2025-1 IoT네트워크 Project 보고서

Weather Monitoring Service

2025. 6. 13

스마트 IoT
20235177
박혜진

한림대학교
HALLYM UNIVERSITY

목 차

제 1 장 서 론 ····· 4
제 1절 필요성 및 목적 ····· 4
1. 프로젝트 필요성 ····· 4
2. 프로젝트 목적 ····· 4
3. 프로젝트 구조 ····· 4

제 2 장 본 론 ····· 5
제 1절 리소스 ····· 5
1. 온도 ····· 5
2. 습도 ····· 5
3. 강수량 ····· 5
4. 강수형태 ····· 5
제 2절 MQTT 통신 ····· 6
1. MQTT Broker ····· 6
2. MQTT Publisher ····· 6
3. MQTT Subscriber ····· 9
제 3절 웹 서버 및 웹 페이지 구현 ····· 11
1. DB 저장 ····· 11
2. 소켓 통신 ····· 12
3. 웹 페이지 화면 구성 ····· 13
4. 웹 페이지 소스코드 ····· 13
제 4절 실행 결과 ····· 14
1. 실행 결과 ····· 14

제 3 장 결 론 ····· 16

참고자료 ····· 17

그 림 목 차

[그림 2-1] 리소스 ····· 5
[그림 2-2] MQTT Broker ····· 6
[그림 2-3] 리소스 정의 ····· 7
[그림 2-4] connectBroker() 메서드 ····· 7
[그림 2-5] get_weather_data() 메서드 1 ····· 8
[그림 2-6] 메서드 2 ····· 8
[그림 2-7] 데이터 JSON 변환 ····· 8
[그림 2-8] 메시지 발행 ····· 8
[그림 2-9] 반복 실행 및 예외 처리 ····· 9
[그림 2-10] messageArrived() 메서드 ····· 9
[그림 2-11] MongoDB 연결 ····· 10
[그림 2-12] 메시지 수신, 처리 ····· 10
[그림 2-13] 메시지 수신, 처리 ····· 10
[그림 2-14] 소켓 통신 데이터 전송 ····· 11
[그림 2-15] 소켓 통신 데이터 전송 ····· 11
[그림 2-16] 메시지 수신 처리 ····· 12
[그림 2-17] 메시지 수신 처리 ····· 12
[그림 2-18] 소켓 통신 처리 ····· 12
[그림 2-19] 소켓 통신 처리 ····· 12
[그림 2-20] 웹 페이지 화면 ····· 13
[그림 2-21] html 소스코드 1 ····· 14
[그림 2-22] html 소스코드 2 ····· 14
[그림 2-23] html 소스코드 3 ····· 14
[그림 2-23] html 소스코드 4 ····· 14
[그림 2-25] 웹 페이지 화면 1 ····· 15
[그림 2-26] 웹 페이지 화면 2 ····· 15

제 1 장 서 론

제 1 절 필요성 및 목적

1. 프로젝트의 필요성

기존 기상예보 서비스는 주로 정부나 대기업에서 제공하며 정보는 방대하지만, 실시간 반영이 제한적이고 맞춤형 제공이 부족해 일반 사용자가 신속하고 정확한 정보를 얻기 어렵다. 또한 복잡한 데이터 형식으로 비전문가가 활용하기 어려운 점도 있다.

이에 본 프로젝트는 공개 API를 통해 실시간 기상 데이터를 자동 수집하고 데이터베이스에 저장하여, 사용자가 필요할 때 쉽고 빠르게 맞춤형 정보를 조회할 수 있는 시스템을 개발하고자 한다.

2. 프로젝트의 목적

이 프로젝트의 목적은 사용자가 실시간 기상 정보를 쉽고 빠르게 제공받는 것이다. 기존의 기상예보 시스템들이 제공하는 정보는 많지만, 일반 사용자가 실시간으로 필요한 정보를 직관적으로 확인하기 어렵다는 문제를 해결하고자 한다. 누구나 간편하게 이용할 수 있는 맞춤형 기상예보 시스템을 개발해, 사용자들이 정확한 날씨 정보를 바탕으로 일상생활이나 계획을 세우는 데 도움을 주고자 한다.

3. 프로젝트의 구조

이 프로젝트는 크게 데이터 수집, 데이터 처리, 그리고 사용자에게 정보 제공하는 세 부분으로 구성되어 있다. 먼저, 기상청에서 제공하는 다양한 기상 데이터 (초단기예보)를 API를 통해 실시간으로 수집한다. 수집된 데이터는 서버에서 정리되고, 필요에 따라 가공되어 데이터베이스(MongoDB)에 저장된다. 마지막으로, 사용자는 웹을 통해 기상 정보를 손쉽게 조회한다. 이러한 구조를 통해 실시간성과 사용자 편의성을 동시에 확보할 수 있다.

제 2 장 본 론

제 1 절 리소스

1. 온도(Temperature, tmp)

대기 중의 온도를 나타내는 실시간 측정값

2. 습도(Humidity, humi)

대기 중의 습도 비율을 나타내는 값

3. 강수량(Rainfall, rainf)

일정 기간 동안 내린 강수의 양을 나타내는 값 (mm 단위)

4. 강수 형태(Precipitation, pre)

비, 눈 등 강수의 형태를 나타내는 값

본 프로젝트에서 정의한 리소스는 온도, 습도, 강수량, 강수형태 4가지이며, 이는 Publisher 프로그램의 TopicInfo 클래스 배열에 정의되어 있다. 이 배열을 통해 각각의 기상 항목에 대응하는 MQTT 토픽과 JSON 필드를 지정해 데이터를 발행한다.

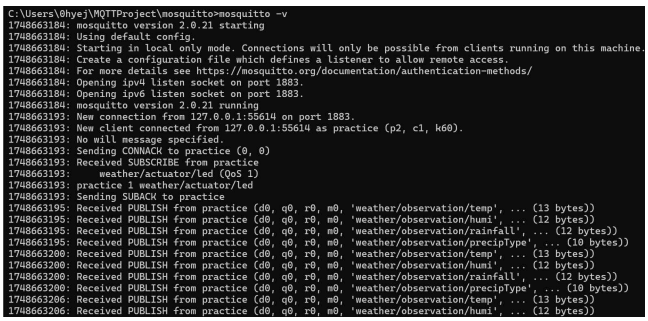
```
// 토픽 배열 선언
TopicInfo[] ObsGrp  = {
        new TopicInfo("weather/observation/temp", "tmp"),
        new TopicInfo("weather/observation/humi", "humi"),
        new TopicInfo("weather/observation/rainfall", "rainf"),
        new TopicInfo("weather/observation/precipType", "pre")
};
```

[그림 2-1] 리소스

제 2 절 MQTT 통신

1. MQTT Broker

Mosquitto는 MQTT 메시지의 송수신 중개자 역할을 한다. Publisher가 보낸 메시지를 해당 토픽의 Subscriber에게 실시간으로 전달해준다. 따라서 Node.js 서버는 Mosquitto와 연결해 데이터를 수신하거나 발행한다. mosquitto –v 명령어를 통해 메시지의 송수신 상태를 실시간으로 확인하며 디버깅할 수 있다.



[그림 2-2] MQTT Broker

2. MQTT Publisher

MQTT Publisher는 MQTT 프로토콜을 이용해 센서 데이터나 상태 정보를 특정 토픽으로 주기적으로 발행하는 역할을 한다. 이 코드에서는 외부 API에서 수집한 기상 관측 데이터를 여러 항목별로 그룹화하여 JSON 형식으로 묶은 뒤, 이를 "weather/observation"이라는 단일 토픽으로 발행한다. MQTT 브로커와 연결을 유지하며, 발행 상태를 확인하여 안정적인 통신을 보장한다. 이렇게 그룹화된 데이터를 발행함으로써, MQTT Subscriber가 효율적으로 최신 기상 정보를 실시간으로 수신할 수 있도록 지원한다.

2.1 리소스 정의 및 토픽 배열 선언

토픽명과 JSON 메시지 내 필드명을 묶어서 관리하기 위해 TopicInfo 클래스를 정의하고, 토픽 배열(ObsGrp)을 선언한다.

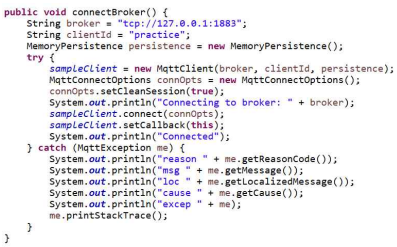
```
// 토픽과 필드 정보를 담는 클래스
static class TopicInfo {
    String topic;
    String jsonField;
    public TopicInfo(String topic, String jsonField) {
        this.topic = topic;
        this.jsonField = jsonField;
    }
}

// 토픽 배열 선언
TopicInfo[] ObsGrp  = {
        new TopicInfo("weather/observation/temp", "tmp"),
        new TopicInfo("weather/observation/humi", "humi"),
        new TopicInfo("weather/observation/rainfall", "rainf"),
        new TopicInfo("weather/observation/precipType", "pre")
};
```

[그림 2-3] 리소스 정의

2.2 MQTT 브로커 연결 및 구독 설정

connectBroker() 메서드에서는 MQTT 브로커 주소와 클라이언트 ID를 지정하고, 메모리 기반 persistence를 사용하여 브로커와 연결한다. 포트번호는 1883을 사용한다. 연결 후에는 콜백을 등록하고, weather/actuator/led 토픽을 구독하여 LED 제어 명령을 수신할 준비를 한다.



```
public void connectBroker() {
    String broker = "tcp://127.0.0.1:1883";
    String clientId = "practice";
    MemoryPersistence persistence = new MemoryPersistence();
    try {
        sampleClient = new MqttClient(broker, clientId, persistence);
        MqttConnectOptions connOpts = new MqttConnectOptions();
        connOpts.setCleanSession(true);
        System.out.println("Connecting to broker: " + broker);
        sampleClient.connect(connOpts);
        sampleClient.setCallback(this);
        System.out.println("Connected");
    } catch (MqttException me) {
        System.out.println("reason " + me.getReasonCode());
        System.out.println("msg " + me.getMessage());
        System.out.println("loc " + me.getLocalizedMessage());
        System.out.println("cause " + me.getCause());
        System.out.println("excep " + me);
        me.printStackTrace();
    }
}
```

[그림 2-4] connectBroker() 메서드

2.3 기상 데이터 수집 및 발행

get_weather_data() 메서드는 기상청의 초단기 실황 API에서 XML 형식으로 데이터를 받아와 Jsoup 라이브러리를 이용해 필요한 항목(기온, 습도, 강수량, 강수형태)을 추출한다.

```
public String[] get_weather_data() {
    Date current = new Date(System.currentTimeMillis());
    SimpleDateFormat d_format = new SimpleDateFormat("yyyyMMddHHmmss");
    String date = d_format.format(current).substring(0, 8);
    String time = d_format.format(current).substring(8, 10);

    String base_time;
    int hour = Integer.parseInt(time);
    if (hour % 3 == 0) {
        base_time = String.format("%02d00", hour);
    } else {
        base_time = String.format("%02d00", (hour / 3) * 3);
    }

    String url = "http://apis.data.go.kr/1360000/VilageFcstInfoService_2.0/getUltraSrtNcst"
            + "?serviceKey=SNgl46lMPdeaFM1357uE4DsNL%2FXQeVycrjXdKSoThh0SQBw2yoDOTgOS2syutF2Yja%2F3dIDLh4e2b197Rv9ACg%3D%3D"
            + "&pageNo=1&numOfRows=1000"
            + "&dataType=XML"
            + "&base_date=" + date
            + "&base_time=" + base_time
            + "&nx=55"
            + "&ny=127";

    String temp = "";
    String humi = "";
    String rainf = "";
    String pre = "";
    Document doc = null;

    try {
        doc = Jsoup.connect(url).get();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

[그림 2-5] get_weather_data() 메서드 1

```
    if (doc != null) {
        Elements elements = doc.select("item");

        for (Element e : elements) {
            String category = e.select("category").text();
            String obsrValue = e.select("obsrValue").text();

            if (category.equals("T1H")) {
                temp = obsrValue;
            }
            if (category.equals("REH")) {
                humi = obsrValue;
            }
            if (category.equals("RN1")) {
                rainf = obsrValue;
            }
            if (category.equals("PTY")) {
                pre = obsrValue;
            }
        }
    }

    return new String[]{temp, humi, rainf, pre};
}
```

[그림 2-6] 메서드 2

2.4 데이터 JSON 변환 및 MQTT 메시지 발행

publishGroup() 메서드는 TopicInfo[] 배열과 데이터 배열을 받아 JSON 형태로 묶은 후, 하나의 통합 토픽 "weather/observation"으로 발행한다. 실제 메시지 발행은 publish_data()에서 수행된다.

```
public void publishGroup(TopicInfo[] group, String[] data) {
    StringBuilder jsonBuilder = new StringBuilder("{");
        for (int i = 0; i < group.length; i++) {
            jsonBuilder.append(String.format("\"%s\": %s", group[i].jsonField, data[i]));
            if (i < group.length - 1) jsonBuilder.append(", ");
        }
        jsonBuilder.append("}");

        String combinedTopic = "weather/observation";
        publish_data(combinedTopic, jsonBuilder.toString());
}
```

[그림 2-7] 데이터 JSON 변환

```
public void publish_data(String topic_input, String data) {
    int qos = 0;
    try {
        System.out.println("Publishing message to " + topic_input + ": " + data);
        sampleClient.publish(topic_input, data.getBytes(), qos, false);
        System.out.println("Message published");
    } catch (MqttException me) {
        System.out.println("reason " + me.getReasonCode());
        System.out.println("msg " + me.getMessage());
        System.out.println("loc " + me.getLocalizedMessage());
        System.out.println("cause " + me.getCause());
        System.out.println("excep " + me);
        me.printStackTrace();
    }
}
```

[그림 2-8] 메시지 발행

2.5 반복 실행 및 예외 처리

run() 메서드에서 브로커 연결 후, 5초 간격으로 데이터를 수집하고 발행한다. 예외 발생 시 연결을 끊고 종료한다.

```
public void run() {
    connectBroker();

    try {
        sampleClient.subscribe("weather/actuator/led");
    } catch (MqttException e1) {
        e1.printStackTrace();
    }

    while (true) {
        try {
            String[] weather_data = get_weather_data();

            publishGroup(ObsGrp, weather_data);

            Thread.sleep(5000);

        } catch (Exception e) {
            try {
                sampleClient.disconnect();
            } catch (MqttException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
            System.out.println("Disconnected");
            System.exit(0);
        }
    }
}
```

[그림 2-9] 반복 실행 및
예외 처리

2.6 메시지 수신 콜백

messageArrived() 메서드는 구독한 토픽에서 메시지가 도착할 때 호출되어 LED 상태 변경 메시지를 출력한다.

```
@Override
public void messageArrived(String topic, MqttMessage msg) throws Exception {
    if (topic.equals("weather/actuator/led")) {
        System.out.println("--------------------Actuator Function--------------------");
        System.out.println("LED Display changed");
        System.out.println("LED: " + msg.toString());
        System.out.println("---------------------------------------------------------");
    }
}
```

[그림 2-10] messageArrived() 메서드

3. MQTT Susbcriber

MQTT Subscriber는 Mosquitto Broker를 통해 실시간 기상 데이터를 수신하며, 이를 MongoDB에 저장한다. Subscriber는 "weather/observation" 토픽을 구독하고, 메시지의 항목별로 데이터를 분리하여 관리한다. 이를 통해 웹 클라이언트가 최신 데이터를 요청할 때, 소켓 통신으로 손쉽게 데이터를 전송할 수 있도록 한다.

3.1 MQTT Subscriber 초기화 및 연결

mqtt 모듈을 사용하여 MQTT 브로커(mqtt://127.0.0.1)에 연결한다. MQTT Subscriber를 초기화한다. "mqtt://127.0.0.1"은 로컬 환경에서 Mosquitto 브로커에 연결하기 위한 URI이다. 이후, "weather/observation" 토픽을 구독하고, Mosquitto Broker가 이 토픽으

로 수신한 메시지를 Subscriber(Node.js 서버)에 전달한다.

```
/**
 * MQTT subscriber
 */
var mqtt = require("mqtt");
var client = mqtt.connect("mqtt://127.0.0.1");

// 상위 토픽 구독
client.on("connect", function () {
  client.subscribe("weather/observation", function () {
    console.log("Subscribed to weather/observation");
  });
});
```

[그림 2-11] MongoDB 연결

3.2 메시지 수신 및 처리

message 이벤트를 통해 구독 중인 토픽의 메시지를 수신하며, 수신된 메시지는 문자열에서 JSON으로 파싱된다. 파싱된 데이터에 수신 시각(create_at)을 추가한 후, MongoDB가 연결된 경우 각 항목(tmp, humi, rainf, pre)을 별도의 컬렉션에 비동기로 저장한다. 저장 작업은 Promise.all로 동시에 처리하며, 오류 발생 시 로그로 출력한다. 이를 통해 메시지를 실시간으로 수신하고, 항목별로 분리하여 데이터베이스에 저장할 수 있다.

```
// MQTT 메시지 수신 시 처리
client.on("message", function (topic, message) {
  if (topic === "weather/observation") {
    console.log(topic + ": " + message.toString());

    let obj;
    try {
      obj = JSON.parse(message.toString());
    } catch (e) {
      console.error("Invalid JSON:", message.toString());
      return;
    }
    obj.create_at = new Date();

    if (!dbClient) {
      console.error("DB not connected yet.");
      return;
    }

    const db = dbClient.db("Resources");
```

[그림 2-12] 메시지 수신, 처리

```
    // 각 항목별로 별도의 컬렉션에 저장
    const insertTasks = [];

    if (obj.tmp !== undefined && obj.tmp !== null) {
      insertTasks.push(
        db.collection("Temperature").insertOne({ tmp: obj.tmp, create_at: obj.create_at })
      );
    }
    if (obj.humi !== undefined && obj.humi !== null) {
      insertTasks.push(
        db.collection("Humidity").insertOne({ humi: obj.humi, create_at: obj.create_at })
      );
    }
    if (obj.rainf !== undefined && obj.rainf !== null) {
      insertTasks.push(
        db.collection("Rainfall").insertOne({ rainf: obj.rainf, create_at: obj.create_at })
      );
    }
    if (obj.pre !== undefined && obj.pre !== null) {
      insertTasks.push(
        db.collection("Precipitation").insertOne({ pre: obj.pre, create_at: obj.create_at })
      );
    }

    Promise.all(insertTasks).catch(err => {
      console.error("Insert error:", err);
    });
  }
});
```

[그림 2-13] 메시지 수신, 처리

3.3 소켓 통신을 통한 최신 데이터 전송

웹 클라이언트가 최신 데이터를 요청할 때, Socket.io를 통해 서버와 실시간으로 통신한다. 클라이언트가 "socket_evt_update" 이벤트를 전송하면, 서버에서는 MongoDB의

각 컬렉션에서 최신 데이터를 조회한다. 이후 조회된 데이터를 JSON 형태로 그룹화하여 클라이언트에 다시 전송한다. 이를 통해 클라이언트는 기상 데이터의 최신 상태를 실시간으로 확인할 수 있다. 클라이언트가 "socket_evt_btn" 이벤트를 통해 LED 제어 등의 액추에이터 요청을 보낼 경우, 서버는 해당 요청 데이터를 다시 MQTT를 통해 발행하여 기상 관측 시스템을 제어할 수 있다.

[그림 2-14] 소켓 통신 데이터 전송

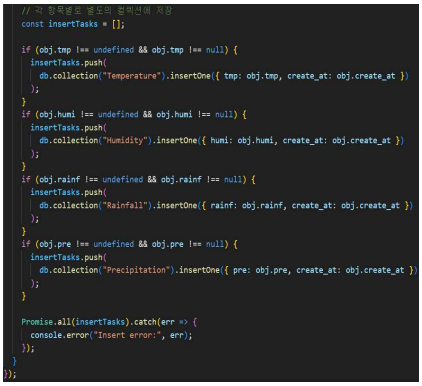
[그림 2-15] 소켓 통신 데이터 전송

제 3 절 웹 서버 및 웹 페이지 구현

1. DB 저장

MQTT를 통해 수신된 날씨 관측 데이터를 MongoDB에 저장한다. MQTT 브로커로부터 weather/observation 토픽으로 전달된 JSON 메시지를 수신하면, 파싱해서 온도, 습도, 강수량, 강수형태 데이터를 추출한다. 각 데이터 항목은 MongoDB의 다른 컬렉션(Temperature, Humidity, Rainfall, Precipitation)에 개별적으로 저장된다.

[그림 2-16] 메시지 수신 처리



[그림 2-17] 메시지 수신 처리

2. 소켓 통신

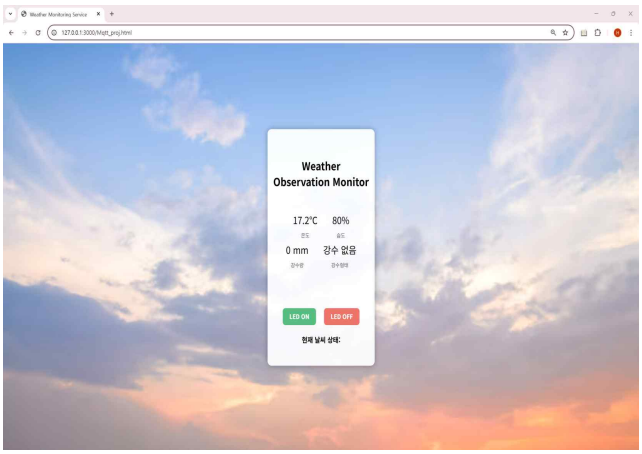
웹 페이지와 서버 간의 실시간 데이터 전달을 위해 Socket.IO를 사용한다. 클라이언트가 socket_evt_update 이벤트를 서버로 전송하면, 서버는 MongoDB에서 최신 날씨 데이터를 항목별로 조회해 JSON 형식으로 클라이언트에 전달한다. 이때, 네 가지 항목(온도, 습도, 강수량, 강수 형태)을 같은 방식으로 처리하기 위해 map()으로 각 항목마다 Promise를 만들어 배열을 생성하고, Promise.all()로 모든 작업이 끝날 때까지 기다려 결과를 모아 클라이언트로 보낸다. LED 제어 버튼이 눌릴 때는 socket_evt_btn 이벤트를 통해 MQTT로 LED 제어 명령을 발행해 사용자가 실시간으로 데이터를 확인하고 LED를 제어할 수 있다.

[그림 2-18] 소켓 통신 처리

[그림 2-19] 소켓 통신 처리

3. 웹 페이지 화면 구성

웹 페이지는 날씨 관측 데이터를 실시간으로 사용자에게 제공한다. 온도, 습도, 강수량, 강수 형태 등을 한눈에 확인할 수 있으며, LED를 켜거나 끄는 기능도 함께 제공한다. 부트스트랩과 jQuery로 구성된 직관적인 UI 덕분에, 사용자들은 손쉽게 정보를 확인하고 LED를 제어할 수 있다. LED를 켜고 끄는 것과 동시에 현재 날씨 상태는 별도로 강조해서 표시되도록 해 사용자가 빠르게 상황을 파악할 수 있다.



[그림 2-20] 웹 페이지 화면

4. 웹 페이지 소스코드

부트스트랩 CSS를 사용해 레이아웃과 스타일을 적용한다. 배경색이나 글자색, 버튼 모양 같은 디자인 요소들도 CSS에서 따로 지정한다. 자바스크립트 부분에서는 Socket.io를 활용해 서버와 실시간으로 연결되도록 했고, 3초마다 서버에 최신 날씨 정보를 요청해서 받아온다. 받은 데이터는 jQuery로 화면에 바로 반영해서 온도, 습도, 강수량, 강수 형태 같은 정보를 계속 업데이트한다. 강수 형태는 숫자로 되어 있어서 사람이 알아보기 쉽게 한글로 바꾸는 함수도 포함되어 있다. LED 켜기, 끄기 버튼을 누르면 서버에 명령이 전달되고, LED가 켜져 있으면 현재 날씨 상태가 화면에 표시되게 만들어서 사용자가 상황을 쉽

게 알 수 있다.

[그림 2-21] html 소스코드 1

[그림 2-22] html 소스코드 2

[그림 2-23] html 소스코드 3

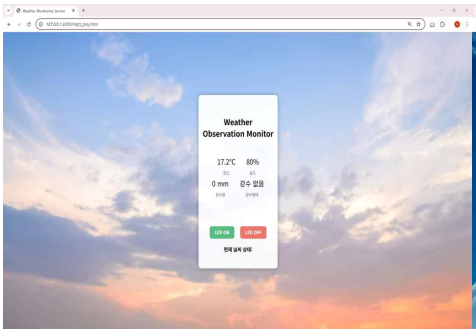
[그림 2-23] html 소스코드 4

제 4 절 실행 결과

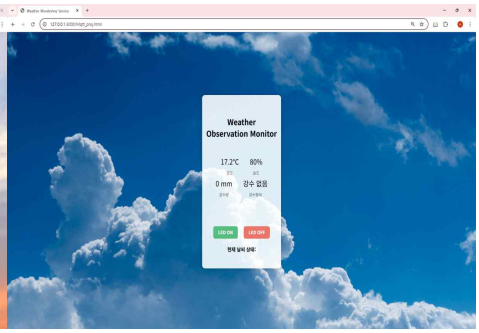
1. 실행 결과

프로젝트를 실행하면 웹 페이지 상에서 실시간으로 온도, 습도, 강수량, 강수 형태 데이터가 업데이트되는 것을 확인할 수 있다. 사용자는 LED ON/OFF 버튼을 눌러 LED 상태를

즉시 제어할 수 있으며, LED가 켜져 있을 때는 현재 날씨 상태가 별도로 강조되어 화면에 표시된다. 데이터는 3초 간격으로 서버와 주기적으로 통신하여 최신 상태를 유지한다.



[그림 2-25] 웹 페이지 화면 1



[그림 2-26] 웹 페이지 화면 2

제 3 장 결 론

본 프로젝트는 실시간 기상 데이터를 API를 통해 수집하고, MQTT 프로토콜로 전송하여 MongoDB에 저장하는 시스템을 구현했다. 웹 페이지는 Socket.IO를 활용한 실시간 통신으로 최신 정보를 사용자에게 제공하며, LED 제어 기능을 포함해 사용자 상호작용이 가능하도록 구성했다. 부트스트랩과 jQuery를 이용해 직관적이고 사용하기 편한 화면을 설계했고, 이를 통해서 사용자는 온도, 습도, 강수량, 강수 형태를 한눈에 확인할 수 있다. LED가 켜져 있을 때는 현재 날씨 상태가 별도로 강조되어 화면에 표시되어, 사용자에게 중요한 정보를 명확하게 전달한다. 누구나 쉽게 실시간 날씨 정보를 확인할 수 있는 웹 기반 기상 모니터링 시스템을 구현했다.

참 고 자 료

- 본 장에서는, 프로젝트 구현을 위해 참고한 자료를 기술할 것

[1] https://velog.io/@broccolism/Promise.all%EC%97%90%EC%84%9C-%EC%97%90%EB%9F%AC-%ED%95%B8%EB%93%A4%EB%A7%81%ED%95%98%EA%B8%B0

[2] https://www.freecodecamp.org/korean/news/javascript-map-method/