

IoT 프로젝트 보고서



과목명	IoT
교수명	최차봉 교수님
학과	소프트웨어학과
팀	7조
팀원	김지원(2018125019) 이수빈(2018125045) 백경환(2016125027) 한기찬(2016125079) 이재민(2016125057)
제출일	2020-12-24

목차

1. 프로젝트 선정 배경 및 개요

- 1) 배경
- 2) 프로젝트 목적
- 3) 프로젝트 요구사항
- 4) 기대 효과
- 5) 예상 시나리오

2. 프로젝트 구현 방법

- 1) 전체 프레임워크
- 2) 하드웨어
- 3) AWS
- 4) 모바일 어플리케이션

3. 개인 역할 및 기여

- 1) 역할 분담
- 2) 일정 및 추가 구매 내역

4. 결론 및 고찰

5. 부록

- 1) 실제 구현 모습

1. 프로젝트 선정 배경 및 개요

가) 배경

전염병의 대유행으로 인해 '실내 환기'는 질병 확산 방지를 위해 어느 때보다 중요하다. 가정의 환기는 주로 창문을 통해 이루어지는데, 열린 창문을 통해 실내와 실외의 온도, 습도, 공기질을 공유한다.

바쁜 현대인 혹은 직장 생활을 하는 1인가구는 집의 환기에 힘쓸 여력이 부족하다. 그렇다고 해서, 창문을 열어 둔 채로 나간다면, 과도한 환기로 인해 여름엔 덥고 겨울엔 춥고 비가 올 때는 집안이 축축히 젓고, 미세먼지가 심한 날엔 집안의 공기가 나빠진다는 문제를 겪게 된다. 또한 창문을 열어 둔 채로 나간다면, 그 열린 창문을 통해 주거침입 등의 범죄에 노출될 확률이 높다.

창문이 자동으로 환기해야 할 때 열리고, 닫혀야 할 때 닫힌다면, 직장에서 마음 놓고 일한 뒤 쾌적한 온도와 습도, 공기질을 유지하고 있는 포근한 집으로 돌아와 편히 쉴 수 있을 것이다.

나) 프로젝트 목적

스스로 개폐를 통한 환기 여부를 결정하는 IoT 제품을 제작하여, 환기에 관한 사용자의 고민을 덜어주고자 한다.

다) 프로젝트 요구사항

- i. 창문을 원격으로 열고 닫는 장치를 만든다.
- ii. 실내의 온습도와 실외의 온습도, 강수량 및 미세먼지를 측정해 환기 여부를 결정하도록 한다.
- iii. 스마트폰 어플을 통해 원격 제어가 가능하게 한다.

라) 기대 효과

스마트한 환기 시스템을 통해 실내 온도와 습도, 공기질을 쾌적한 수준으로 조절할 수 있다. 공기 중으로 전파되는 전염병의 확산을 막는 효과가 있다. 또한 창문의 개폐 여부를 어플로 확인할 수 있기 때문에 창문 방범 효과가 있다.

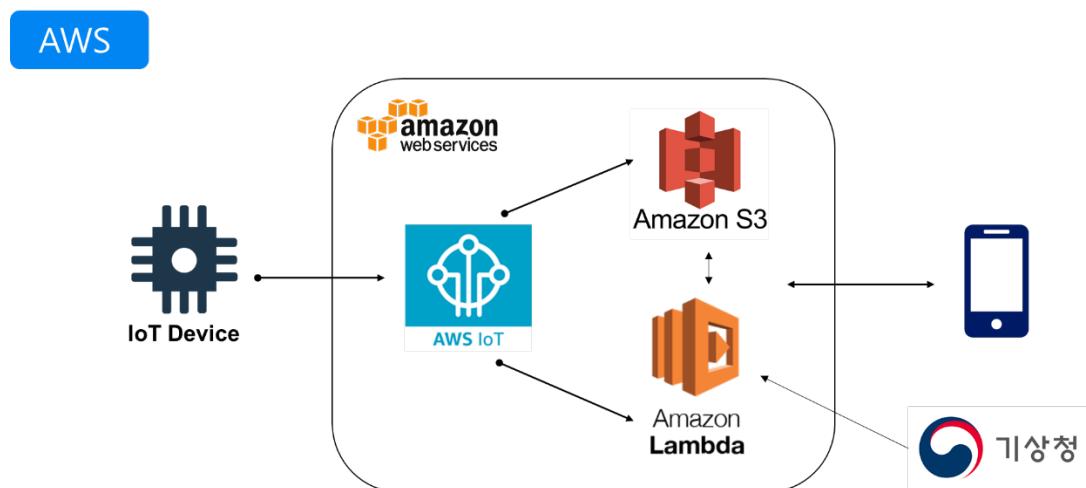
마) 예상 시나리오

- i. 사용자 어플 첫 실행 시 자신의 주거지 위치정보를 입력한다. 이 위치는 외부 온도, 습도, 강수량, 미세먼지 농도 등의 정보를 가져올 때 쓰인다. 사물은 실내의 온도와 습도를 측정하여, AWS Lambda를 통해 AWS IoT로 보낸다. Lambda는 웹사이트에서 설정한 위치의 날씨 정보를 크롤링하여 사물에서 받은 실내 정보와 비교한다. 이를 통해 환기(창문 개폐) 여부를 결정하여, 환기가 필요할 경우 A의 핸드폰으로 알려준다. 알림을 받은 사용자는 ON/OFF 버튼을 이용해 원격으로 창문을 개폐할 수 있다.
- ii. 기상청 자료에서 적정온도(22-26도)라는 정보를 인용하여 실내온도가 적정온도를 벗어난 상황: 외부온도가 정상이고, 실내온도가 비정상이라면, 환기를 위해 창문을 여는 게 좋다는 알림 메세지를 사용자의 어플로 전송한다. 사용자가 어플에서 열기 버튼을 누르면, 장치는 자동으로 창문을 열어준다. 만약 실내온도가 정상이라면 환기를 추천하지 않는다.
- iii. 기상청의 자료에서 적정습도는 40-70도라는 정보를 인용하여. 실내습도가 적정습도가 아니고, 외부 습도가 적정습도라면, 환기를 추천하는 알람을 사용자에게 보낸다.
- iv. 창문이 열려 있을 때, 열기 버튼을 누른다면 수행하지 않음
- v. 창문이 닫혔 있을 때, 닫기 버튼을 누른다면 수행하지 않음

- vi. 사용자는 어플에서 창문을 열고 나왔는지, 닫고 나왔는지 확인할 수 있다. 그러므로 창문을 열고 나와도 어플 확인 후 원격으로 창문을 닫을 수 있다.

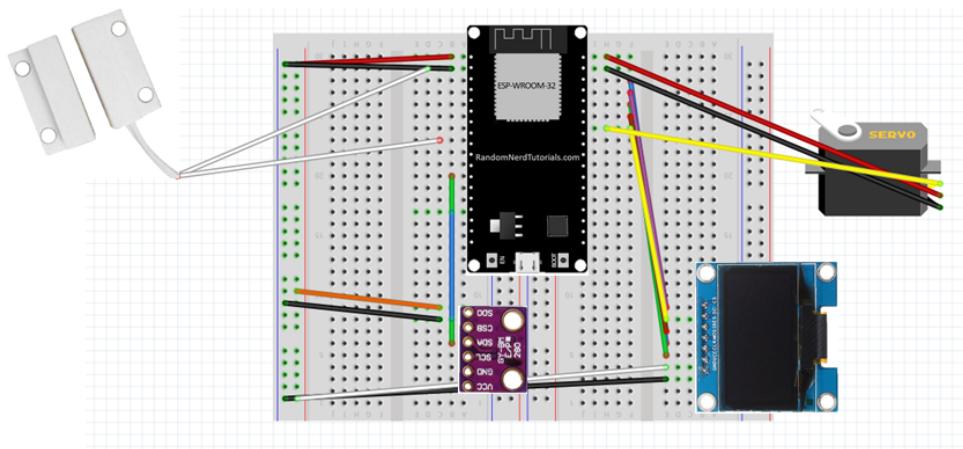
2. 프로젝트 구현 방법

가) 전체 프레임워크



- 개발환경: Arduino, AWS, Android Studio
- AWS와 아두이노를 통해서 자동으로 창문 개폐를 조절하는 시스템
- 아두이노에서 센서값을 AWS로 보냄
- AWS에서 개폐를 지시하면 아두이노에서 명령을 받아 창문을 개폐
- 아두이노에서 수집한 센서값을 S3에 저장하고 AWS Lambda로 제어

나) 하드웨어



- 온습도 센서: 실내의 온도 및 습도를 주기마다 측정하는 센서
- 마그네틱 센서: 창문의 개폐 여부(OPEN/CLOSE)를 구분하는 센서
- OLED Display: 온습도 센서, 마그네틱 센서로부터 정보를 얻어와 시각화하여 보여주는 장치
- 서보모터: 창문과 연결하여 서보모터가 회전하여 OPEN/CLOSE를하게 해주는 장치

다) AWS

(1) IoTcore

규칙
sensor_testing
활성

작업 ▾

개요	설명	편집
Tags	esp32 send data to iot core & store in s3 -> trigger lambda \$ send data and alarm to android	
규칙 쿼리 설명문	이 규칙을 사용하여 처리하고자 하는 메시지의 소스입니다.	편집
<pre>SELECT * FROM 'esp32/bme280'</pre>		
SQL 버전 사용 2016-03-23		
작업		
작업은 규칙이 트리거되면 이루어지는 것입니다. 자세히 알아보기		
Amazon S3 빅켓에 메시지 저장 sensor-esp32		제거 편집 ▾

Esp32에서 bme280으로 현재의 온도, 습도를 측정하여 topic esp32/bme280으로 publish 하면 미리 생성된 규칙에 의거하여 sensor-esp32 S3에 측정값이 저장된다.

클라이언트의 문 개폐여부를 람다함수에서 mqtt protocol로 전송하면 위의 규칙이 실행되어 s3에 저장하고 재게시를 하여 esp32로 전송한다.

(2) lambda

<cawling-region>

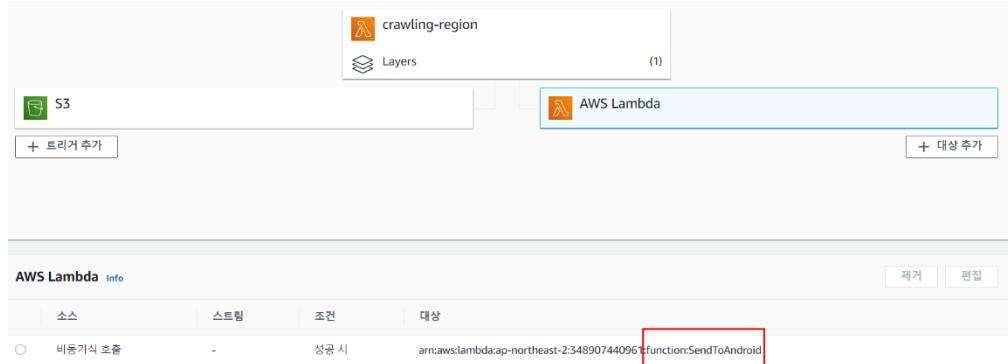
```

cawling-region
File Edit Find View Go Tools Window Test Deploy
Environment crawling-region - / lambda_function.x
File Edit Find View Go Tools Window Test Deploy
lambda_function x +
9 # import matplotlib.pyplot as plt
10
11 source = requests.get('https://www.weather.go.kr/weather/observation/currentweather.jsp')
12 soup = BeautifulSoup(source.content,'html.parser')
13
14 table = soup.find('table',{'class':'table_develop3'})
15 data = []
16
17 ## s3 버킷 저장
18 BUCKET_NAME = 'all-region-data'
19 KEY = 'all_location_data.txt'
20 s3_client = boto3.client('s3')
21
22 s3 = boto3.resource('s3')
23
24 def lambda_handler(event, context):
25     s3.Object(BUCKET_NAME, KEY).delete()
26     print("지역 온도")
27     for tr in table.findAll('tr'):
28         tds = list(tr.findAll('td'))
29         for td in tds:
30             if td.find('a'):
31                 point = td.find('a').text
32                 temp = tds[5].text
33                 humidity = tds[10].text
34
35                 print("{0:<7} {1:<7} {2:<7}".format(point,temp,humidity))
36                 data.append([point,temp,humidity])
37
38     ## 버킷에 저장하는 코드
39     s3_client.put_object(Body=json.dumps(data, ensure_ascii=False), Bucket=BUCKET_NAME, Key=KEY)

```

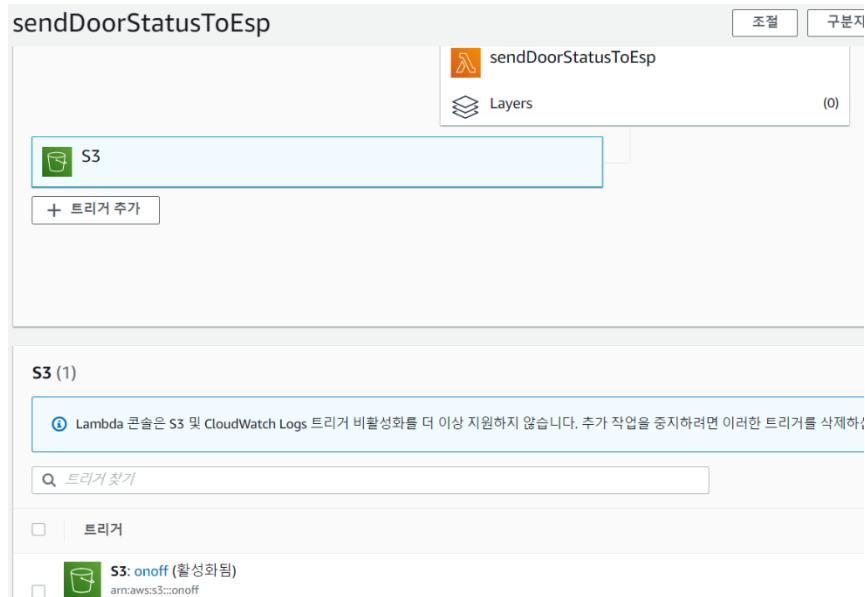
Sensor-esp32 S3가 trigger로 발동하여 crawling-region 람다 함수로 크롤링하여 전국 지역의 온도와 습도에 대한 정보를 가져온다. 이 람다 함수의 대상을 또 다른 람다 함수(SendToAndroid)로 작동한다.

<SendToAndroid>



크롤링한 데이터 중 사용자지정 위치에 해당하는 온도와 습도를 비교하여 기준을 벗어나면 알람을 보내는 기능을 구현하였다. 알림을 보내는 프레임워크는 파이어베이스의을 사용하였고, 적정온도와 습도를 벗어나는 상황이 오면 사용자가 원치 않는 경우도 있기 때문에 30분 간격으로 알람을 가게 설정하였다. 알람을 보내는 시간을 측정하여 s3에 저장을 한 다음, 반복적으로 해당 s3에서 시간 데이터를 읽어와 30분이 소요됐는지 확인하고 알림을 보낸다.

<SendDoorStatusToEsp>



어플에서 사용자가 문 개폐 여부를 onoff S3에 전송하면 이 람다함수의 트리거로 작동하여 mqtt protocol을 기반으로 하여 topic을 publish 한다.

(3) S3

<sensor-esp32>

액체 (4)						
액체는 Amazon S3에 저장되어 있는 기본 엔터티입니다. 다른 사용자가 액체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다. 자세히 알아보기						
	삭제	작업 ▾	클더 만들기	업로드		
<input type="text"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>	<input type="button"/>		
이름	유형	마지막 수정	크기	스토리지 클래스		
alarm_current_time.txt	txt	2020. 12. 23. am 5:35:20 AM KST	5.0B	Standard		
location_data.txt	txt	2020. 12. 23. am 1:46:42 AM KST	7.0B	Standard		
text.txt	txt	2020. 12. 23. am 2:29:59 AM KST	23.0B	Standard		
text2.txt	txt	2020. 12. 3. pm 3:14:02 PM KST	32.0B	Standard		

Alarm_current_time : 가장 최근에 알림을 보낸 시간을 저장, 30분 간격으로 전송하기 위해 기록해둠

Location_data : 사용자 지정 위치 저장

Text : bme280에서 측정한 온도, 습도 측정값

<onoff>

The screenshot shows the AWS S3 console interface. At the top, there's a search bar with the placeholder '접두사로 객체 찾기'. Below it is a table with columns: 이름 (Name), 유형 (Type), 마지막 수정 (Last modified), 크기 (Size), and 스토리지 클래스 (Storage class). A single row is visible for 'door.txt', which is a txt file last modified on 2020.12.23 at 12:35:51 AM KST, with a size of 31.0B and a Standard storage class.

Door : 사용자가 원하는 문의 개폐 여부

<all-region-data>

The screenshot shows the AWS S3 console interface again. It displays two objects in a bucket: 'all_location_data.txt' and 'location_data.txt'. Both are txt files. 'all_location_data.txt' was last modified on 2020.12.23 at 5:35:22 AM KST and is 19.7KB in size. 'location_data.txt' was last modified on 2020.12.22 at 10:55:03 PM KST and is 4.9KB in size. Both are stored in the Standard storage class.

All_location_data : 기상청에서 크롤링한 전국의 온도 습도

라) 어플리케이션

<S3-location-upload>

```
// s3에 지역 업로드
public void uploadWithTransferUtility(String dummy, String filename, String bucketname) throws IOException {
    FileOutputStream fileOutputStream = new FileOutputStream( name: getFilesDir() + filename, append: false);
    byte []tempByte = dummy.getBytes();
    int length = dummy.length();
    fileOutputStream.write(length);
    fileOutputStream.write(tempByte);
    fileOutputStream.write();
    fileOutputStream.close();
    File file2 = new File( pathname: getFilesDir() +filename );
    CognitoCachingCredentialsProvider credentialsProvider = new CognitoCachingCredentialsProvider(
        getApplicationContext(),
        identityPoolId: "ap-northeast-2:f80fe4d0-e4d3-4e09-be94-f5d452722201", // 지역 층명 풀 ID
        Regions.AP_NORTHEAST_2 // 리전
    );
    TransferNetworkLossHandler.getInstance(getApplicationContext());
    TransferUtility transferUtility =
        TransferUtility.builder()
            .context(getApplicationContext())
            .defaultBucket(bucketname)
            .s3Client(new AmazonS3Client(credentialsProvider, Region.getRegion(Regions.AP_NORTHEAST_2)))
            .build();
    TransferObserver uploadObserver =
        transferUtility.upload(
            bucketname,
            filename,
            file2);
    uploadObserver.setTransferListener(new TransferListener() {
        @Override
        public void onStateChanged(int id, TransferState state) {
            if(state == TransferState.COMPLETED){
                System.out.println("upload 완료");
            }
        }
    });
}
```

클라이언트가 선택한 지역을 s3->sensor-esp32(bucket)->lacation_data.txt 파일에 저장

<S3-esp32_control-upload>

```

public void uploadWithTransferUtility(String str) throws IOException {
    File file = new File(getfilesDir(), child: "text.txt");
    BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(file)));
    String line;
    String dummy="";
    ArrayList<String> tv = new ArrayList<>();

    while ((line=br.readLine())!=null){
        tv.add(line);
    }

    for(int i=0;i<tv.size();i++){
        System.out.println(tv.get(i));
    }
    String[] str2=tv.get(0).split( regex ",\n");
    System.out.println(str2[2]);
    if(str.equals("열림") &&str2[2].equals("0")) dummy+="{ \"state\" : {\"order\" : \"OPEN\"}}";
    else if(str.equals("닫힘")&&str2[2].equals("1")) dummy +="{ \"state\" : {\"order\" : \"CLOSE\"}}";
    else return;
    br.close();
}

```

Esp32 제어 시 문이 닫혀 있을 때 닫힘을 누르면 아무 동작이 안 하도록 현재 창문의 개폐여부상태를 가져온 뒤 버튼 값과 비교 후 다르면 업로드하여 esp32 창문 여닫이를 제어, 업로드 방법은 위의 location-update와 같음

<S3-download>

```

CognitoCachingCredentialsProvider credentialsProvider = new CognitoCachingCredentialsProvider(
    getApplicationContext(),
    identityPoolId: "ap-northeast-2:f80fe4d0-e4d3-4e09-be94-f5d452722201", // 자격 증명 폴 ID
    Regions.AP_NORTHEAST_2 // 리전
);
TransferNetworkLossHandler.getInstance(getApplicationContext());
TransferUtility transferUtility =
    TransferUtility.builder()
        .context(getApplicationContext())
        .defaultBucket("sensor-esp32")
        .s3Client(new AmazonS3Client(credentialsProvider, Region.getRegion(Regions.AP_NORTHEAST_2)))
        .build();
TransferUtility transferUtility2=TransferUtility.builder()
    .context(getApplicationContext())
    .defaultBucket("all-region-data")
    .s3Client(new AmazonS3Client(credentialsProvider,Region.getRegion(Regions.AP_NORTHEAST_2)))
    .build();
File file = new File(getFilesDir(), child: "text.txt");
File file2 = new File(getFilesDir(), child: "location_data.txt");
File file3 = new File(getFilesDir(), child: "all_location_data.txt");
TransferObserver downloadObserver = transferUtility.download( key: "text.txt", file);
TransferObserver downloadObserver2 = transferUtility.download( key: "location_data.txt",file2);
TransferObserver downloadObserver3 = transferUtility2.download( key: "all_location_data.txt",file3);

```

Layout에 띄워야 할 esp32 상태 값, client가 선택한 지역, 크롤링한 기상청 데이터를 가져오기 위해 s3와 연결

```

downloadObserver.setTransferListener(new TransferListener() {
    @Override
    public void onStateChanged(int id, TransferState state) {
        try {
            BufferedReader buf = new BufferedReader(new FileReader(file));
            String line = null;
            ArrayList<String> tv = new ArrayList<>();
            while ((line = buf.readLine()) != null) {
                tv.add(line);
            }
            for (int i = 0; i < tv.size(); i++) {
                System.out.println(tv.get(i));
            }
            String[] str = tv.get(0).split( regex: ", " );
            str[0] = Double.toString( d: Math.round(Double.parseDouble(str[0]) * 10) / 10.0 );
            str[1] = Double.toString( d: Math.round(Double.parseDouble(str[1]) * 10) / 10.0 );
            window_state= str[2];
            home_temp = findViewById(R.id.home_temp);
            home_humid = findViewById(R.id.home_humid);
            window = findViewById(R.id.home_door);
            home_temp.setText("실내온도 : " + str[0] + "도");
            home_humid.setText("실내습도 : " + str[1] + "%");
            if (str[2].equals("1")) window.setText("현재 문이 열려있습니다.");
            else window.setText("현재 문이 닫혀있습니다.");
            buf.close();
        }catch (FileNotFoundException e){
            e.printStackTrace();
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
})

```

Esp32 상태값을 가져와서 실내온도, 실내습도, 현재 창문의 개폐상태를 가져와서 적용

```

downloadObserver2.setTransferListener(new TransferListener() {
    @Override
    public void onStateChanged(int id, TransferState state) {
        try {
            BufferedReader buf = new BufferedReader(new FileReader(file2));
            String line = null;
            ArrayList<String> tv = new ArrayList<>();
            while ((line = buf.readLine()) != null) {
                tv.add(line);
            }
            for (int i = 0; i < tv.size(); i++) {
                System.out.println(tv.get(i));
            }
            region=tv.get(0);
            region=region.substring(1);
            buf.close();
            home_address.setText(region);
            System.out.println(region);
        }catch (FileNotFoundException e){
            e.printStackTrace();
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
})

```

클라이언트가 기존에 설정했던 지역을 가져와 layout에 적용

```

public void onStateChanged(int id, TransferState state) {
    try {
        BufferedReader buf = new BufferedReader(new FileReader(file));
        String line = null;
        ArrayList<String> tv = new ArrayList<>();
        while ((line = buf.readLine()) != null) {
            tv.add(line);
        }
        for (int i = 0; i < tv.size(); i++) {
            System.out.println(tv.get(i));
        }
        int i=tv.get(0).indexOf(region);
        int j=tv.get(0).indexOf("["),i;
        System.out.println(region);
        String region_str="";
        for(int k=i+1;k<j;k++){
            region_str +=tv.get(0).charAt(k);
        }
        System.out.println(region_str);
        String[] str = region_str.split( regex: ", " );
        System.out.println(str.length);
        System.out.println(str[0]);
        for (int q=0;q<str.length;q++){
            str[q]=str[q].substring(1);
            str[q]=str[q].substring(0,str[q].length()-1);
            System.out.println(str[q]);
        }
        out_temp =findViewById(R.id.out_temp);
        out_humid =findViewById(R.id.out_humid);
        out_temp.setText("외부온도 : " + str[1]+"도");
        out_humid.setText("외부습도 : "+str[2]+"%");
        buf.close();
    }catch (FileNotFoundException e){
        e.printStackTrace();
    }
}

```

lambda에서 기상청을 크롤링하여 저장한 text file을 가져와서 기존 설정했던 지역에 해당하는 온도, 습도를 가져와서 layout에 적용

3. 개인 역할 및 기여

가)역할 분담

- i. 김지원: 하드웨어 조립 및 AWS 연결, 서보모터 회전각 조절, 어플 ui 구현
- ii. 백경환: 하드웨어 조립 및 AWS 연결, 크롤링, 디스플레이 조정
- iii. 이수빈: 하드웨어 조립 및 AWS 연결, 마그네틱 센서값 조절, ppt
- iv. 이재민: 람다 함수 작성, 크롤링, aws 클라우드 서비스 연결, s3 관리
- v. 한기찬: 어플 기능 구현 및 람다 함수 작성

나)일정 및 추가 구매 내역

i. 일정

ii. 추가 구매 내역

마그네틱 센서 2쌍 – 800원

4. 결론 및 고찰

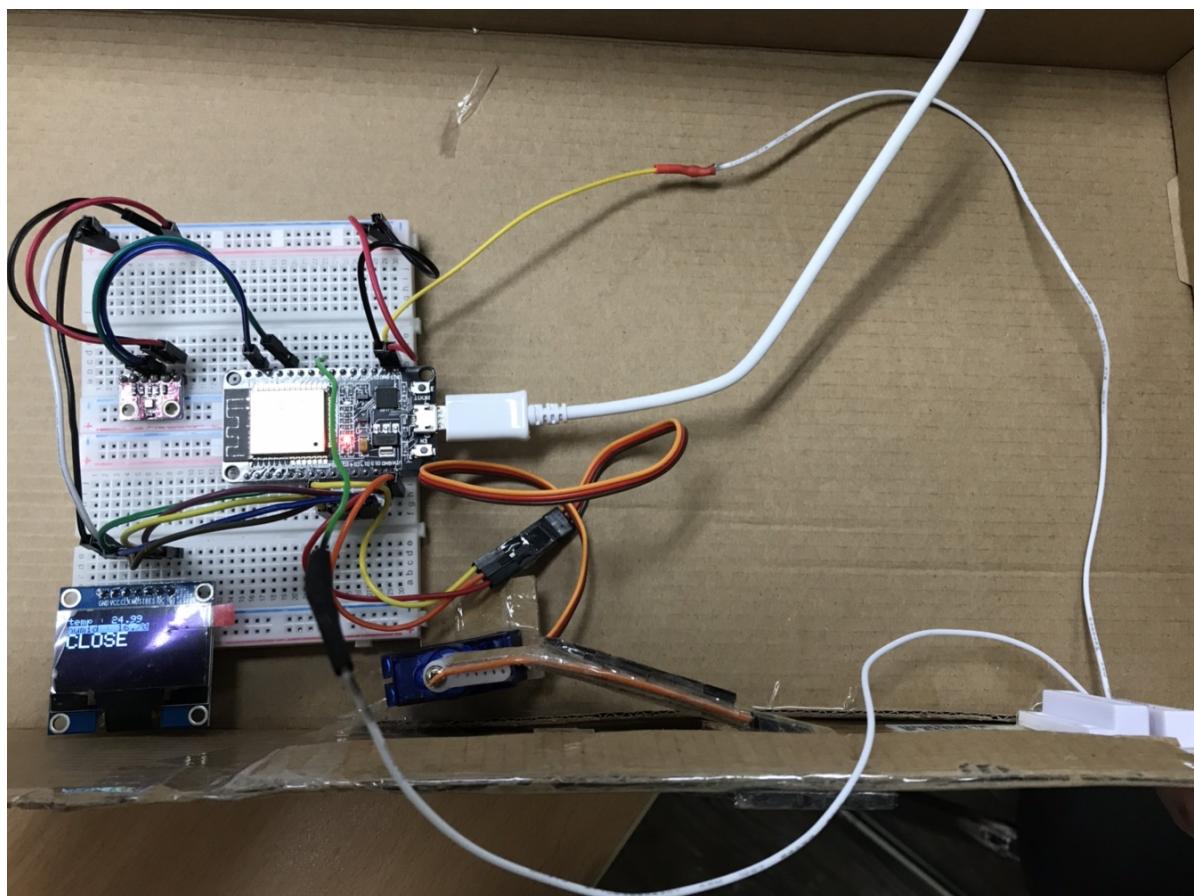
가) 겪은 문제점과 해결방안

- i. esp32의 서보모터는 아두이노의 서보모터와는 다르게 작동한다. 아두이노의 서보모터 패키지는 모터가 360도 회전하는 것을 지원하지만, esp32의 서보모터는 180도까지만 지원한다. 이를 해결해서 창문을 열고 닫기 위해, 창문이 안전하게 열릴 만큼 파라미터를 조절하였다. 만약 실제 창문에 적용하게 된다면, 그만큼 힘이 강한 모터, 팔이 긴 모터를 사용해야 할 것이다.
 - ii. 기본적인 통합개발환경이나 로컬과 다르게 aws 클라우드에서 파이썬 라이브러리를 활용하는 데 있어서 어려움이 있었다 파이썬 패키지를 설치하는 관리시스템 pip없이 어떻게 구현을 해야할지 하다가 구글링을 통해 aws lambda에서는 layer에 패키지를 추가해놓

고 필요할 때 마다 사용할 수 있게 해놨다는 것을 알았다. layer에 추가를 하려다가 용량 제한이 있다는 것을 알고 패키지 zip파일을 선별하여 layer에 추가하였다. 추후에 aws lambda로 개발을 할 때는 기능과 사용목적을 기준으로 패키지를 구분하여 사전에 layer를 만들고 버전을 적용하면 좀 더 효율적으로 개발을 할 수 있을 것 같다.

5. 부록

가) 실제 구현 모습





: 열림 상태



: 닫힘 상태