount FS");  }  }  //callback notifying us of the need to save config  void saveConfigCallback () {  Serial.println("Should save config");  shouldSaveConfig = true;  }  void initializeWifiManager() {  // The extra parameters to be configured (can be either global or just in the setup)  // After connecting, parameter.getValue() will get you the configured value  // id/name placeholder/prompt default length  WiFiManagerParameter custom\_client\_id("client\_id", "mqtt client id", mqttSettings.clientId, 40);  WiFiManagerParameter custom\_mqtt\_server("server", "mqtt server", mqttSettings.hostname, 40);  WiFiManagerParameter custom\_mqtt\_port("port", "mqtt port", mqttSettings.port, 6);  WiFiManagerParameter custom\_mqtt\_user("user", "mqtt user", mqttSettings.user, 20);  WiFiManagerParameter custom\_mqtt\_pass("pass", "mqtt pass", mqttSettings.password, 20);  //WiFiManager  //Local intialization. Once its business is done, there is no need to keep it around  WiFiManager wifiManager;  //set config save notify callback  wifiManager.setSaveConfigCallback(saveConfigCallback);  //add all your parameters here  wifiManager.addParameter(&custom\_client\_id);  wifiManager.addParameter(&custom\_mqtt\_server);  wifiManager.addParameter(&custom\_mqtt\_port);  wifiManager.addParameter(&custom\_mqtt\_user);  wifiManager.addParameter(&custom\_mqtt\_pass);  if (!wifiManager.autoConnect(wifiSettings.apName, wifiSettings.apPassword)) {  Serial.println("failed to connect and hit timeout");  delay(3000);  //reset and try again, or maybe put it to deep sleep  ESP.reset();  delay(5000);  }  //if you get here you have connected to the WiFi  Serial.println("connected...yeey :)");  //read updated parameters  strcpy(mqttSettings.clientId, custom\_client\_id.getValue());  strcpy(mqttSettings.hostname, custom\_mqtt\_server.getValue());  strcpy(mqttSettings.port, custom\_mqtt\_port.getValue());  strcpy(mqttSettings.user, custom\_mqtt\_user.getValue());  strcpy(mqttSettings.password, custom\_mqtt\_pass.getValue());  }  void saveConfig() {  Serial.println("saving config");  StaticJsonDocument<1024> doc;  doc[mqttSettings.wm\_client\_id\_identifier] = mqttSettings.clientId;  doc[mqttSettings.wm\_mqtt\_hostname\_identifier] = mqttSettings.hostname;  doc[mqttSettings.wm\_mqtt\_port\_identifier] = mqttSettings.port;  doc[mqttSettings.wm\_mqtt\_user\_identifier] = mqttSettings.user;  doc[mqttSettings.wm\_mqtt\_password\_identifier] = mqttSettings.password;  File configFile = SPIFFS.open("/config.json", "w");  if (!configFile) {  Serial.println("failed to open config file for writing");  }  configFile.close();  }  void initializeMqttClient() {  Serial.println("local ip");  Serial.println(WiFi.localIP());  client.setServer(mqttSettings.hostname, atoi(mqttSettings.port));  client.subscribe(sub\_topic);  }  bool tryConnectToMqttServer() {  if(strlen(mqttSettings.user) == 0) {  return client.connect(mqttSettings.clientId);  } else {  return client.connect(mqttSettings.clientId, mqttSettings.user, mqttSettings.password);  }  }  void reconnect() {  // Loop until we're reconnected  while (!client.connected()) {  Serial.print("Attempting MQTT connection...");  // Attempt to connect  // If you do not want to use a username and password, change next line to  if (tryConnectToMqttServer()) {  Serial.println("connected");  client.subscribe(sub\_topic);  } else {  Serial.print(mqttSettings.hostname);  Serial.print(" failed, rc=");  Serial.print(client.state());  Serial.println(" try again in 5 seconds");  // Wait 5 seconds before retrying  delay(5000);  }  }  }  void handleMqttState() {  if (!client.connected()) {  reconnect();  }  client.loop();  }  void callback(char\* topic, byte\* payload, unsigned int length) {  // if(strcmp(topic,"iot/21900059") != 0){  // return;  // }  Serial.println((char\*)payload);  const size\_t bufferSize = JSON\_OBJECT\_SIZE(2);  DynamicJsonDocument jsonBuffer(bufferSize);  // JSON 파싱  DeserializationError error = deserializeJson(jsonBuffer, payload);  if (!error) {  // 파싱된 데이터 가져오기  const char\* value302 = jsonBuffer["302"];  const char\* valueYangdeok = jsonBuffer["yangdeok"];  // 결과 출력  Serial.print("302: ");  Serial.println(value302);  Serial.print("yangdeok: ");  Serial.println(valueYangdeok);  display.clearDisplay();  display.setCursor(0, 0);  display.print(F("302: "));  display.print(value302);  // display.print(char(248)) ;  display.println(F(" min"));  display.print(F("yangdeok: "));  display.print(valueYangdeok);  // display.print(char(248)) ;  display.println(F(" min"));  display.display();  } else {  Serial.println("JSON parsing failed!");  }  }  void setup() {  Serial.begin(115200);  //Pin setup  pinMode(GAS\_PIN, INPUT);  pinMode(IR\_O\_PIN, INPUT);  pinMode(LED\_PIN, OUTPUT);  //dht setup  dht.setup(DHT\_PIN, DHTesp::DHTTYPE) ;  if (!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {  Serial.println(F("SSD1306 allocation failed"));  for (;;)  ; // Don't proceed, loop forever  }  display.cp437(true); // Use full 256 char 'Code Page 437' font  display.clearDisplay();  display.setTextSize(1); // Normal 1:1 pixel scale  display.setTextColor(SSD1306\_WHITE); // Draw white text  //indicate sensor type in json  dhtJson["sensor"] = "dht22" ;  gasJson["sensor"] = "gas" ;  peopleJson["sensor"] = "people" ;  //wifi and mqtt  readSettingsFromConfig();  initializeWifiManager();  if (shouldSaveConfig) {  saveConfig();  }  initializeMqttClient();  client.setCallback(callback);  }  void loop() {  handleMqttState();  //get current time  unsigned long currentMillis = millis();  //store json data as String  String dhtjsonData = "" ;  String peoplejsonData = "" ;  String gasjsonData = "" ;  char buf[100];  if (currentMillis - previousMillis\_IR >= interval\_IR) {  previousMillis\_IR = currentMillis;  oSensorState = digitalRead(IR\_O\_PIN);  sprintf(buf,"%d",oSensorState);  client.publish(pub\_people, buf) ;  Serial.println(buf);  }  //Get DHT information every interval\_DHT(milisec)  //And publish jsons, dht data and cds data  if (currentMillis - previousMillis\_DHT >= interval\_DHT) {  previousMillis\_DHT = currentMillis;  gas\_data = analogRead(GAS\_PIN);  humidity = dht.getHumidity();  temperature = dht.getTemperature();  int humidity\_temp = humidity \* 100 ;  int temperature\_temp = temperature \* 100 ;  data["humidity"] = humidity\_temp / 100 ;  data["temperature"] = temperature\_temp / 100 ;  serializeJsonPretty(dhtJson, dhtjsonData) ;  dhtjsonData.toCharArray(buf, 100);  client.publish(pub\_dht, (uint8\_t \*)buf, dhtjsonData.length(), true) ;  data2["concentration"] = gas\_data;  serializeJsonPretty(gasJson, gasjsonData) ;  gasjsonData.toCharArray(buf, 100);  sprintf(buf,"%d",gas\_data);  client.publish(pub\_gas, buf) ;  }  } |

ESP32\_cam을 제외한 나머지 센서를 사용하기 위하여 사용된 Nodemcu에 업로드 된 코드. 대부분의 코드는 WiFi에 자동 연결을 위해 사용한 <WiFiManager> library에 관련된 코드다.

코드가 길기 때문에 초반에 핀, 변수의 정의를 제외한 함수를 부분별로 나누어서 설명한다.

|  |
| --- |
| void setup() {  Serial.begin(115200);  //Pin setup  pinMode(GAS\_PIN, INPUT);  pinMode(IR\_O\_PIN, INPUT);  pinMode(LED\_PIN, OUTPUT);  //dht setup  dht.setup(DHT\_PIN, DHTesp::DHTTYPE) ;  if (!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {  Serial.println(F("SSD1306 allocation failed"));  for (;;)  ; // Don't proceed, loop forever  }  display.cp437(true); // Use full 256 char 'Code Page 437' font  display.clearDisplay();  display.setTextSize(1); // Normal 1:1 pixel scale  display.setTextColor(SSD1306\_WHITE); // Draw white text  //indicate sensor type in json  dhtJson["sensor"] = "dht22" ;  gasJson["sensor"] = "gas" ;  peopleJson["sensor"] = "people" ;  //wifi and mqtt  readSettingsFromConfig();  initializeWifiManager();  if (shouldSaveConfig) {  saveConfig();  }  initializeMqttClient();  client.setCallback(callback);  } |

이 setup 함수에서는 Pin설정, dht초기화, SSD1306 (OLED) 초기화, json초기화를 하고 wifi 연결을 시도하고 mqtt에 연결하며 Hass로부터 버스 시간정보를 받아올 토픽을 subscribe한다.

기존에 mqtt에 연결된 정보가 있는지 확인하는 함수 readSettingsFromConfig(); 를 호출한다. 기존에 연결된 정보가 있다면 그 정보를 저장해 두고 initializeWifiManager(); 를 호출한다. initializeWifiManager(); 에서 기존에 사용했던 정보로 연결을 성공하면 다음에 있는 조건문 if (shouldSaveConfig) 를 실행하지 않고 통과하고 만약 기존의 정보로 연결을 성공하지 못하였다면 이 부분을 넘어가게 된다. 이후 WiFi가 연결 되었다면 initializeMqttClient();

client.setCallback(callback); Mqtt를 초기화 및 topic을 subscribe 하고 setup()을 마친다.

이어서 setup()에서 호출했던 함수를 설명한다.

|  |
| --- |
| void readSettingsFromConfig() {  //read configuration from FS json  Serial.println("mounting FS...");  if (SPIFFS.begin()) {  Serial.println("mounted file system");  if (SPIFFS.exists("/config.json")) {  //file exists, reading and loading  Serial.println("reading config file");  File configFile = SPIFFS.open("/config.json", "r");  if (configFile) {  Serial.println("opened config file");  size\_t size = configFile.size();  // Allocate a buffer to store contents of the file.  std::unique\_ptr<char[]> buf(new char[size]);  configFile.readBytes(buf.get(), size);  // Use arduinojson.org/v6/assistant to compute the capacity.  StaticJsonDocument<1024> doc;  DeserializationError error = deserializeJson(doc, configFile);  if (error) {  Serial.println(F("Failed to read file, using default configuration"));  } else {  Serial.println("\nparsed json");  strcpy(mqttSettings.clientId, doc[mqttSettings.wm\_client\_id\_identifier]);  strcpy(mqttSettings.hostname, doc[mqttSettings.wm\_mqtt\_hostname\_identifier]);  strcpy(mqttSettings.port, doc[mqttSettings.wm\_mqtt\_port\_identifier]);  strcpy(mqttSettings.user, doc[mqttSettings.wm\_mqtt\_user\_identifier]);  strcpy(mqttSettings.password, doc[mqttSettings.wm\_mqtt\_password\_identifier]);  }  }  }  } else {  Serial.println("failed to mount FS");  }  } |

이 readSettingsFromConfig() 함수는 <FS> library를 사용한다. 이 라이브러리는 SPIFFS라는 간단한 파일시스템을 사용할 수 있게 해준다. MCU에 wifi config정보를 가진 파일을 가지고 와서 그 내용을 가져와서 이후 사용될 mqtt세팅 정보를 가져온다.

|  |
| --- |
| void initializeWifiManager() {  // The extra parameters to be configured (can be either global or just in the setup)  // After connecting, parameter.getValue() will get you the configured value  // id/name placeholder/prompt default length  WiFiManagerParameter custom\_client\_id("client\_id", "mqtt client id", mqttSettings.clientId, 40);  WiFiManagerParameter custom\_mqtt\_server("server", "mqtt server", mqttSettings.hostname, 40);  WiFiManagerParameter custom\_mqtt\_port("port", "mqtt port", mqttSettings.port, 6);  WiFiManagerParameter custom\_mqtt\_user("user", "mqtt user", mqttSettings.user, 20);  WiFiManagerParameter custom\_mqtt\_pass("pass", "mqtt pass", mqttSettings.password, 20);  //WiFiManager  //Local intialization. Once its business is done, there is no need to keep it around  WiFiManager wifiManager;  //set config save notify callback  wifiManager.setSaveConfigCallback(saveConfigCallback);  //add all your parameters here  wifiManager.addParameter(&custom\_client\_id);  wifiManager.addParameter(&custom\_mqtt\_server);  wifiManager.addParameter(&custom\_mqtt\_port);  wifiManager.addParameter(&custom\_mqtt\_user);  wifiManager.addParameter(&custom\_mqtt\_pass);  if (!wifiManager.autoConnect(wifiSettings.apName, wifiSettings.apPassword)) {  Serial.println("failed to connect and hit timeout");  delay(3000);  //reset and try again, or maybe put it to deep sleep  ESP.reset();  delay(5000);  }  //if you get here you have connected to the WiFi  Serial.println("connected...yeey :)");  //read updated parameters  strcpy(mqttSettings.clientId, custom\_client\_id.getValue());  strcpy(mqttSettings.hostname, custom\_mqtt\_server.getValue());  strcpy(mqttSettings.port, custom\_mqtt\_port.getValue());  strcpy(mqttSettings.user, custom\_mqtt\_user.getValue());  strcpy(mqttSettings.password, custom\_mqtt\_pass.getValue());  } |

이 initializeWifiManager() 함수는 이전에 연결했던 wifi가 있다면 그곳에 연결하고 그렇지 않으면 NodeMCU를 AP가 되도록 만든다. AP가 되면 사용자는 NodeMCU에 접속하여 wifi를 직접 연결할 수 있고, 원한다면 mqtt연결 정보도 수정할 수 있다.

|  |
| --- |
| void initializeMqttClient() {  Serial.println("local ip");  Serial.println(WiFi.localIP());  client.setServer(mqttSettings.hostname, atoi(mqttSettings.port));  client.subscribe(sub\_topic);  } |

여기서는 mqtt에 연결을 정보와 subscribe topic을 객체에 저장한다.

|  |
| --- |
| void loop() {  handleMqttState();  //get current time  unsigned long currentMillis = millis();  //store json data as String  String dhtjsonData = "" ;  String peoplejsonData = "" ;  String gasjsonData = "" ;  char buf[100];  if (currentMillis - previousMillis\_IR >= interval\_IR) {  previousMillis\_IR = currentMillis;  oSensorState = digitalRead(IR\_O\_PIN);  sprintf(buf,"%d",oSensorState);  client.publish(pub\_people, buf) ;  Serial.println(buf);  }  //Get DHT information every interval\_DHT(milisec)  //And publish jsons, dht data and cds data  if (currentMillis - previousMillis\_DHT >= interval\_DHT) {  previousMillis\_DHT = currentMillis;  gas\_data = analogRead(GAS\_PIN);  humidity = dht.getHumidity();  temperature = dht.getTemperature();  int humidity\_temp = humidity \* 100 ;  int temperature\_temp = temperature \* 100 ;  data["humidity"] = humidity\_temp / 100 ;  data["temperature"] = temperature\_temp / 100 ;  serializeJsonPretty(dhtJson, dhtjsonData) ;  dhtjsonData.toCharArray(buf, 100);  client.publish(pub\_dht, (uint8\_t \*)buf, dhtjsonData.length(), true) ;  data2["concentration"] = gas\_data;  serializeJsonPretty(gasJson, gasjsonData) ;  gasjsonData.toCharArray(buf, 100);  sprintf(buf,"%d",gas\_data);  client.publish(pub\_gas, buf) ;  }  } |

모든 초기설정이 끝났다면 loop()에 들어오게된다.

처음handleMqttState(); 는 mqtt연결이 계속 정상적으로 되어있는지 확인하는 함수다. 만약 mqtt연결이 끊어지면 재연결을 시도한다.  
이후, non-blocking으로 코드가 동작할 수 있도록 현재 시간을 가져오고 현재시간에 기반하여 interval\_IR, interval\_DHT 만큼 시간이 흘렀는지 확인하고 IR 혹은 DHT, Gas센서에서 센싱한다. 이후 센싱된 데이터는 연결된 mqtt서버에 publish 한다.

|  |
| --- |
| void callback(char\* topic, byte\* payload, unsigned int length) {  // if(strcmp(topic,"iot/21900059") != 0){  // return;  // }  Serial.println((char\*)payload);  const size\_t bufferSize = JSON\_OBJECT\_SIZE(2);  DynamicJsonDocument jsonBuffer(bufferSize);  // JSON 파싱  DeserializationError error = deserializeJson(jsonBuffer, payload);  if (!error) {  // 파싱된 데이터 가져오기  const char\* value302 = jsonBuffer["302"];  const char\* valueYangdeok = jsonBuffer["yangdeok"];  // 결과 출력  Serial.print("302: ");  Serial.println(value302);  Serial.print("yangdeok: ");  Serial.println(valueYangdeok);  display.clearDisplay();  display.setCursor(0, 0);  display.print(F("302: "));  display.print(value302);  // display.print(char(248)) ;  display.println(F(" min"));  display.print(F("yangdeok: "));  display.print(valueYangdeok);  // display.print(char(248)) ;  display.println(F(" min"));  display.display();  } else {  Serial.println("JSON parsing failed!");  }  } |

이 아두이노 코드에서 마지막으로, mqtt에서 데이터를 받았을 때 처리하는 callback 함수다.

mqtt에 subscribe한 topic으로부터 데이터를 받으면 이 함수가 실행된다. 이 함수에서는 302, yangdeok버스의 정보를 json형태로 전달받아서 이 데이터를 파싱하고 OLED에 출력한다.

* CameraWebServer.ino

|  |
| --- |
| #include "esp\_camera.h"  #include <WiFi.h>  //  // WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality  // Ensure ESP32 Wrover Module or other board with PSRAM is selected  // Partial images will be transmitted if image exceeds buffer size  //  // You must select partition scheme from the board menu that has at least 3MB APP space.  // Face Recognition is DISABLED for ESP32 and ESP32-S2, because it takes up from 15  // seconds to process single frame. Face Detection is ENABLED if PSRAM is enabled as well  // ===================  // Select camera model  // ===================  //#define CAMERA\_MODEL\_WROVER\_KIT // Has PSRAM  //#define CAMERA\_MODEL\_ESP32S3\_EYE // Has PSRAM  //#define CAMERA\_MODEL\_M5STACK\_PSRAM // Has PSRAM  //#define CAMERA\_MODEL\_M5STACK\_V2\_PSRAM // M5Camera version B Has PSRAM  //#define CAMERA\_MODEL\_M5STACK\_WIDE // Has PSRAM  //define CAMERA\_MODEL\_M5STACK\_ESP32CAM // No PSRAM  //#define CAMERA\_MODEL\_M5STACK\_UNITCAM // No PSRAM  #define CAMERA\_MODEL\_AI\_THINKER // Has PSRAM  //#define CAMERA\_MODEL\_TTGO\_T\_JOURNAL // No PSRAM  //#define CAMERA\_MODEL\_XIAO\_ESP32S3 // Has PSRAM  // \*\* Espressif Internal Boards \*\*  //#define CAMERA\_MODEL\_ESP32\_CAM\_BOARD  //#define CAMERA\_MODEL\_ESP32S2\_CAM\_BOARD  //#define CAMERA\_MODEL\_ESP32S3\_CAM\_LCD  //#define CAMERA\_MODEL\_DFRobot\_FireBeetle2\_ESP32S3 // Has PSRAM  //#define CAMERA\_MODEL\_DFRobot\_Romeo\_ESP32S3 // Has PSRAM  #include "camera\_pins.h"  // ===========================  // Enter your WiFi credentials  // ===========================  const char\* ssid = "\*\*\*\*\*\*\*\*\*\*\*\*";  const char\* password = "\*\*\*\*\*\*\*\*\*\*\*\*\*";  void startCameraServer();  void setupLedFlash(int pin);  void setup() {  Serial.begin(115200);  Serial.setDebugOutput(true);  Serial.println();  camera\_config\_t config;  config.ledc\_channel = LEDC\_CHANNEL\_0;  config.ledc\_timer = LEDC\_TIMER\_0;  config.pin\_d0 = Y2\_GPIO\_NUM;  config.pin\_d1 = Y3\_GPIO\_NUM;  config.pin\_d2 = Y4\_GPIO\_NUM;  config.pin\_d3 = Y5\_GPIO\_NUM;  config.pin\_d4 = Y6\_GPIO\_NUM;  config.pin\_d5 = Y7\_GPIO\_NUM;  config.pin\_d6 = Y8\_GPIO\_NUM;  config.pin\_d7 = Y9\_GPIO\_NUM;  config.pin\_xclk = XCLK\_GPIO\_NUM;  config.pin\_pclk = PCLK\_GPIO\_NUM;  config.pin\_vsync = VSYNC\_GPIO\_NUM;  config.pin\_href = HREF\_GPIO\_NUM;  config.pin\_sccb\_sda = SIOD\_GPIO\_NUM;  config.pin\_sccb\_scl = SIOC\_GPIO\_NUM;  config.pin\_pwdn = PWDN\_GPIO\_NUM;  config.pin\_reset = RESET\_GPIO\_NUM;  config.xclk\_freq\_hz = 20000000;  config.frame\_size = FRAMESIZE\_UXGA;  config.pixel\_format = PIXFORMAT\_JPEG; // for streaming  //config.pixel\_format = PIXFORMAT\_RGB565; // for face detection/recognition  config.grab\_mode = CAMERA\_GRAB\_WHEN\_EMPTY;  config.fb\_location = CAMERA\_FB\_IN\_PSRAM;  config.jpeg\_quality = 12;  config.fb\_count = 1;  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality  // for larger pre-allocated frame buffer.  if(config.pixel\_format == PIXFORMAT\_JPEG){  if(psramFound()){  config.jpeg\_quality = 10;  config.fb\_count = 2;  config.grab\_mode = CAMERA\_GRAB\_LATEST;  } else {  // Limit the frame size when PSRAM is not available  config.frame\_size = FRAMESIZE\_SVGA;  config.fb\_location = CAMERA\_FB\_IN\_DRAM;  }  } else {  // Best option for face detection/recognition  config.frame\_size = FRAMESIZE\_240X240;  #if CONFIG\_IDF\_TARGET\_ESP32S3  config.fb\_count = 2;  #endif  }  #if defined(CAMERA\_MODEL\_ESP\_EYE)  pinMode(13, INPUT\_PULLUP);  pinMode(14, INPUT\_PULLUP);  #endif  // camera init  esp\_err\_t err = esp\_camera\_init(&config);  if (err != ESP\_OK) {  Serial.printf("Camera init failed with error 0x%x", err);  return;  }  sensor\_t \* s = esp\_camera\_sensor\_get();  // initial sensors are flipped vertically and colors are a bit saturated  if (s->id.PID == OV3660\_PID) {  s->set\_vflip(s, 1); // flip it back  s->set\_brightness(s, 1); // up the brightness just a bit  s->set\_saturation(s, -2); // lower the saturation  }  // drop down frame size for higher initial frame rate  if(config.pixel\_format == PIXFORMAT\_JPEG){  s->set\_framesize(s, FRAMESIZE\_QVGA);  }  #if defined(CAMERA\_MODEL\_M5STACK\_WIDE) || defined(CAMERA\_MODEL\_M5STACK\_ESP32CAM)  s->set\_vflip(s, 1);  s->set\_hmirror(s, 1);  #endif  #if defined(CAMERA\_MODEL\_ESP32S3\_EYE)  s->set\_vflip(s, 1);  #endif  // Setup LED FLash if LED pin is defined in camera\_pins.h  #if defined(LED\_GPIO\_NUM)  setupLedFlash(LED\_GPIO\_NUM);  #endif  WiFi.begin(ssid, password);  WiFi.setSleep(false);  while (WiFi.status() != WL\_CONNECTED) {  delay(500);  Serial.print(".");  }  Serial.println("");  Serial.println("WiFi connected");  startCameraServer();  Serial.print("Camera Ready! Use 'http://");  Serial.print(WiFi.localIP());  Serial.println("' to connect");  }  void loop() {  // Do nothing. Everything is done in another task by the web server  delay(10000);  } |

기존 ESP32-cam의 예제 코드 중 하나인 ESP32 - Camera - CameraWebServer 코드를 활용하였다. 해당 코드에 웹서버에서 사용하는 WiFi의 SSiD 및 Password를 입력하여 사용하였다.

**2.3.2. Home Assistant Source Code**

* bus\_info.py

|  |
| --- |
| import requests  import json  url = 'https://apis.data.go.kr/1613000/ArvlInfoInqireService/getSttnAcctoArvlPrearngeInfoList?serviceKey=7fbmhI1JTcfdHRUVxPC1Q5dMRWaGT1iO9DR2RfJtxsghQ7Xc4tqtwAi3rBu5vIZmStXhsobOsYAC7m8qUCS3GA%3D%3D&pageNo=1&numOfRows=100&\_type=json&cityCode=37010&nodeId=PHB351001140'  response = requests.get(url)  data = response.json()  bus\_info = {}  for item in data['response']['body']['items']['item']:  bus\_info[item['routeno']] = round(item['arrtime']/60)  print(json.dumps(bus\_info)) |

공공 데이터 포털에서 제공하는 국토교통부\_(TAGO)\_버스도착정보 API를 사용하여 코드를 구성하였다. 서비스 인증 키, 페이지 번호, 행 수, 데이터 형식, 도시 코드, 정류장 ID 등의 파라미터를 포함하는 URL을 입력하였고, requests.get() 함수를 사용하여 API에 HTTP GET 요청을 보내 받아온 응답을 JSON 형식으로 파싱한다. 이후 빈 Dictionary인 bus\_info를 생성하고, 받아온 JSON 데이터 중 버스 노선 번호(routeno)와 도착 예정 시간(arrtime)을 추출하고, 도착 예정시간을 분 단위로 변환하여 bus\_info Dictionary에 저장한다. 이후 이 Dictionary를 JSON 형식으로 출력하는 기능을 가진다. 해당 코드를 활용 시, 특정 도시의 특정 버스 정류장에 도착하는 모든 버스의 노선 및 도착 예정 시간을 JSON 형식으로 받아올 수 있다.

* configuration.yaml

|  |
| --- |
| # Loads default set of integrations. Do not remove.  default\_config:  homeassistant:  name: IoTClass-ParkSH  latitude: 36.103295  longitude: 129.387009  elevation: 30  customize:  sensor.21900059\_temperature:  icon: mdi:oil-temperature  sensor.21900059\_humidity:  icon: mdi:water-percent  sensor.21900059\_light\_intensity:  icon: mdi:lightbulb  "sensor.yangdeog":  icon: mdi:bus-clock  "sensor.302\_gibon":  icon: mdi:bus-clock  lovelace:  # mode: yaml  mode: storage  # Load frontend themes from the themes folder  frontend:  themes: !include\_dir\_merge\_named themes  # Text to speech  tts:  - platform: google\_translate  automation: !include automations.yaml  script: !include scripts.yaml  scene: !include scenes.yaml  mqtt:  sensor:  - state\_topic: "iot/21900059/dht22"  name: 21900059\_temperature  value\_template: "{{ value\_json.data.temperature }}"  - state\_topic: "iot/21900059/dht22"  name: 21900059\_humidity  value\_template: "{{ value\_json.data.humidity }}"  - state\_topic: "iot/21900059/gas"  name: 21900059\_gas  - state\_topic: "iot/21900059/people"  name: 21900059\_people  switch:  - name: "Bus Wait Mode"  state\_topic: "iot/22001020"  command\_topic: "iot/22001020"  qos: 0  payload\_on: "awayon"  payload\_off: "awayoff"  retain: false  sensor:  - platform: command\_line  name: Bus information  command: python3 ~/.homeassistant/bus\_info.py  json\_attributes:  - '302(기본)'  - '양덕'  scan\_interval: 60  group:  default\_view:  MyData:  name: "NodeMCU"  entities:  - sensor.21900059\_temperature  - sensor.21900059humidity  - sensor.21900059\_light\_intensity  - sensor.21900059\_gas  - sensor.21900059\_people  template:  - sensor:  - name: "302(기본)"  state: "{{ state\_attr('sensor.bus\_information', '302(기본)') }}"  - name: "양덕"  state: "{{ state\_attr('sensor.bus\_information', '양덕') }}"  panel\_iframe:  esp32\_cam:  title: ESP32-CAM  icon: mdi:camera  url: http://192.168.137.150/  device\_tracker:  - platform: gpslogger  webhook\_ID: fdbafae9ee586d00adec0664fc8ebed419b9bf80312218bd278b86fdcd3bfa0c |

configuration.yaml파일은 Home Assistant에서 시스템 전반의 설정을 관리하는 데 사용되는 파일이다. 이 파일은 Home Assistant의 구성 요소, 센서, 통합, 플랫폼, 그룹, 자동화 및 사용자 정의 컴포넌트 등의 설정을 포함한다. Ui 편집은 yaml 파일로 하는 것이 아닌 HASS에서 직접 편집 가능하게 하였다.

mqtt의 sensor 부분에서는 NodeMCU의 온도, 습도, 가연성 가스의 농도, 인원을 NodeMCU의 payload에 따라 json 형식의 메시지로 각각의 데이터를 받아온다. 또한 버스의 도착 정보를 sensor로 설정하여, bus\_info.py에서 받아온 버스들의 도착 대기 시간을 HASS에서 표시 가능하게 하였다. 센서 형식으로 버스들의 도착 대기 시간을 표시하기에, automation을 활용하여 각종 이벤트를 발생시킬 수 있다. 버스 도착 대기 시간의 갱신 시간은 1분으로 설정하였다.

switch 부분에서는 Bus Wait Mode라는 스위치를 만들어서 버스 대기 알림을 받을 수 있는 트리거 역할을 하도록 하였다.

카메라의 경우 HASS에서 card를 사용하여 추가하는 것이 아닌, ESP-32 Cam의 url을 받아와서 그 페이지를 panel\_iframe을 통해 HASS에 별도로 띄우는 형식으로 하였다.

또한, 사용자의 위치 정보를 받아오기 위하여 device\_tracker 중 gpslogger라는 Intergration을 활용하였다. 해당 Intergration의 개발자 깃허브에서 안드로이드용 애플리케이션을 다운받아 활용하였다. Intergration을 추가할 때 지정되는 Webhook\_ID를 사용하여 모바일 애플리케이션에서 사용자 url로 데이터를 전송하도록 지정하였고, 이를 통해 현재 사용자의 gps 정보를 받아오도록 설정하였다.

automations.yaml

* + alert\_bus\_time

|  |
| --- |
| - id: alert\_bus\_time  alias: Alert bus time  trigger:  - platform: state  entity\_id:  - sensor.yangdeog  condition:  - condition: and  conditions:  - condition: state  entity\_id: switch.bus\_wait\_mode  state: 'on'  - condition: numeric\_state  entity\_id: sensor.yangdeog  below: 11  action:  - service: notify.mobile\_app\_iphone  data:  message: 양덕 버스가 10분 이내로 도착합니다  initial\_state: true |

* + alert\_bus\_time2

|  |
| --- |
| - id: alert\_bus\_time2  alias: Alert bus time2  initial\_state: true  trigger:  platform: state  entity\_id: sensor.302\_gibon  condition:  condition: and  conditions:  - condition: state  entity\_id: switch.bus\_wait\_mode  state: 'on'  - condition: numeric\_state  entity\_id: sensor.302\_gibon  below: 11  action:  service: notify.mobile\_app\_iphone  data:  message: 302(기본) 버스가 10분 이내로 도착합니다  initial\_state: true |

이 automation들은 302번 버스와 302(기본) 버스의 도착 알림을 제공하는 부분이다. configuration.yaml 파일에서 버스의 대기 시간을 센서 형식으로 받아와서 마치 HW4의 조도 값의 변화에 따른 전등 제어와 비슷한 시스템을 구축하였다. Bus Wait Mode라는 스위치가 On 상태이고, 센서 값으로 받아온 302번 혹은 302(기본) 버스의 도착 예상 시간이 11분 미만일 경우, HASS를 설치한 사용자의 아이폰으로 알림이 도착한다. 이 알림은 버스 도착 시간이 갱신될 때마다 조건을 만족하는 경우 알림이 도착한다.

* + alert\_thief

|  |
| --- |
| - id: alert\_thief  alias: Alert thief  trigger:  - platform: state  entity\_id: sensor.21900059\_people  condition:  - condition: and  conditions:  - condition: state  entity\_id: device\_tracker.sm\_n981n  state: 'not\_home'  - condition: numeric\_state  entity\_id: sensor.21900059\_people  above: 0  action:  - service: notify.mobile\_app\_iphone  data:  message: 외부인이 침입했습니다  initial\_state: true |

alert\_thief는 외부인이 집에 침입하였을 때 알림을 보내는 automation이다. 트리거는 동작 감지 센서의 상태가 변하는 것이고, 조건은 사용자가 집에 없는 상태인 것, 그리고 동작 감지 센서가 움직임을 감지하는 것이며, 액션은 외부인이 침임했다는 알림을 사용중인 아이폰에 보내는 것이다.

* + alert\_fire

|  |
| --- |
| - id: alert\_fire  alias: Fire Alert  initial\_state: true  trigger:  - platform: numeric\_state  entity\_id: sensor.21900059\_gas  above: 210  action:  - service: notify.mobile\_app\_iphone # 알림 서비스 선택  data:  message: "Fire!!!" # 원하는 알림 메시지 |

alert\_fire는 가연성 가스 감지 센서의 값이 특정 수치를 초과할 때 화재 경보를 보내는 automation이다. 트리거는 가연성 가스 감지 센서의 값이 210을 초과하는 것이고, 이 때의 액션은 사용중인 스마트폰에 화재 경보 알림을 보내는 것이다.

* + json\_bus\_time

|  |
| --- |
| - id: json\_bus\_time  alias: "Send bus data to MQTT"  trigger:  platform: state  entity\_id:  - sensor.bus\_information  action:  - service: mqtt.publish  data:  topic: "iot/21900059/bus\_info"  payload: >  {  "302": "{{ state\_attr('sensor.bus\_information', '302(기본)') }}",  "yangdeok": "{{ state\_attr('sensor.bus\_information', '양덕') }}"  } |

위 automation은 Home Assistant에 기록된 버스 도착 시간을 MQTT 통신을 이용하여 전송하는 기능을 가진다. 트리거는 bus\_information 센서의 state가 변경되는 것이고, 액션은 MQTT를 통해 버스 도착 시간을 전송하는 것이다.

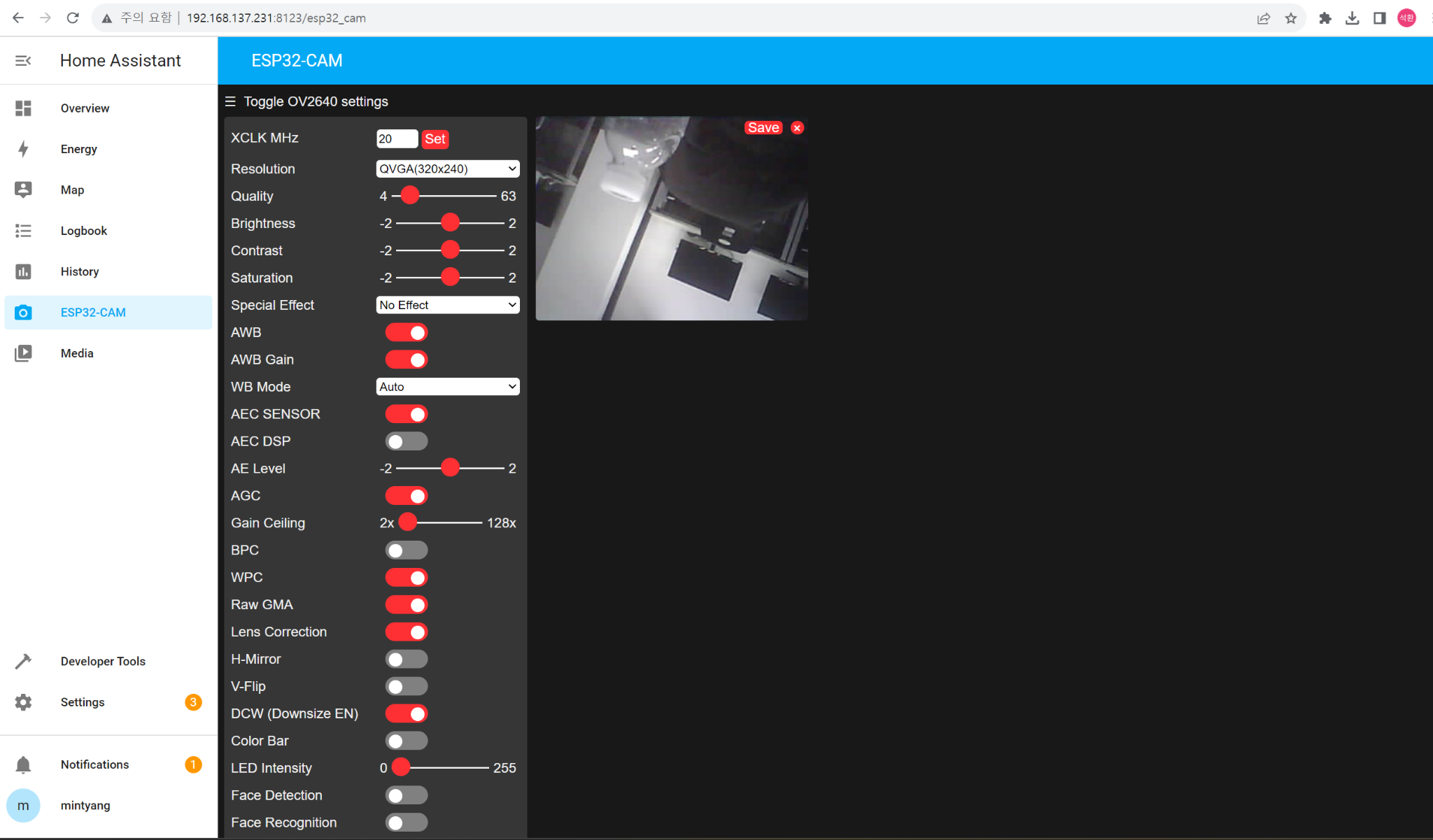
* ui-lovelace.yaml

|  |
| --- |
| title: 동아리 방 관리 시스템  views:  - path: default\_view  title: Home  cards:  - type: entities  entities:  - entity: sensor.21900059\_temperature  - entity: sensor.21900059\_humidity  - entity: sensor.21900059\_gas  - entity: sensor.21900059\_people  title: NodeMCU  show\_header\_toggle: true  - type: entities  entities:  - switch.bus\_wait\_mode  show\_header\_toggle: false  - type: entities  entities:  - sensor.302\_gibon  - sensor.yangdeog  - type: entities  entities:  - device\_tracker.sm\_n981n  - type: map  entities:  - device\_tracker.sm\_n981n |

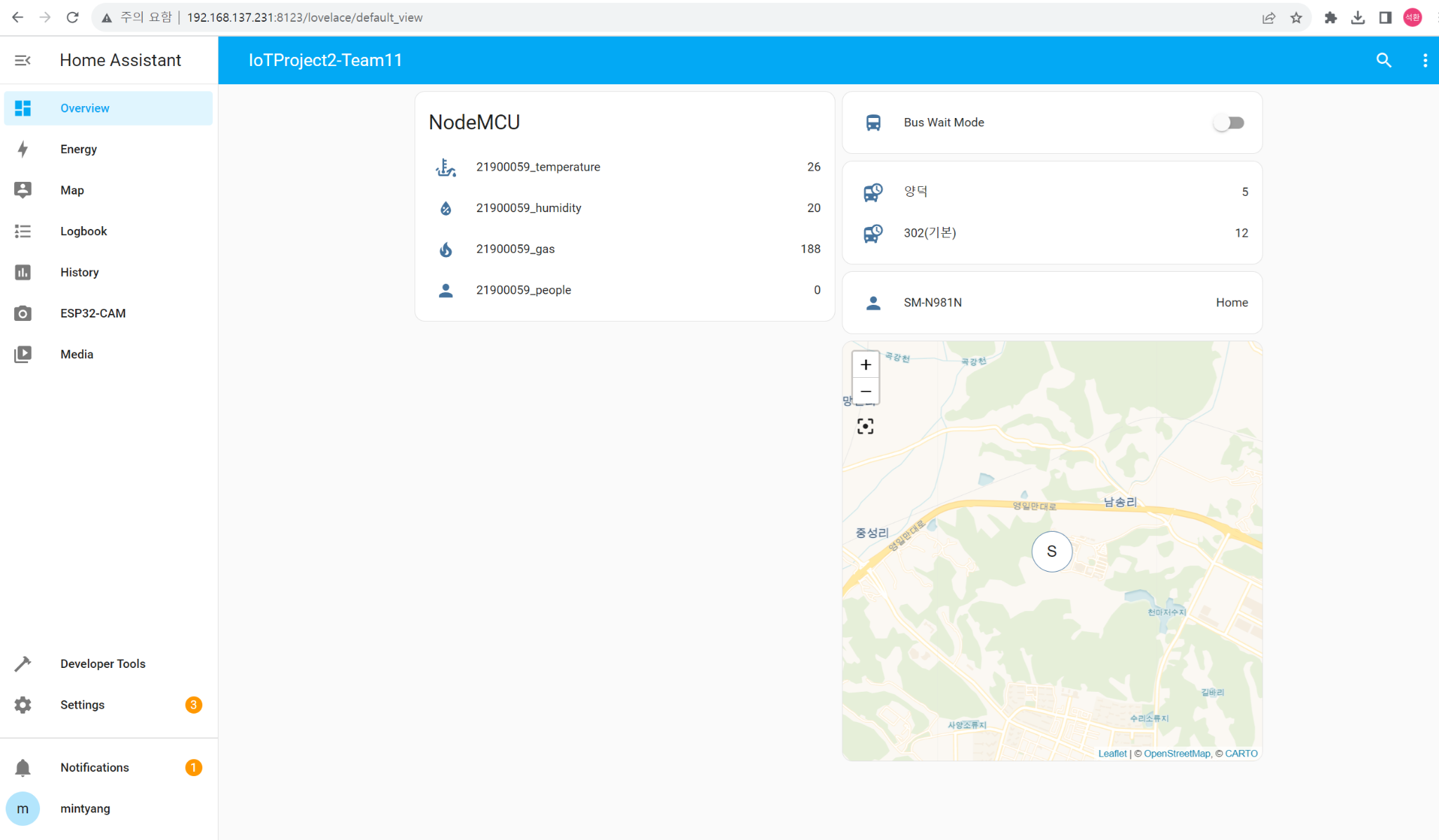
해당 파일은 Home Assistant에서 ui를 조정하기 위한 파일이다. 다양한 카드를 위 코드를 통하여 수정 및 생성할 수 있다.

**2.4. 최종 구현 및 데모 결과**

**2.4.1. HASS UI 및 Cam**

****

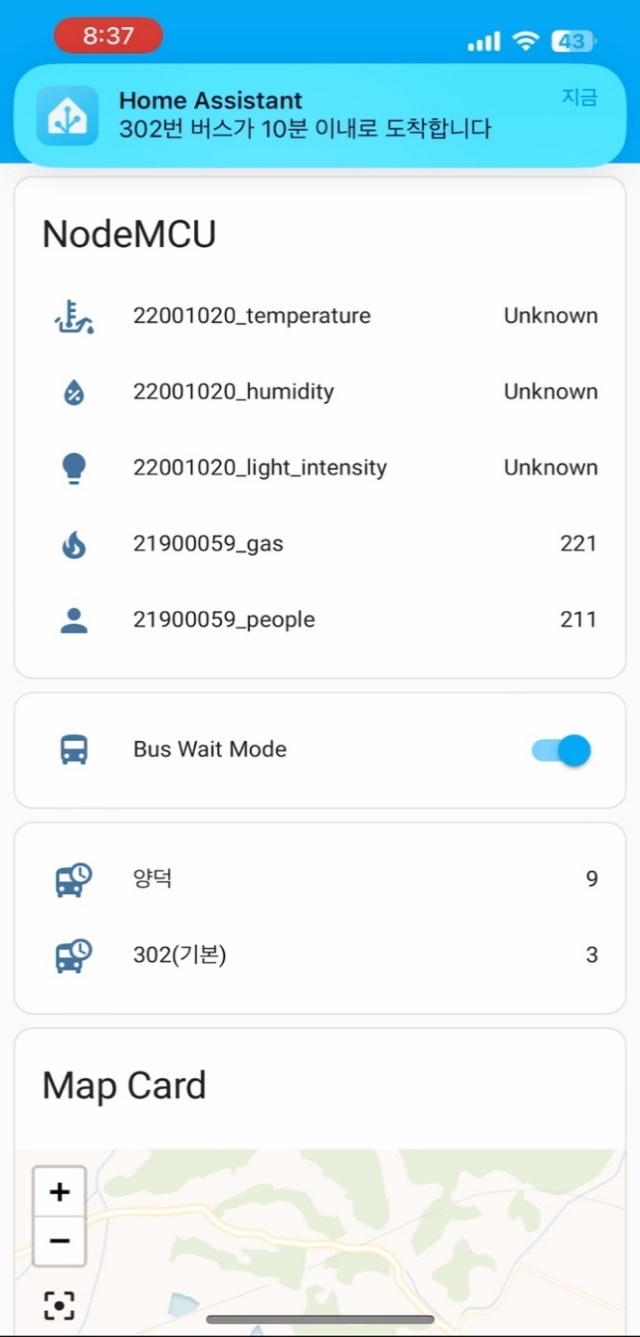
**ESP32 CAM 구동 화면**

****

**HASS Ui**

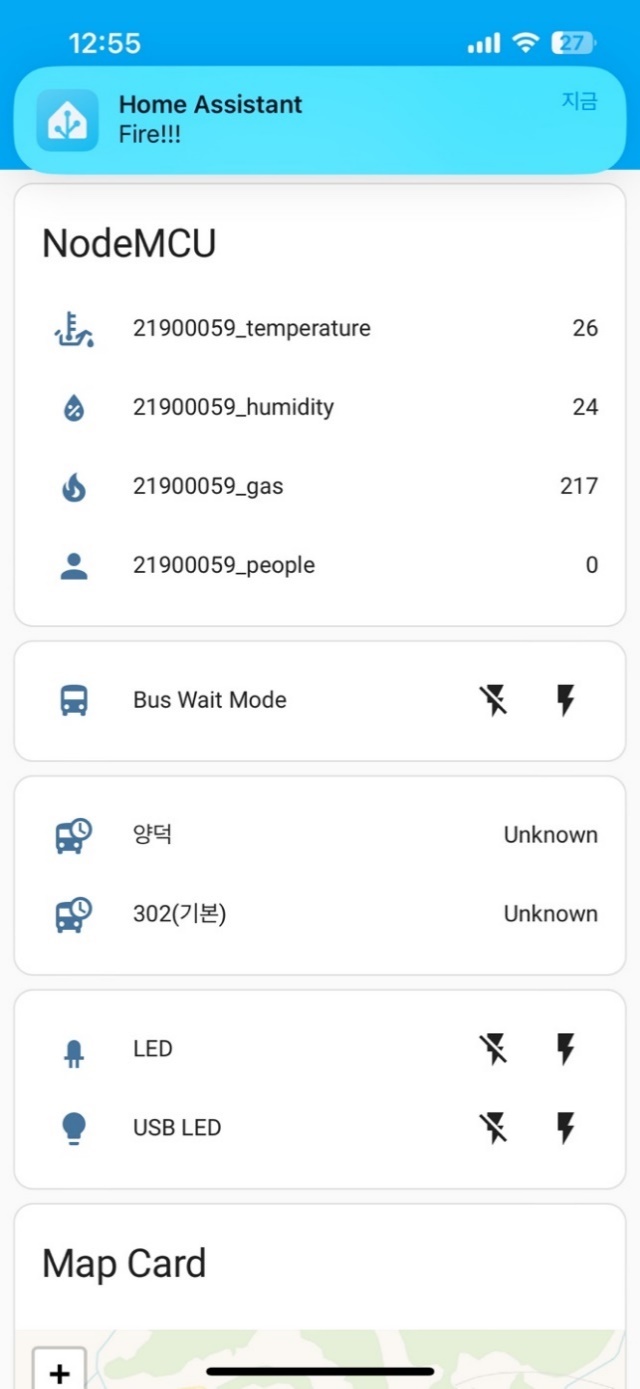
**2.4.2 Alert**

**302(기본) 알림 302번 알림**

**텍스트, 포유류, 개, 실내이(가) 표시된 사진

자동 생성된 설명**

**외부인 침입 알림 화재 경보**

**텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**3. 결론**

**3.1. 작업 중 발생한 문제점**

본래 외부 인원 감지 기능의 경우 이 기능이 아닌, 동아리 방의 인원수를 세는 기능을 구현하려고 하였다. 적외선 센서 2개를 NodeMCU에 설치하여 감지된 순서에 따라 인원이 입장하였는지, 퇴장하였는지를 측정하고, 이를 JSON 형식으로 Raspberry Pi로 보내려고 하였다. 적외선 센서를 활용한 구현을 뒤늦게 고안해 냈기에, 사비로 적외선 센서를 구입하였다. 그러나 적외선 센서가 불량인 탓에, 각종 유튜브나 블로그 등의 외부 자료를 토대로 예제를 똑같이 구현해 봐도 적외선 센서가 물체를 감지하지 못했다. 적외선 센서의 감도를 조절해봐도 항상 물체가 감지된다고 뜨거나, 항상 물체가 감지 되지 않는다고 뜨는 현상만이 발생하였다. 그래서 인체 감지 센서를 개조하고 활용하여 정확한 인원 수의 측정이 아닌 그저 인원 감지 기능만을 구현하고, GPS를 활용하여 HASS에 등록된 사용자가 외부에 있는 경우 외부 인원을 감지하면 스마트폰에 알림을 보내는 기능으로 변경하였다.

**3.2. 팀워크**

* 공통: 보고서 작성
* 정지우(33.3%): 1차 발표 자료 준비, Home/Away GPS 시스템, 교통 정보 API 받아오기, 카메라 구현
* 김가륜(33.3%): 전반적인 NodeMCU 구현. 회로 설계 및 적외선 센서로 인원 감지, 인체 감지 센서로 인원 감지 및 센서 범위 조정을 위한 기구 제작
* 박석환(33.3%): 2차 발표 자료 준비, 전반적인 HASS구현. 각종 yaml 파일 및 자동화, 스마트폰 알림

**3.3. 결과 및 고찰**

본 설계 과제를 통해 IOT 시스템을 실생활에 적용시키고, 동아리 방에서 겪었던 불편했던 점들을 개선하는 기회가 되었다. 특히 HASS를 활용하여 각종 기능들을 간편하고, 세련되게 구현할 수 있었다. 또한 Api를 적극적으로 활용하는 방법을 알 수 있었으며, HASS를 통해 스마트폰에 알림을 보내는 기능 등 각종 유익한 기능을 배울 수 있었던 기회가 되었다.

또한, 예기치 못한 센서의 불량과 같은 문제가 발생했을 때, 팀원들 간 협력을 통해 유사한 기능을 고안하고 구현하는 등의 유연한 대응이 가능했다. 이러한 상황에서 우리는 방향을 조정하고 현명한 문제 해결 방법을 모색하는 데에 있어서 팀 프로젝트 경험의 중요성을 깨달았다. 이를 통해 다양한 경험을 쌓으며 문제를 해결하는 방법에 대해 배울 수 있었다.

Github: <https://github.com/jjw2814/11_-_-_-_-_-_.git>

**4. 참고문헌**

1. Github. 2023/05/12. WifiManager. tzapu.  
   <https://github.com/tzapu/WiFiManager/tree/master/examples>
2. Home Assistant. n.d. Notification. Home Assistant.  
   <https://www.home-assistant.io/integrations/notify/>
3. Youtube. 2020/05/13. Home Assistant How To - Alert integration. BeardedTinker.  
   <https://www.youtube.com/watch?v=KzPy3zeDB5g>
4. Makerspace. 2022/01/04. ESP32 CAM-MB 보드 webserer 사용하기.로니킴.   
   <https://makerspace.steamedu123.com/entry/ESP32-ESP32-CAM-%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0>
5. 네이버 블로그. 2023/02/17. [ESP32 CAM] 아두이노 웹캠 + 업로드 보드 사용법 (웹 스트리밍, AI Thinker, MB). 디바이스마트  
   <https://m.blog.naver.com/no1_devicemart/223018835439>
6. Home Assistant. n.d. GPSLogger. Home Assistant.  
   <https://www.home-assistant.io/integrations/gpslogger>
7. Github. n.d. gpslogger. mendhak.  
   <https://github.com/mendhak/gpslogger>
8. Home Assistant. n.d. iFrame panel. Home Assistant  
   <https://www.home-assistant.io/integrations/panel_iframe/>
9. Tistory. 2020/02/29. [HA] Lovelace UI 기초. psychoria  
   <https://psychoria.tistory.com/722>
10. Home Assistant Community. 2019/11/19. Where is the ui-lovelace.yaml file?. bedfellow  
    <https://community.home-assistant.io/t/where-is-the-ui-lovelace-yaml-file/151744>
11. ChatGPT 3.5. OpenAI  
    <https://chat.openai.com/>
12. 공공데이터포털. 2022/01/24. 국토교통부\_(TAGO)\_버스도착정보. 국토교통부 모빌리티총괄과  
    <https://www.data.go.kr/iim/api/selectAPIAcountView.do>