「스마트 출석부」 시스템

3A 21660072 이경민

3A 21760003 구본성

목 차

Ⅰ. 서 론

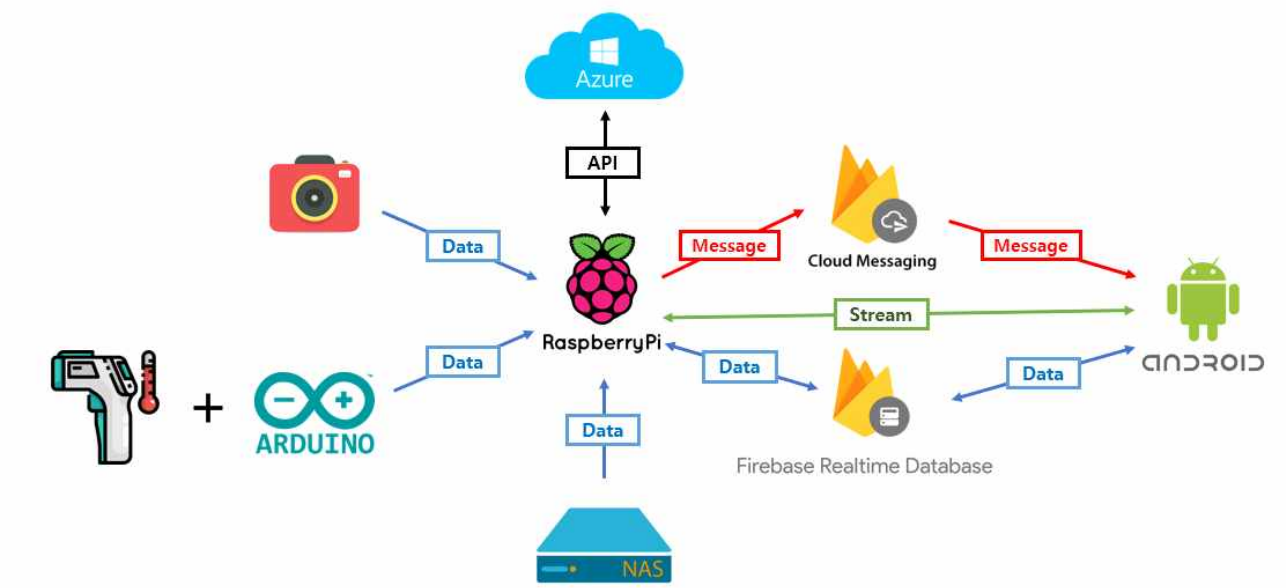
1. 프로젝트 선정 배경 및 목적

[그림] 출석부와 체온 측정 사진

「스마트 출석부」란 기존 오프라인 출석부와 AIoT(사물지능, Artificial Intelligence of Things)를 융합하여 기존의 자동화가 없어 불편한 점들과 불필요한 인력 낭비를 최소화하여 코로나로부터 안전한 환경을 조성하였습니다. 코로나 팬데믹 시대 학교생활에서 AIoT를 이용하여 개선할 수 있는 점이 무엇이 있을까 고민한 결과 감염자의 바이러스가 들어 있는 비말이나 작은 입자를 통해 코로나19가 전염됨에 따라 코로나 의심 환자와의 거리를 두기를 위해 기존의 오프라인 출석과 접촉시 체온 측정 과정을 융합 & 자동화 & 비대면화 하여 비대면 안면인식을 사용한 온라인 출석 & 비대면 안면인식 자동 온도측정 시스템 구축으로 개선할 수 있게 되었습니다.

또한 교수자의 입장에서 편의성을 높이기 위해 안드로이드 애플리케이션을 통해 클라이언트로 확인하고 제어할 수 있는 기능들을 추가하였습니다. 이러한 기능들을 만듦으로써 학생과 관리자의 편의, 보안, 안전 등을 제공하는 것을 목적으로 프로젝트를 시작하게 되었습니다.

2. 프로젝트 구현 개요



[그림] 프로젝트 전체적 개요

프로젝트의 전체적인 흐름을 간략히 살펴보자면

1. Android Application에서 소켓통신을 이용해 Raspberry PI에 출석 오픈을 요청합니다.
2. 출석이 오픈되면 Raspberry PI에 달려있는 카메라를 이용해 Stream을 엽니다.
3. Stream에서 OpenCV의 얼굴 인식을 통해 얼굴이 인식될때까지 기다립니다.
4. 얼굴이 인식되면 NAS에 있는 Target Image와 인식된 얼굴(Source Image)을 Azure API를 이용해 비교합니다.
5. Target Image와 Source Image가 일치하는 얼굴이 있다면 Arduino에 온도 측정을 요청합니다.
6. Arduino와 Raspberry PI는 시리얼 통신을 통해 온도정보를 주고받습니다.
7. Raspberry PI는 받은 온도정보와 얼굴 인식 정보, 날짜, 결과코드 등을 학번을 PK로 하는 JSON 형식으로 가공해 Firebase에 업로드 합니다.
8. Firebase DB에 변동사항이 생기면 Cloud Messaging Service를 이용해 교수자에게 Push 알림을 보냅니다.
9. 교수자는 Android Application을 통해 출석정보를 열람하고 출석을 종료할 수 있습니다.

2.1 사용 라이브러리

2.1.1 Standard Library

configparser : 소스와 설정파일 분리를 위해 사용.

threading : thread 사용.

os : os 명령어 실행.

datetime : 현재 날짜 체크와 날짜형식 지정을 위해 사용.

time : sleep 기능을 위해 사용.

socket : Android – Raspberry PI 소켓통신을 위해 사용.

json : Firebase에 JSON형식으로 업로드하기 위해 사용.

2.1.1 3rd party Library

azure : Azure API 사용.

cv2 : openCV 영상처리 사용.

pyserial : Raspberry PI – Arduino Serial 통신 사용.

simpleaudio : 소리 출력

pyfcm : Firebase Cloud Messaging Notification. 클라우드 메시징 사용

firebase_admin : Firebase 사용.

requests : NAS에 접근해 데이터 사용.

2.1.1 Local Application Library

ArduinoSerialProtocol : Arduino와 시리얼 통신 관련 기능 구현.

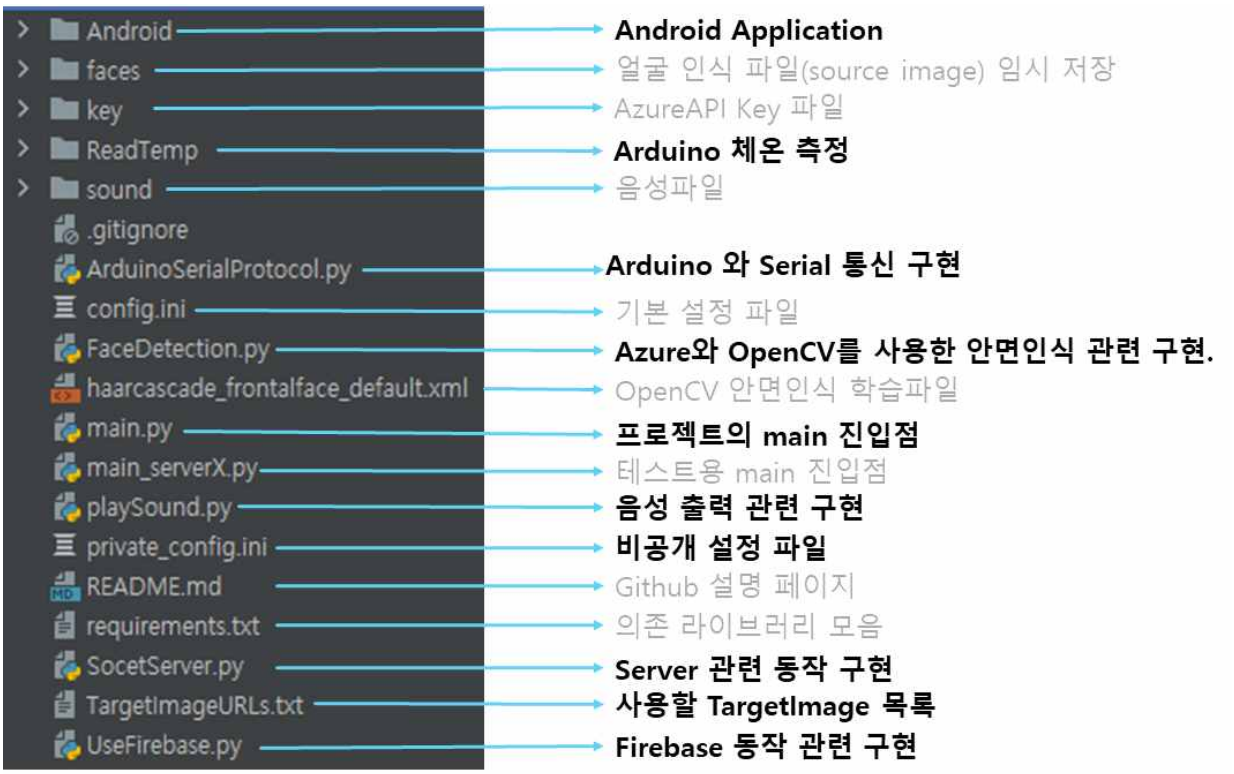
FaceDetection : 얼굴 인식, 비교 관련 기능 구현.

playSound : 사운드 출력 관련 기능 구현.

SocetServer : Android 소켓통신 관련 기능 구현.

UseFirebase : Firebase 관련 기능 구현.

2.2 프로젝트 구조



프로젝트 구조는 위와 같습니다.

다음으로 프로젝트 구조입니다.

main 파일을 중심으로 기능들을 기준으로 모듈화를 구현해 기능 추가나 변경 등에 빠르게 대응할 수 있습니다.

모듈화한 기능들을 간략히 살펴보자면

- Android 어플리케이션
- Arduino 체온측정 기능
- Arduino와 라즈베리 파이간 시리얼 통신 기능
- Azure와 OpenCV를 사용한 안면인식, 비교 기능
- 음성 출력 기능
- Server 관련 동작 기능
- Firebase 관련 동작 기능

으로 기능 모듈화를 구현했으며, 또한 코드와 리소스를 config파일, TargetImageURL 파일 private config파일등과 같이 구분하여 구현 하였습니다.

Ⅱ. 본 론

1. Raspberry Pi와 Arduino 연동

[그림] Raspberry Pi 와 Arduino 연동

Arduino Uno 보드에는 적외선 온도측정 센서를 연결하여 센서 값을 Raspberry Pi에 전달하도록 하였고, Raspberry Pi는 Arduino에서 전달된 값과 AzureAPI에서 얼굴 비교를 통해 얻어온 데이터를 바탕으로 하여 Firebase DB에 넣을 값을 JSON 형식으로 가공합니다.

Arduino에서 측정된 센서의 값을 Raspberry Pi에 전달하는 방법으로는 Raspberry Pi와 Arduino간 Serial Communication 방식을 사용하였습니다. Serial Communication 방식을 사용하기 위해 Raspberry Pi에 Arduino를 연결한 뒤 (USB Cable), Arduino 포트를 설정해주어야 합니다. 또, Arduino가 넘기는 값을 Raspberry Pi의 Python 환경에서 값을 받고 처리할 수 있도록 pyserial 라이브러리를 설치하여 사용했습니다.

```python
class ArduinoSerialProtocol:
    config = configparser.ConfigParser()
    config.read("config.ini", encoding="utf-8")
    data = []
    __instance = None
    connected = False

    try:
        ser = serial.Serial(
            port=config["Arduino"]["port"],
            baudrate=config["Arduino"]["baudrate"],
            timeout=1
        )
        connected = True
    except serial.SerialException as se:
        print(se)
        connected = False
```

Python Connection Code (Raspberry Pi)

```
void loop() {
  // put your main code here, to run repeatedly:

 while(Serial.available()){   //시리얼에 읽을 값이 있으면
   //Serial.println(Serial.available());
   digitalWrite(4, HIGH);    //초록불 켜기
   delay(1000);
   digitalWrite(4, LOW);    //초록불 켜기
   income += (char)Serial.read(); //income안에 해당 내용 저장
   if(income == 's'){
     break;
   }
   //Serial.println(income);//시리얼에 해당 내용 전송
 }
 if(income != 0){ //income에 내용이 있으면

   if(income == 's'){
     //Serial.println("1");
     while (count++ < 5){
       if(Timer1_Flag){                // 50ms마다 반복 실행(Timer 1 Flag check)
         Timer1_Flag = 0;                          // Reset Flag
         iOBJECT = SPI_COMMAND(OBJECT,DELAY_10us);     // 대상 온도 Read
         delayMicroseconds(10);                    // 10us : 이 라인을 지우지 마세요
         iSENSOR = SPI_COMMAND(SENSOR,DELAY_10us);     // 센서 온도 Read
         delayMicroseconds(10);                    // 10us : 이 라인을 지우지 마세요
         cEratio = SPI_COMMAND(ERATIO,DELAY_35us);     // 방사율 READ, 반드시 35us

         //Serial.print("Object Temp : ");             // 하이퍼터미널 출력
         Serial.println((float(iOBJECT)/10 + 6.5),1);
         //Serial.print("     Sensor Temp : ");
         //Serial.print(float(iSENSOR)/10,1);
         //Serial.print("    Eratio : ");
         //Serial.println(float(cEratio)/100,2);
         delay(500);
         SPI_COMMAND(LASER,DELAY_10us);
       }
       /*if(Laser_Flag == 1){
         Laser_Flag = 0;
         SPI_COMMAND(LASER,DELAY_10us);
         while(digitalRead(2)==LOW);
         attachInterrupt(digitalPinToInterrupt(2), LASER_ISR, FALLING);
       }*/
     }
     LASER_ISR();
     count = 0;
   }
   income = '\0'; //전송한 income내용 초기화
 }
}
```

Arduino Connection Code

먼저 Raspberry PI 코드를 살펴보면, 소스와 리소스 분리를 위해 ConfigParser를 사용해 외부에서 설정값을 불러와 Arduino와 Connection을 생성합니다.

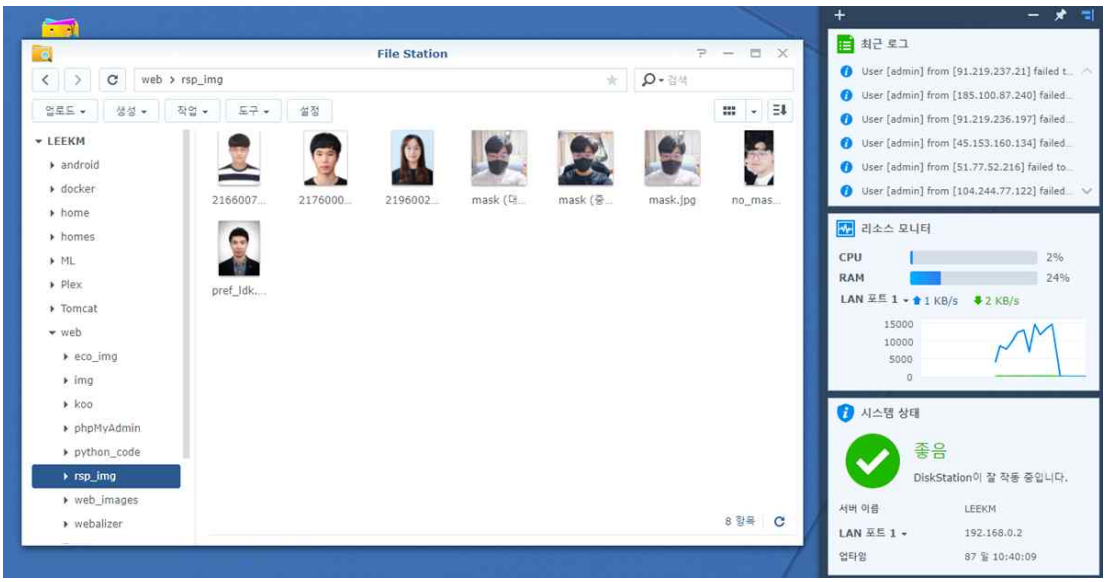
이후 생성된 Connection으로 Arduino에게 온도를 측정해달라는 신호를 보내고, 신호를 보내면 아두이노에서 신호에 해당하는 동작을 한 후, 결과를 Raspberry PI로 보냅니다.

시리얼 통신에 대한 부분을 설명하자면, Raspberry PI에서 Serial.write()를 통해 Arduino에게 데이터를 전송하고, Arduino의 Serial.print()를 이용하여 센서의 값을 측정하여 작성, Raspberry Pi에서는 Serial.readline()을 통해 개행문자 (‘\n’)까지의 Arduino에서 전달한 문자열을 읽어오는 방식입니다. 또한 체온 측정에 약간의 텀을 주기위해 sleep()을 사용하였습니다.

2. Raspberry Pi와 Private Cloud(NAS) 연동

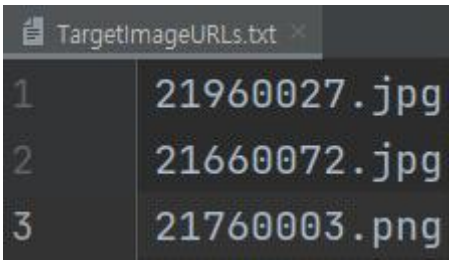
[그림] Raspberry Pi 와 Private Cloud(NAS)연동

Private Cloud의 경우에는 얼굴 비교에서 Target Image 역할을 할 사진들을 저장하기 위해 사용하였습니다.



[그림] Private Cloud안에 저장된 Taget Image

위 사진처럼 Target Image를 NAS에 넣어둔 후



[그림] 사용할 Target Image목록

비교를 할 사진의 이름을 외부 리소스로 저장하면 해당 파일이름에 맞는 사진들을 불러와 Target Image로 사용합니다.

```python
def init_target_faces(self):
    # Detect faces from target image url list, returns a list[DetectedFaces]
    for image_file_name in self.target_image_file_names:
        # We use detection model 3 to get better performance.
        detected_faces = self.face_client.face.detect_with_url(self.IMAGE_BASE_URL + image_file_name,
                                                               detection_model='detection_03')
        # Add the returned face's face ID
        for detected in detected_faces:
            self.detected_faces_ids.append(detected.face_id)
    print("target faces initialize complete")
    return detected_faces
```

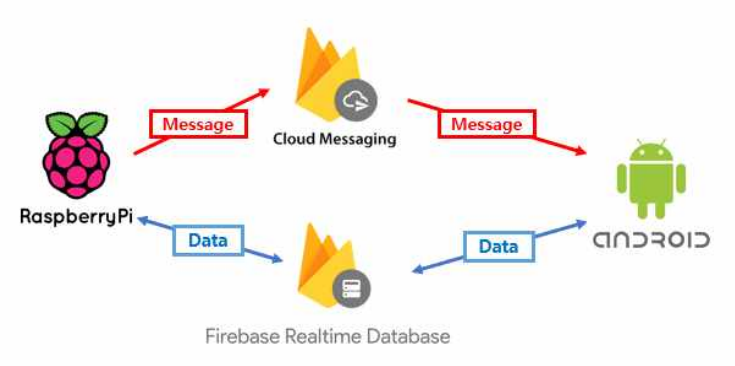
[그림] Python Code

```python
# 클라이언트 토큰을 NAS에서 다운로드 후 저장.
with open("client_token.txt", "wb") as file:  # open in binary mode
    response = get(private_config["firebase_cloudmessaging"]["TOKEN_URL"])  # get request
    print("download")
    file.write(response.content)  # write to file
```

[그림] 사용할 Target Image목록

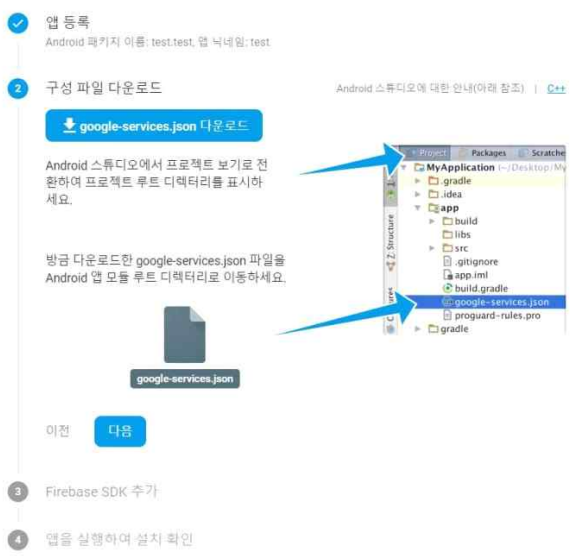
또한 보안을 위해 Client Tkoen은 requests 모듈을 이용하여 NAS에서 Client Token을 불러와서 사용합니다. Client Token의 자세한 내용은 5.1.1절에서 살펴보겠습니다.

3. Firebase 연동



[그림] Firebase 연동

Android와 Raspberry Pi의 원활한 연동 및 기능 수행을 위해 Google에서 제공해주는 Firebase를 사용하였습니다. Firebase를 사용하기 위해서는 Google의 Firebase 홈페이지에 접속 후, Firebase 프로젝트를 생성한 뒤, Web 또는 Android / iOS 환경의 어플리케이션을 Firebase 프로젝트에 추가하여 사용하는 형식입니다. 스마트 출석부 시스템은 Android 환경을 사용하므로 Android 어플리케이션을 추가하여 Firebase가 제공하는 기능을 이용할 수 있도록 등록합니다.



[그림] Firebase에 Android App 등록 – 1

[그림] Firebase에 Android App 등록 – 2

[그림] Firebase에 Android App 등록 – 3

[그림] Firebase에 Android App 등록 – 4

4. Raspberry Pi와 Android 연동

[그림] Raspberry Pi 와 Arduino 연동

RaspberryPI와 Android 연결에는 Java와 Python간의 소켓 통신을 이용해서 구현하였습니다.

```python
def __init__(self):
    # 소켓을 만든다.
    self.data = []
    self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 소켓 레벨과 데이터 형태를 설정한다.
    self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # 서버는 복수 ip를 사용하는 pc의 경우는 ip를 지정하고 그렇지 않으면 None이 아닌 ''로 설정한다.
    # 포트는 pc내에서 비어있는 포트를 사용한다. cmd에서 netstat -an | find "LISTEN"으로 확인할 수 있다.
    self.server_socket.bind(('', 9999))
    self.server_state = False

# binder함수는 서버에서 accept가 되면 생성되는 socket 인스턴스를 통해
# client로 부터 데이터를 받으면 echo형태로 재송신하는 메소드이다.
def binder(self, client_socket, addr):
    # 커넥션이 되면 접속 주소가 나온다.
    print('Connected by', addr)
    try:
        self.server_state = True
        # 접속 상태에서는 클라이언트로 부터 받을 데이터를 무한 대기한다.
        # 만약 접속이 끊기게 된다면 except가 발생해서 접속이 끊기게 된다.
        while True:
            # socket의 recv함수는 연결된 소켓으로부터 데이터를 받을 대기하는 함수입니다. 최초 4바이트를 대기합니다.
            data = client_socket.recv(4)
            # 최초 4바이트는 전송할 데이터의 크기이다. 그 크기는 little big 엔디언으로 byte에서 int형식으로 변환한다.
            # C#의 BitConverter는 big엔디언으로 처리된다.
            length = int.from_bytes(data, "little")
            # 다시 데이터를 수신한다.
            data = client_socket.recv(length)
            # 수신된 데이터를 str형식으로 decode한다.
            msg = data.decode()
            # 수신된 메시지를 콘솔에 출력한다.
            if msg:
                self.data.append(msg)
            if msg == "end":
                self.server_state = False
                return
            print('Received from', addr, msg)
            # 수신된 메시지 앞에 「echo:」 라는 메시지를 붙힌다.
            msg = "echo : " + msg
            # 바이너리(byte)형식으로 변환한다.
            data = msg.encode()
            # 바이너리의 데이터 사이즈를 구한다.
            length = len(data)
            # 데이터 사이즈를 little 엔디언 형식으로 byte로 변환한 다음 전송한다.
            client_socket.sendall(length.to_bytes(4, byteorder='little'))
            # 데이터를 클라이언트로 전송한다.
            client_socket.sendall(data)

    except Exception as e:
        # 접속이 끊기면 except가 발생한다.
        print("except : ", e)
        print("addr:",addr)
        #self.server_state = False
    finally:
        # 접속이 끊기면 socket 리소스를 닫는다.
        client_socket.close()
```

Python Connection Code (Raspberry Pi)

```java
public class SocketProcess {
    public static void sendMsg(String result) {
        // 소켓을 선언한다.
        try (Socket client = new Socket()) {
            // 소켓에 접속하기 위한 접속 정보를 선언한다.
            InetSocketAddress ipep = new InetSocketAddress(AppData.SERVER_IP, port: 9999);
            // 소켓 접속!
            client.connect(ipep);
            // 소켓이 접속이 완료되면 inputstream과 outputstream을 받는다.
            try (OutputStream sender = client.getOutputStream(); InputStream receiver = client.getInputStream();) {
                // 메시지는 for 문을 통해 10번 메시지를 전송한다.

                    // 전송할 메시지를 작성한다.
                    String msg = result;
                    //String msg = "end";
                    // string을 byte배열 형식으로 변환한다.
                    byte[] data = msg.getBytes();
                    // ByteBuffer를 통해 데이터 길이를 byte형식으로 변환한다.
                    ByteBuffer b = ByteBuffer.allocate(4);
                    // byte포맷은 little 엔디언이다.
                    b.order(ByteOrder.LITTLE_ENDIAN);
                    b.putInt(data.length);
                    // 데이터 길이 전송
                    sender.write(b.array(), off: 0, len: 4);
                    // 데이터 전송
                    sender.write(data);
                    data = new byte[4];
                    // 데이터 길이를 받는다.
                    receiver.read(data, off: 0, len: 4);
                    // ByteBuffer를 통해 little 엔디언 형식으로 데이터 길이를 구한다.
                    b = ByteBuffer.wrap(data);
                    b.order(ByteOrder.LITTLE_ENDIAN);
                    int length = b.getInt();
                    // 데이터를 받을 버퍼를 선언한다.
                    data = new byte[length];
                    // 데이터를 받는다.
                    receiver.read(data, off: 0, length);
                    // byte형식의 데이터를 string형식으로 변환한다.
                    msg = new String(data, charsetName: "UTF-8");
                    // 콘솔에 출력한다.
                    System.out.println(msg);

                }
            } catch (Throwable e) {
                e.printStackTrace();
            }
        }
    }
```

Java Code (Android)

통신 소켓으로는 9999포트를 지정하였으며, 학교 내부에 서버가 위치할때는 학교 방화벽 문제로 소켓통신이 어려웠으나 공유기를 이용해 같은 네트워크에 위치하도록 조치하여 방화벽 부분은 해결하였습니다.

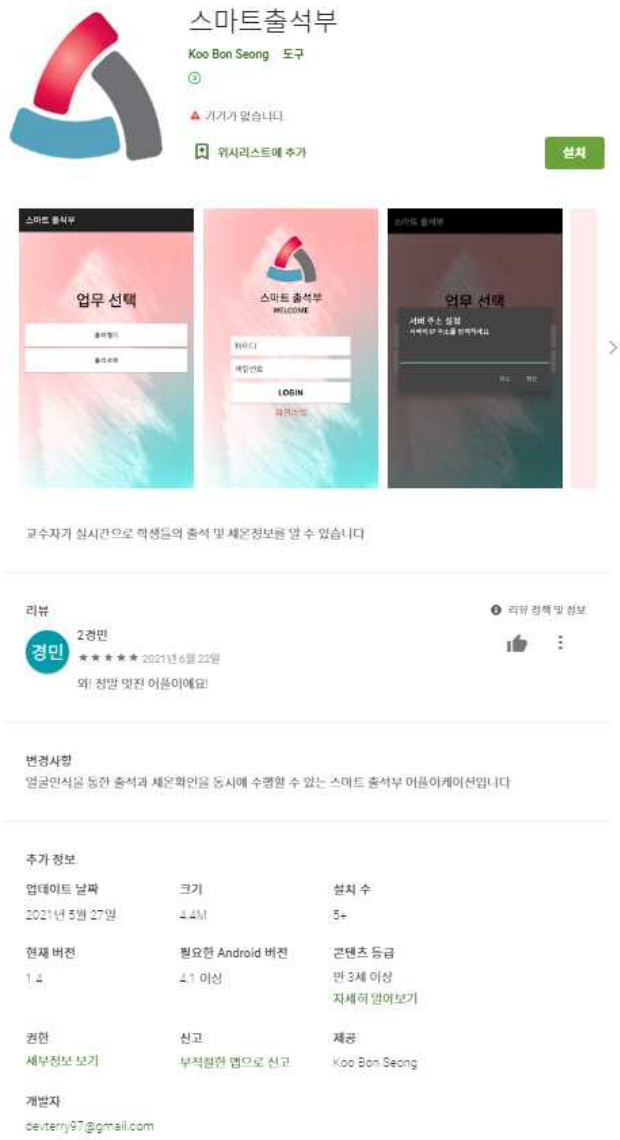
Python(Server)의 경우 socket.socket(IP, SOCKET)을 이용해 Connection을 열고,

bind()를 통해 서버로 들어오는 데이터(출석 open 정보)를 수신하도록 구성하였습니다.
또한 내부적으로 main 코드안에 while !ss.data의 조건으로 반복문을 적용시켜 데이터가 없다면 data필드를 이용해 데이터를 입력받을때까지 기다리고, Java쪽에서 데이터를 준다면 작업을 진행하도록 작성하였습니다.

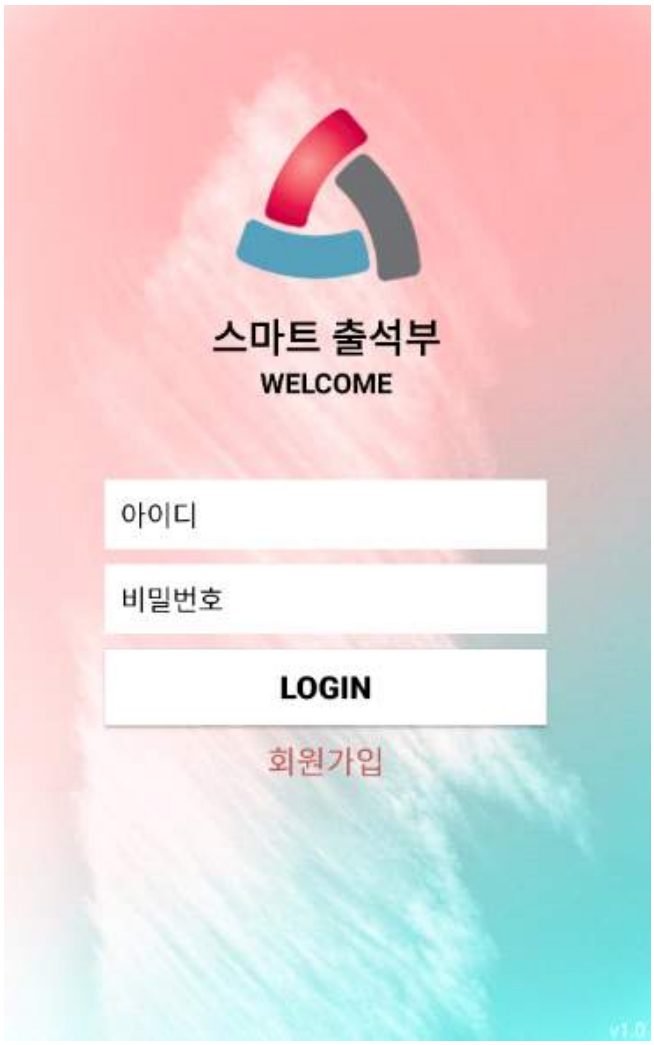
5. 구현기능

5.1 안드로이드 어플리케이션

교수자의 입장에서 편의성을 높이기 위해 안드로이드 애플리케이션을 통해 출석 결과와 체온 검사 결과를 클라이언트(Android 스마트폰)로 확인하고 출석 열기, 출석 닫기 기능을 직접 제어할 수 있도록 구현하였습니다. 현재 완성된 어플리케이션은 Google Play Store에서 「스마트 출석부」 로 설치 할 수 있습니다.



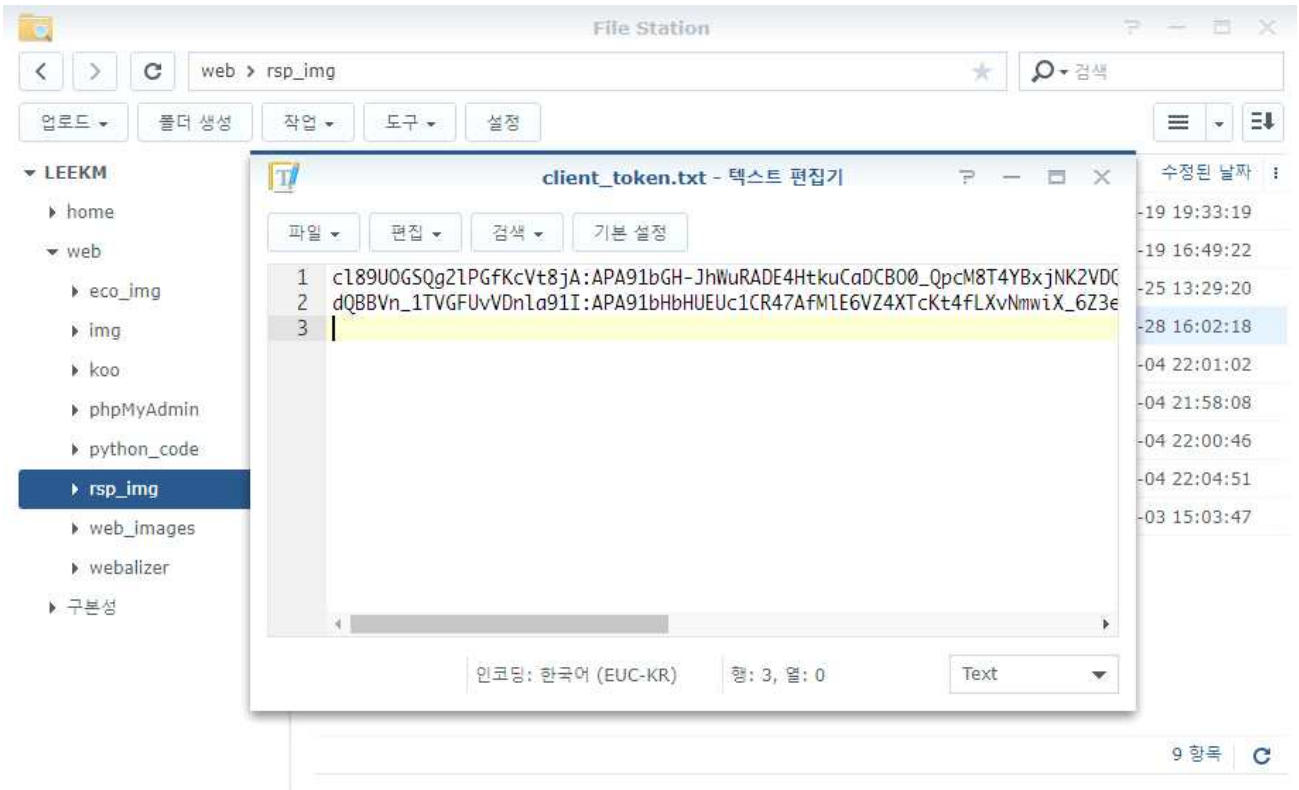
[그림] Google Play Store



[그림] 「스마트 출석부」 어플리케이션

5.1.1 클라이언트 확인

접근성과 보안성을 높이기 위해 Google Play Store에서 「스마트 출석부」 어플리케이션을 설치하고 어플리케이션을 사용하기 위해서는 로그인 과정에서 사전에 승인된 클라이언트를 통해 어플리케이션 사용이 허가된 교수자가 접속하였는지 접속한 클라이언트의 TOKEN 과 프라이빗 클라우드에 저장된 승인된 클라이언트의 TOKEN 목록을 비교해 확인하는 과정이 수행됩니다.



[그림] 프라이빗 클라우드 client_token 파일

5.1.2 출석 열기

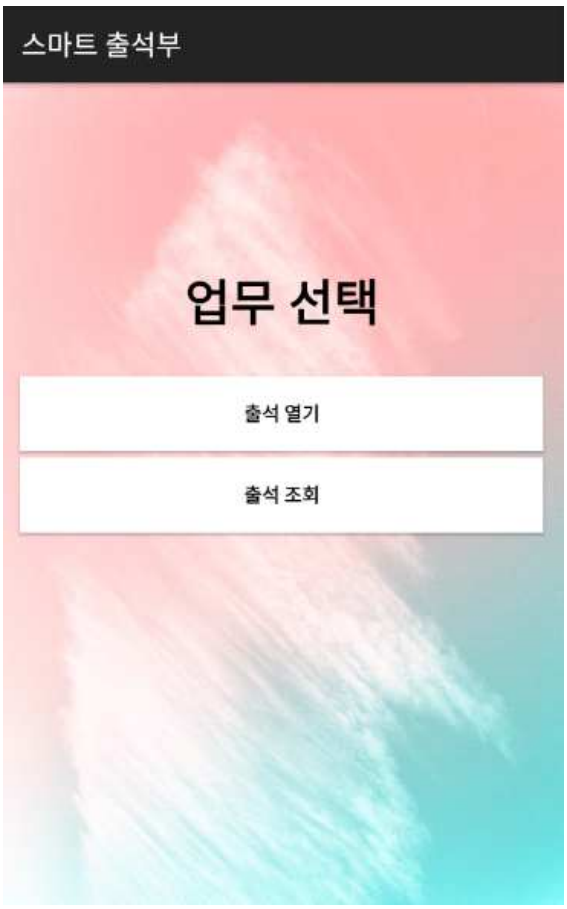
클라이언트 확인 과정이 성공적으로 수행되어 로그인 수행이 완료되면 업무 선택란에서 출석 열기 기능과 출석 조회 기능을 선택할 수 있습니다.

출석 열기를 수행하기 위해서는 서버 주소를 직접 선택하여 출석 열기를 수행하는 데 이는 필요에 따라 여러 개의 서버를 동시에 사용하여 출석 열기를 수행하기 위함입니다.

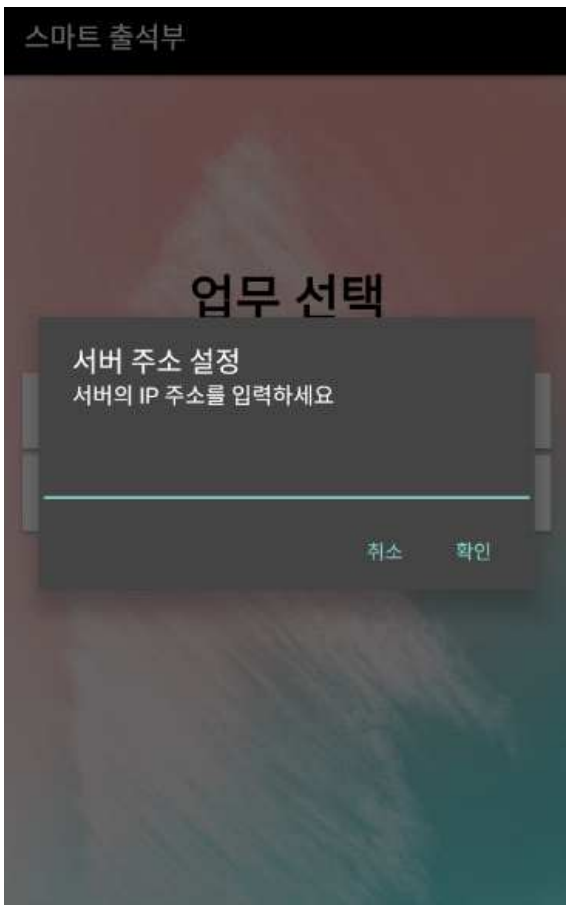
[그림3]과 같이 출석을 수행할 과목과 교시를 선택한 후 출석 열기 버튼을 선택합니다.

출석 열기를 수행하면 AlertDialog를 띄어서 사용자로부터 출석을 열어둘 시간을 선택하게 합니다. 출석이 성공적으로 열리게 되면 Real-Time Database에 해당 출석에 해당하는 날짜_교시_과목코드 형식으로 스키마를 [그림5]와 같이 생성합니다.

성공적으로 DB스카마가 생성되었을 경우 소켓통신을 사용하여 Raspberry PI로 데이터를 보내 카메라와 적외선 온도계를 작동시킵니다.



[그림1] 기능 선택



[그림2] 출석 열기

[그림3] 출석 열기

[그림4] 시간 선택

[그림5] Real TIme Database

5.1.3 출석 조회

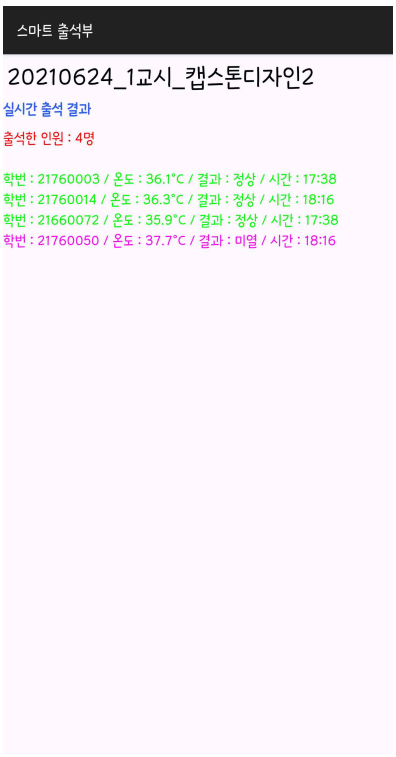
출석 조회 기능도 마찬가지로 클라이언트 확인 과정이 성공적으로 수행되어 로그인 수행이 완료되면 업무 선택란에서 출석 열기 기능과 출석 조회 기능을 선택할 수 있습니다.
단순 출석을 조회하는 업무에서는 라즈베리파이를 구동시킬 필요가 없기 때문에 서버주소를 선택할 필요 또한 없습니다.

조회하고자 하는 과목, 교시, 날짜를 선택하여 출석 결과를 안드로이드 어플리케이션에서 [그림3], Real Time Database에서 [그림4] 와 같이 실시간으로 조회할 수 있습니다.

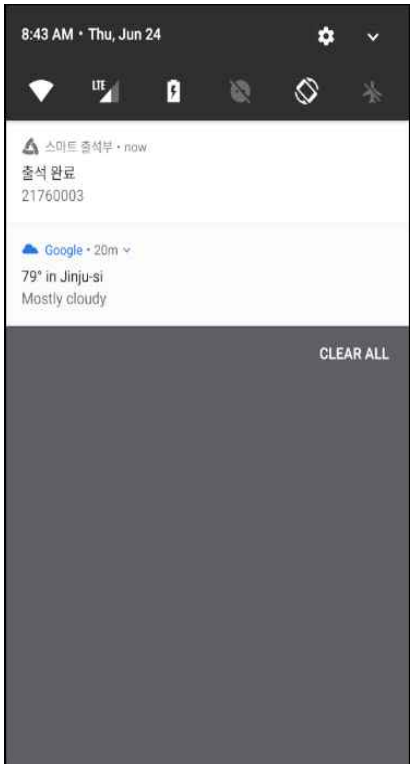
안드로이드 어플리케이션에서도 Real-Time Database와 마찬가지로 DB 데이터의 변화가 변화가 발생할때마다 Callback Method를 사용하여 출석결과 업데이트와 동시에 Cloud Messaging을 사용하여 교수자의 클라이언트에 [그림 5]와 같이 Push 알림을 통해 문제(미열, 고열)가 있는 학생이 출석하였을 경우 즉각 대응할 수 있도록 구현하였습니다.

[그림1] 기능 선택

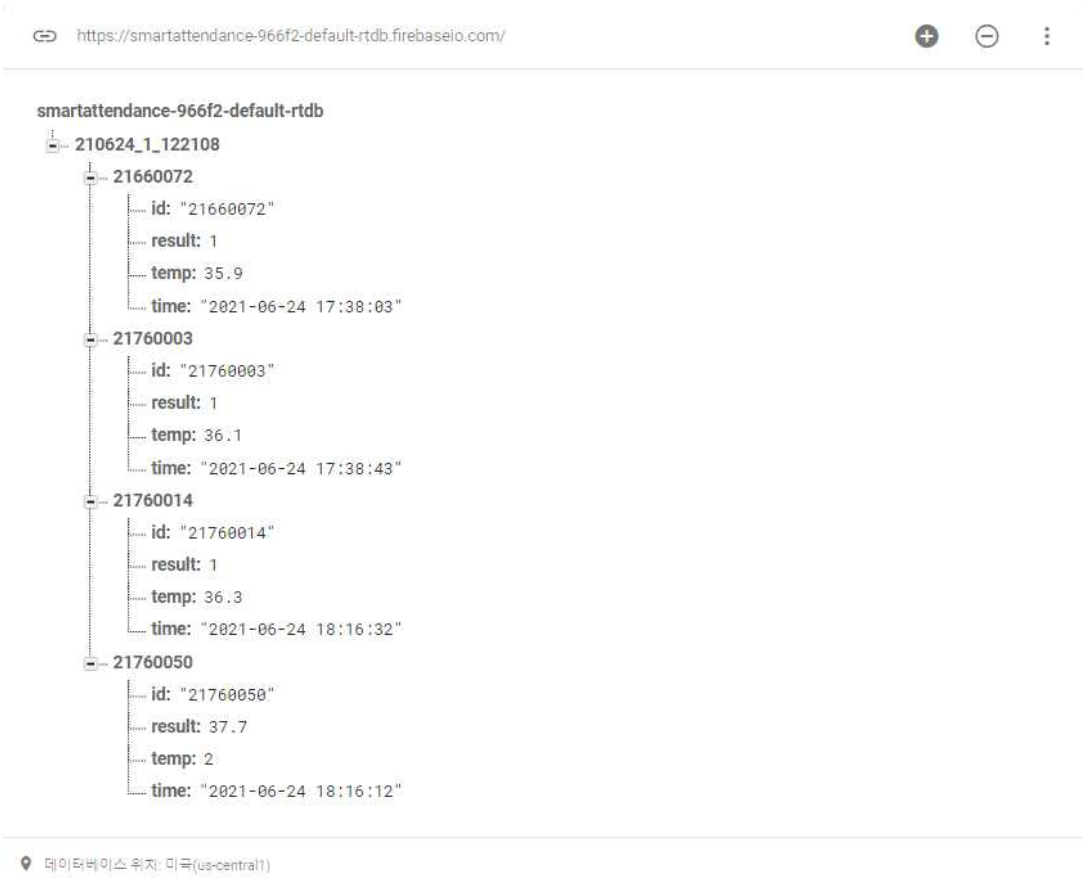
[그림2] 출석 조회



[그림3] 안드로이드에서 출석 조회



[그림4] Cloud Messaging Push 알림



[그림5] Real Time Database에서 출석 조회

5.2 OpenCV 활용

먼저 얼굴인식을 위해 OpenCV를 활용하였습니다.

```
# 얼굴 인식용 xml 파일
self.face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

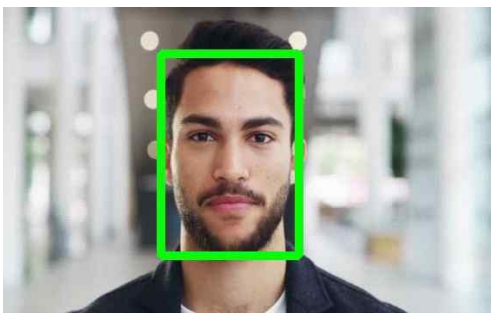
[그림] 얼굴 인식이 학습된 xml을 불러와 얼굴 인식 기능 객체를 만드는 코드

얼굴 인식을 위한 학습 데이터가 있는 xml파일을 불러와 얼굴 인식을 위한 객체를 만듭니다.

```
# 얼굴 인식 함수
def face_extractor(self, img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = self.face_classifier.detectMultiScale(gray, 1.1, 5)

    # 찾은 얼굴이 없으면 None 리턴
    if faces is ():
        return None

    # 찾은 얼굴이 있으면 얼굴을 잘라서 cropped_face에 넣어서 리턴
    for (x, y, w, h) in faces:
        cropped_face = img[y:y + h, x:x + w]
    return cropped_face
```

[그림] 얼굴 인식 메서드

얼굴 인식을 위해 라즈베리파이의 카메라 모듈의 실시간 이미지 stream을 내부적으로는 흑백으로 변환하여 얼굴 인식을 한 후, 인식된 얼굴이 있다면 좌표를 구해 리턴해 줍니다.



[그림] openCV 얼굴 인식 사진

그 후 인식된 얼굴을 Azure API에 넘겨 얼굴 비교 작업을 수행합니다.

5.3 Azure API 활용

앞선 중간발표에서도 언급했듯이 얼굴 비교를 위해 YOLO등의 툴을 통해 직접 학습을 시키기 보다는 Azure API를 사용했습니다. Azure API를 사용하면, 단 한 장의 Target Image만 가지고도 같은 인물인지 판별이 가능하므로, 실제 학교에 도입한다고 가정했을 때 학생 한명한명을 모두 학습시키는 것이 아니라 출석부 사진 한 장만 가지고도 판별이 가능하다고 생각하였고, 실제 출석부 사진을 Target Image로 사용해서 해본결과 성공적으로 작동이 되어 Azure API를 선택하였습니다.

[그림] Azure 얼굴 비교

```
[AZURE]
# 프리티어
#KEY = 33bfc96e88064b7[redacted]
#END_POINT = https://leekm.cogniti[redacted]

# 스탠다드
KEY = 5b3e3f2f73ac47a9[redacted]
END_POINT = https://21660072.c[redacted]

IMAGE_BASE_URL = http://leekm09[redacted]
```

[그림] Priavte 설정파일

```
    def __init__(self):
        self.config = configparser.ConfigParser()
        self.config.read("config.ini", encoding="utf-8")

        private_config = configparser.ConfigParser()
        private_config.read("private_config.ini", encoding="utf-8")

        # 얼굴 인식용 xml 파일
        self.face_classifier = cv2.CascadeClassifier(
'haarcascade_frontalface_default.xml')

        # This key will serve all examples in this document.
        self.KEY = private_config["AZURE"]["KEY"]

        # This endpoint will be used in all examples in this quicksta
        self.ENDPOINT = private_config["AZURE"]["END_POINT"]

        # Azure 객체 생성
        self.face_client = FaceClient(self.ENDPOINT,
CognitiveServicesCredentials(self.KEY))
```

[그림] Azure 설정

Azure API 사용은 Microsoft Console에서 구독을 한 후 Key를 발급받고, FaceClient의 인자로 Key와 EndPoint를 전달하여 객체를 생성한 후 사용할 수 있습니다.

역시 보안적인 이유로, private config로 따로 구분을 하였고 해당 config는 .gitignore에 추가해 github에 자동으로 올라가지 않도록 설정하였습니다.

```
def verify_face_to_face(self):
    for i in range(len(self.source_image_id)):
        for j in range(len(self.detected_faces_ids)):
            verify_result = self.face_client.face.verify_face_to_face(self.source_image_id[i][0],
                                                                      self.detected_faces_ids[j])
            if verify_result.is_identical:
                print('Faces from {} & {} are of the same person, with confidence: {}'
                      .format(self.source_image_id[i][1], self.target_image_file_names[j],
                              verify_result.confidence * 100))

                # json data 가공
                # 21660072.jpg면 21660072를 id로
                self.json_data["result"] = True
                self.student_id = self.target_image_file_names[j].split(".")[0]
                self.json_data[self.student_id] = {}
                self.json_data[self.student_id]["id"] = self.student_id
                now = datetime.datetime.now()
                now_date_time = now.strftime('%Y-%m-%d %H:%M:%S')
                self.json_data[self.student_id]["time"] = now_date_time

                print(json.dumps(self.json_data, ensure_ascii=False, indent="\t"))
                return self.student_id, self.json_data
            else:
                self.json_data["result"] = False
                print('Faces from {} & {} are of a different person, with confidence: {}')
                # playsound("sound/audio_7.mp3")
    return False
```

[그림] Azure를 활용한 얼굴비교 코드

얼굴 비교의 경우 얼굴인식을 통해 전달받은 이미지를 verify_face_to_face()라는 메소드를 통해 Azure API를 통하여 Target Image와 source Image의 얼굴 비교작업을 실시합니다.

일치한다고 판단되는 데이터가 있다면, 학번으로 저장해놓은 이미지 파일의 학번을 파싱하여 PK로 사용하고, 측정 시간을 포함한 JSON 형태로 가공합니다.

5.4 온도 측정 기능

온도 측정기능은 적외선 온도 측정 센서를 이용하였습니다.

라즈베리파이에서 아두이노로 Serial.write() 메소드를 사용한 시리얼 통신을 통해 온도 측정을 해달라는 신호를 주면, 아두이노에서 연결된 적외선 온도 측정 센서에서 온도 측정을 한 후 Sreial.println()으로 출력, 라즈베리파이에서는 이를 Serial.readline()을 통해 전달받은 후 전달받은 온도 중 가장 높은값을 리턴해주는 식으로 구현하였습니다.

평균 온도는 가끔 값이 튀는 경우가 있어, 최고 온도 기준으로 하도록 코드를 작성하였습니다.

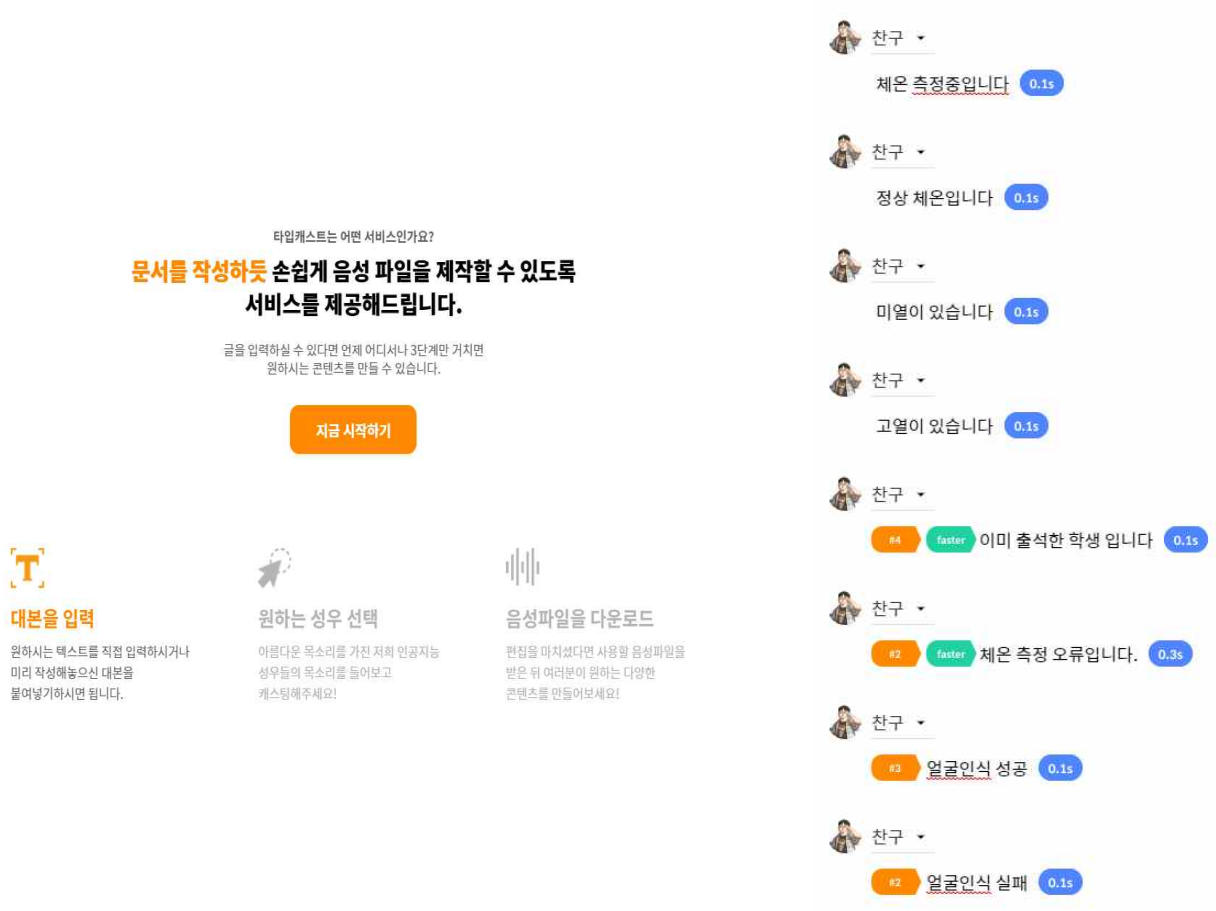
```
void loop() {
  // put your main code here, to run repeatedly:

  while(Serial.available()){   //시리얼에 읽을 값이 있으면
    //Serial.println(Serial.available());
    digitalWrite(4, HIGH);    //초록불 켜기
    delay(1000);
    digitalWrite(4, LOW);    //초록불 켜기
    income += (char)Serial.read(); //income안에 해당 내용 저장
    if(income == 's'){
      break;
    }
    //Serial.println(income);//시리얼에 해당 내용 전송
  }
  if(income != 0){ //income에 내용이 있으면

    if(income == 's'){
      //Serial.println("1");
      while (count++ < 5){
        if(Timer1_Flag){                // 50ms마다 반복 실행(Timer 1 Flag check)
          Timer1_Flag = 0;                          // Reset Flag
          iOBJECT = SPI_COMMAND(OBJECT,DELAY_10us);     // 대상 온도 Read
          delayMicroseconds(10);                    // 10us : 이 라인을 지우지
          iSENSOR = SPI_COMMAND(SENSOR,DELAY_10us);     // 센서 온도 Read
          delayMicroseconds(10);                    // 10us : 이 라인을 지우지
          cEratio = SPI_COMMAND(ERATIO,DELAY_35us);     // 방사율 READ, 반드시 35us

          //Serial.print("Object Temp : ");            // 하이퍼터미널 출력
          Serial.println((float(iOBJECT)/10 + 6.5),1);
          //Serial.print("     Sensor Temp : ");
          //Serial.print(float(iSENSOR)/10,1);
          //Serial.print("    Eratio : ");
          //Serial.println(float(cEratio)/100,2);
          delay(500);
          SPI_COMMAND(LASER,DELAY_10us);
        }
        /*if(Laser_Flag == 1){
          Laser_Flag = 0;
          SPI_COMMAND(LASER,DELAY_10us);
          while(digitalRead(2)==LOW);
          attachInterrupt(digitalPinToInterrupt(2), LASER_ISR, FALLING);
        }*/
      }
      LASER_ISR();
      count = 0;
    }
    income = '\0'; //전송한 income내용 초기화
  }
}
```

[그림] Arduino 시리얼 통신

```
@classmethod
def start(cls):
    cls.data = []
    print("전송된 byte 길이 =", cls.ser.write("s".encode()))
    while len(cls.data) < 5:
        try:

            res = cls.ser.readline().decode()
            # decode byte data and slice \n
            if res:
                cls.data.append(res.strip())
            print(res)

        except serial.SerialException as se:
            print(se)
            print("SerialException")

        except ValueError as ve:
            print(ve)
    print("받은 데이터", cls.data)
    return max(cls.data)
```

[그림] Python 시리얼 통신

5.5 음성 출력 기능



[그림] 타입캐스트 온라인 TTS 사이트

```
class playSound:
    @classmethod
    def play(cls, path):
        wave_obj = sa.WaveObject.from_wave_file(path)
        play_obj = wave_obj.play()
        play_obj.wait_done()
```

[그림] simpleAudio 사용 코드

음성 출력기능은 타입캐스트를 통한 TTS 사이트를 통해 원하는 음성을 출력한 후 Simple Audio 라이브러리와 블루투스 스피커를 연결하여 구현하였습니다.

5.6 Google Firebase

5.6.1 Real-Time Database

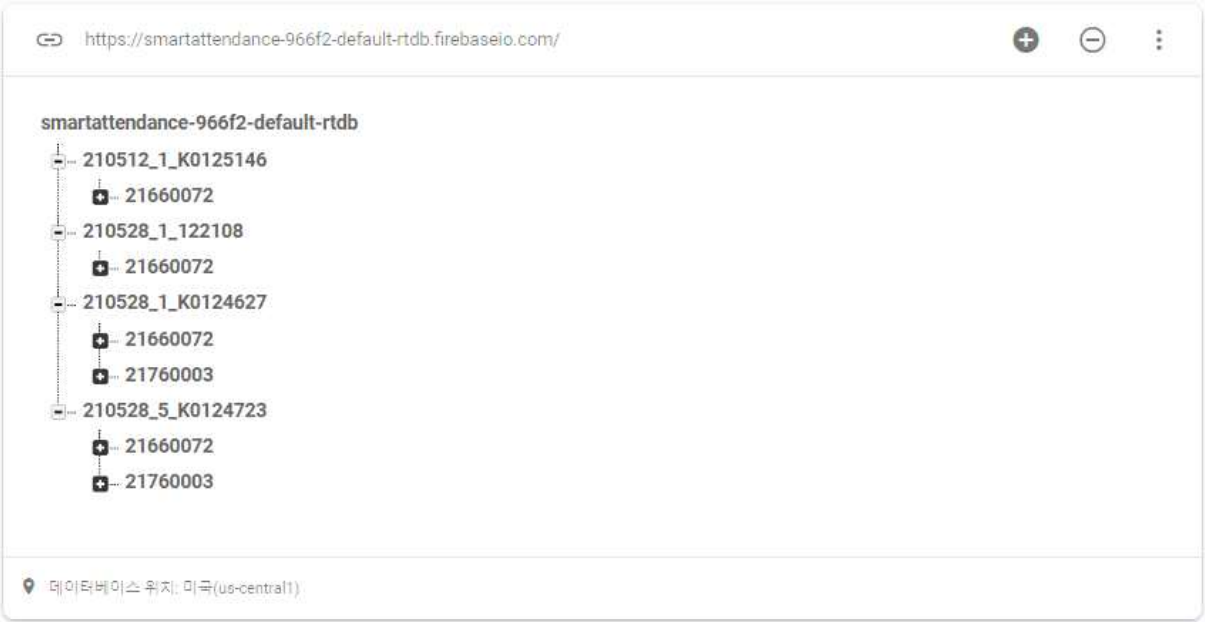
Arduino나 Raspberry Pi에서 측정한 센서의 값을 Android 환경에서도 사용할 수 있게 하려면, 센서의 값을 측정할 때, 데이터베이스에 저장 후, Android에서 해당 데이터베이스에 접근하여 필요한 센서 값을 가져온 뒤, 그 값으로 기능을 수행하는 것이 가장 간단한 방법입니다.

Google Firebase는 데이터베이스 기능을 지원하는 Cloud Firestore와 Real-time Database를 제공하며, 「스마트 출석부」 에서는 Real-time Database 기능만을 사용하였습니다.

Firebase에 Android 어플리케이션을 등록하는 과정에서 Android의 manifest에 Firebase SDK를 등록하였다면 Real-time Database를 쉽게 사용할 수 있습니다.

Firebase의 Real-time Database는 JSON 형식으로 저장되어 관리되기 때문에 DataSnapshot 형태로 가져와 데이터를 파싱하여 사용합니다.

Real-time Database에 결괏값들을 저장할 스키마를 교수자의 클라이언트에서 소켓 통신을 통한 데이터로 만들어두고 Arduino와 Raspberry Pi에서 측정한 결괏값들을 저장합니다.



[그림1] Real-Time Database

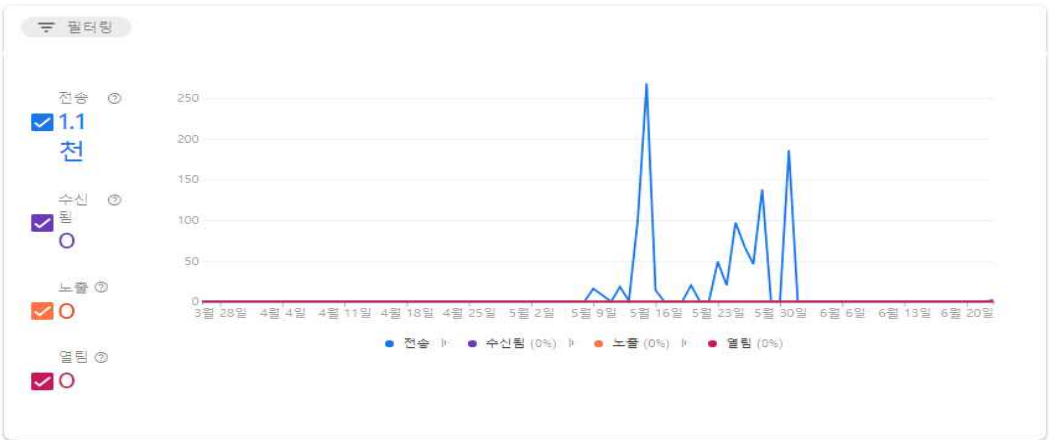
5.6.2 Firebase Cloud Messaging

Cloud Messaging은 무료로 메시지를 안정적으로 전송할 수 있는 교차 플랫폼 메시징 솔루션입니다.

「스마트 출석부」에 Cloud Messaging을 사용하여 정상 범위보다 높은 체온을 나타내는 학생이 식별될 경우 교수자가 어플리케이션 직접 실행시켜 확인하지 않아도 Raspberry PI에서 Cloud Messaging을 사용해 교수자의 클라이언트에 메시지를 보내주고, 안드로이드 어플리케이션의 리스너 메서드에서 즉시 해당 메시지를 수신해 교수자의 클라이언트로 Push 알림을 보내 교수자가 코로나 의심 환자에 즉각 대응 할 수 있습니다.

```
@classmethod
def cloudMessaging(cls, student_id, data, result):
    # 파이어베이스 콘솔에서 얻어 온 서버 키를 넣어 줌
    push_service = FCMNotification(cls.CLOUDMESSAGING_APIKEY)

    def sendMessage(body, title):
        # 메시지 (data 타입)
        data_message = {
            "body": body,
            "title": title
        }
        # 토큰값을 이용해 등록한 사용자에게 푸시알림을 전송함
        for token in cls.token_list:
            token = token.split("\n")[0]
            result = push_service.notify_single_device(registration_id=token, message_title=title,
                                                       message_body=body)

        # 전송 결과 출력
        print("클라우드 메시징 전송 결과 : ", result)

    if result == 1:
        sendMessage(data[student_id]["id"], "출석 완료")
    elif result == 2:
        sendMessage(data[student_id]["id"], "[주의] 미열")
    elif result == 3:
        sendMessage(data[student_id]["id"], "[주의] 고열")
    elif result == 0:
        sendMessage(data[student_id]["id"], "출석 오류")
```

[그림1] Cloud Messaging 전송



[그림2] Cloud Messaging

Ⅲ. 결 론

「스마트 출석부」 프로젝트를 통해 코로나19 펜데믹 시대에 접어들면서 불편했던 점들과 불필요한 인력 낭비를 최소화하기 위해 어떤 기능을 구현하면 좋을지 수많은 회의를 진행하였고 그 결과로 코로나 전염 예방에 가장 효과적인 출석, 발열 검사에서 자동화/비대면 시스템 구성하여 코로나 전염 가능성을 최소화함과 동시에 기존 시스템의 불편했던 사항들을 개선하고자 하였습니다.

출석, 발열 검사 결과 또한 교수자의 클라이언트로 통보해주어 코로나 의심 환자에 대한 불안감을 최소화하여 수업을 진행할 수 있도록 하였습니다.

또한, 「스마트 출석부」를 사용하기 위해서는 검사 대상 인물에 대한 사진 1장만 있으면 출석과 발열 검사가 가능하고 결과 또한 Real_Time Database에 저장되기 때문에 어디서든 간편하게 구현하여 사용할 수 있습니다.

Github : https://github.com/Leekm0912/3-1_SmartAttendance

[그림2] 스마트 출석부 시연