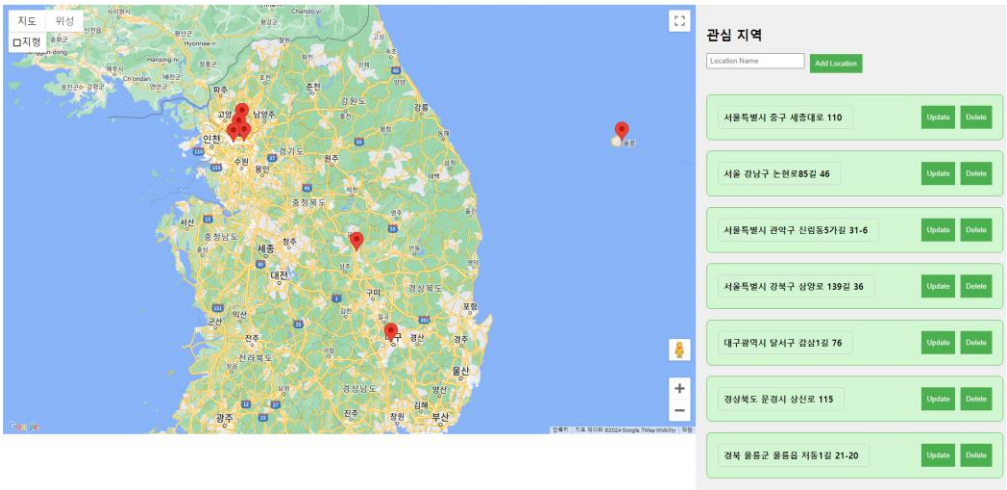
개발 API 설명 및 주의사항

Git link : https://github.com/kimgh5303/Weather_API

프론트 화면 (React)

구글 maps javascript api 를 이용하여 화면을 구성

```
<LoadScript googleMapsApiKey="AIzaSyAruD8tKBw4IEUms327Muaq5clYvn10YxA">
                    <GoogleMap
                    mapContainerStyle={{ height: '800px', width: '70%' }}
                    center={currentPosition || { lat: 0, lng: 0 }}
                    zoom={10}
                    >
                    {currentPosition && <Marker position={currentPosition} />}
                    {locations.map((location) => (
                        <Marker
                            key={location.locIdx}
                            position={{ lat: parseFloat(location.locLatitude), lng: parseFloat(location.locLongitude) }}
                            onClick={() => handleLocationClick(location)}
                        />
                    ))}
                    </GoogleMap>
                </LoadScript>
```



프론트 <-> 백

rest api 로 통신

POST, GET, PUT url 엔드포인트는 http://localhost:8080/weathers/locations 로 동일

DELETE 는 http://localhost:8080/weathers/locations/{locationId}로 구성

백 로직 (springboot)

관심 지역 입력시 그에 관한 정보를 부르는 api 를 구성

➔ 이때, 도로명 주소를 입력해야됨

(구글에서 제공하는 주소 찾기 api 가 무슨 이유에선지 프론트에서 javascript 로 돌아가지 않는 현상이 발생. 그리하여 비교적 신주소인 도로명 주소로 결정)

주소 -> 위,경도 좌표

브이월드에서 제공하는 Geocoder API 2.0 으로 주소를 위,경도 좌표로 변환해줌

```
if (coordDto.getAddress() == null) {
    throw new IllegalArgumentException("주소가 null 입니다.");
}

StringBuilder sb = new StringBuilder("https://api.vworld.kr/req/address");
sb.append("?service=address");
sb.append("&request=getCoord");
sb.append("&format=json");
sb.append("&crs=" + epsg);
sb.append("&key=" + apikey);
sb.append("&type=" + "ROAD");
sb.append("&address=" + URLEncoder.encode(coordDto.getAddress(), StandardCharsets.UTF_8));

try {
    URL url = new URL(sb.toString());
    BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream(),
StandardCharsets.UTF_8));

    JSONParser jspa = new JSONParser();
    JSONObject jsob = (JSONObject) jspa.parse(reader);
    JSONObject jsrs = (JSONObject) jsob.get("response");
    JSONObject jsResult = (JSONObject) jsrs.get("result");
    JSONObject jspoitn = (JSONObject) jsResult.get("point");

    System.out.println(jspoitn.get("x"));
    System.out.println(jspoitn.get("y"));
    double longitude = Double.valueOf((String) jspoitn.get("x")).doubleValue();
```

```
    double latitude = Double.valueOf((String) jspoitn.get("y")).doubleValue();

    coordDto.setLongitude(longitude);
    coordDto.setLatitude(latitude);
} catch (MalformedURLException e) {
    throw new RuntimeException(e);
} catch (IOException e) {
    throw new RuntimeException(e);
} catch (ParseException e) {
    throw new RuntimeException(e);
}
```

변환된 위,경도 좌표를 다음 Weather 객체의 locLatitude 와 locLongitude 에 담아줌

➔ Setter 메서드 사용 : @Setter

```
public class Weather {
    @Id
    @Column(name="loc_id")
    private String locId;
    @Column(name="loc_name")
    private String locName;
    @Column(name="loc_latitude")
    private double locLatitude;
    @Column(name="loc_longitude")
    private double locLongitude;

    @Transient
    private double temp;
    @Transient
    private int windDir;
    @Transient
    private double windSpeed;
    @Transient
    private int humidity;
}
```

위,경도 좌표-> 격자점 X, Y

Geocoder api를 이용하여 구한 좌표값을 그대로 활용시 APPLICATION ERROR 오류가 뜸.

➔ Api 활용가이드 명세서에 적힌 정보의 좌표값은 위,경도 좌표가 아닌 격자점 X,Y를 바탕으로 입력해야함

※ 격자점이란? 지구를 근사화하여 평면 상태에 나타내기 위해 사용

요청변수(Request Parameter)

항목명(국문)	항목명(영문)	항목크기	항목구분	샘플데이터	항목설명
서비스키	ServiceKey	4	필수	-	공공데이터포털에서 받은 인증키
페이지 번호	pageNo	4	필수	1	페이지번호
한 페이지 결과 수	numOfRows	4	필수	1000	한 페이지 결과 수
응답자료형식	dataType	4	옵션	XML	요청자료형식(XML/JSON) Default: XML
발표일자	base_date	8	필수	20210628	'21년 6월 28일 발표
발표시각	base_time	4	필수	0600	06시 발표(정시단위)
예보지점 X 좌표	nx	2	필수	55	예보지점의 X 좌표값
예보지점 Y 좌표	ny	2	필수	127	예보지점의 Y 좌표값

격자식 (기상청 제공)

Java

```
double RE = 6371.00877; // 지구 반경(km)
double GRID = 5.0; // 격자 간격(km)
double SLAT1 = 30.0; // 투영 위도 1(degree)
double SLAT2 = 60.0; // 투영 위도 2(degree)
double OLON = 126.0; // 기준점 경도(degree)
double OLAT = 38.0; // 기준점 위도(degree)
double XO = 43; // 기준점 X 좌표(GRID)
double YO = 136; // 기 1 준점 Y 좌표(GRID)

// LCC DFS 좌표변환 ( code : "TO_GRID"(위경도->좌표, lat_X:위도,  lng_Y:경도), "TO_GPS"(좌표-
>위경도,  lat_X:x, lng_Y:y) )
```

```
double DEGRAD = Math.PI / 180.0;
double re = RE / GRID;
double slat1 = SLAT1 * DEGRAD;
double slat2 = SLAT2 * DEGRAD;
double olon = OLON * DEGRAD;
double olat = OLAT * DEGRAD;

double sn = Math.tan(Math.PI * 0.25 + slat2 * 0.5) / Math.tan(Math.PI * 0.25 + slat1 * 0.5);
sn = Math.log(Math.cos(slat1) / Math.cos(slat2)) / Math.log(sn);
double sf = Math.tan(Math.PI * 0.25 + slat1 * 0.5);
sf = Math.pow(sf, sn) * Math.cos(slat1) / sn;
double ro = Math.tan(Math.PI * 0.25 + olat * 0.5);
ro = re * sf / Math.pow(ro, sn);
double ra = Math.tan(Math.PI * 0.25 + (weather.getLocLatitude()) * DEGRAD * 0.5);
ra = re * sf / Math.pow(ra, sn);
double theta = weather.getLocLongitude() * DEGRAD - olon;
if (theta > Math.PI) theta -= 2.0 * Math.PI;
if (theta < -Math.PI) theta += 2.0 * Math.PI;
theta *= sn;
int x = (int)Math.floor(ra * Math.sin(theta) + XO + 0.5);
int y = (int)Math.floor(ro - ra * Math.cos(theta) + YO + 0.5);
```

기상청 초단기 실황 api

기상청에서 제공하는 초단기실황은 각 정시의 30분에 업데이트 되므로 그 이전 정시를 기준으로 가져와야함

Ex) 6시 27분 -> 5시

시간 설정

➔ 시간에서 분은 제외해야하므로 "HH00"으로 포맷 형식을 설정

```
LocalDateTime currentDateTime = LocalDateTime.now();
        // 현재 시간에서 가장 가까운 이전 정시 계산
        int currentHour = currentDateTime.getHour();
        int closestHour = (currentHour - 1 + 24) % 24; // 이전 정시 계산
        // 형식화하여 출력 (뒤에 "00" 붙이기)
        DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("yyyyMMdd");
        DateTimeFormatter hourFormatter = DateTimeFormatter.ofPattern("HH00");
        // 이전 정시로부터 1시간 전의 시간을 계산
        LocalDateTime previousHour = currentDateTime.withHour(closestHour);
```

```
        String formattedDate = previousHour.format(dateFormatter);
        String formattedHour = previousHour.format(hourFormatter);
        System.out.println("Formatted Date: " + formattedDate);
        System.out.println("Formatted Hour: " + formattedHour);
```

다음과 같이 기상청 api를 이용해 초단기실황을 XML 데이터 형태로 가져옴

```
StringBuilder urlBuilder = new
StringBuilder("http://apis.data.go.kr/1360000/VilageFcstInfoService_2.0/getUltraSrtNcst"); /*URL*/
urlBuilder.append("?" + URLEncoder.encode("serviceKey", "UTF-8") + "=" + apikey); /*Service Key*/
urlBuilder.append("&" + URLEncoder.encode("pageNo", "UTF-8") + "=" + URLEncoder.encode("1", "UTF-
8")); /*페이지번호*/
urlBuilder.append("&" + URLEncoder.encode("numOfRows", "UTF-8") + "=" + URLEncoder.encode("1000",
"UTF-8")); /*한 페이지 결과 수*/
urlBuilder.append("&" + URLEncoder.encode("dataType", "UTF-8") + "=" + URLEncoder.encode("XML",
"UTF-8")); /*요청자료형식(XML/JSON) Default: XML*/
urlBuilder.append("&" + URLEncoder.encode("base_date", "UTF-8") + "=" +
URLEncoder.encode(formattedDate, "UTF-8")); /* '21 년 6 월 28 일 발표*/
urlBuilder.append("&" + URLEncoder.encode("base_time", "UTF-8") + "=" +
URLEncoder.encode(formattedHour, "UTF-8")); /*06 시 발표(정시단위) */
urlBuilder.append("&" + URLEncoder.encode("nx", "UTF-8") + "=" +
URLEncoder.encode(Integer.toString(x), "UTF-8")); /*예보지점의 X 좌표값*/
urlBuilder.append("&" + URLEncoder.encode("ny", "UTF-8") + "=" +
URLEncoder.encode(Integer.toString(y), "UTF-8")); /*예보지점의 Y 좌표값*/
URL url = new URL(urlBuilder.toString());
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Content-type", "application/json");
System.out.println("Response code: " + conn.getResponseCode());
BufferedReader rd;
if (conn.getResponseCode() >= 200 && conn.getResponseCode() <= 300) {
    rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
} else {
    rd = new BufferedReader(new InputStreamReader(conn.getErrorStream()));
}
StringBuilder sb = new StringBuilder();
String line;
while ((line = rd.readLine()) != null) {
    sb.append(line);
}
rd.close();
conn.disconnect();
String xmlData = sb.toString();
```

다음과 같은 XML 형식의 데이터를 가져옴

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <header>
        <resultCode>00</resultCode>
        <resultMsg>NORMAL_SERVICE</resultMsg>
    </header>
    <body>
        <dataType>XML</dataType>
        <items>
            <item>
                <baseDate>20240110</baseDate>
                <baseTime>0400</baseTime>
                <category>PTY</category>
                <nx>60</nx>
                <ny>127</ny>
                <obsrValue>0</obsrValue>
            </item>
            <item>
                <baseDate>20240110</baseDate>
                <baseTime>0400</baseTime>
                <category>REH</category>
                <nx>60</nx>
                <ny>127</ny>
                <obsrValue>92</obsrValue>
            </item>
            <item>
                <baseDate>20240110</baseDate>
                <baseTime>0400</baseTime>
                <category>RN1</category>
                <nx>60</nx>
                <ny>127</ny>
                <obsrValue>0</obsrValue>
.....생략
```

기온, 풍향, 풍속, 습도를 추출하고 싶으므로 XML 데이터에서 각 코드에 해당하는 것을 추출

- ➔ 기온 : T1H
- ➔ 풍향 : VEC
- ➔ 풍속 : WSD
- ➔ 습도 : REH

```
try {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    ByteArrayInputStream input = new ByteArrayInputStream(xmlData.getBytes("UTF-8"));
```

```
    Document document = builder.parse(input);

    // 기온 카테고리의 obsrValue 추출
    NodeList itemList = document.getElementsByTagName("item");
    for (int index = 0; index < itemList.getLength(); index++) {
        Node itemNode = itemList.item(index);

        if (itemNode.getNodeType() == Node.ELEMENT_NODE) {
            Element itemElement = (Element) itemNode;

            // 카테고리 추출
            NodeList categoryNodeList = itemElement.getElementsByTagName("category");
            Node categoryNode = categoryNodeList.item(0);
            String category = categoryNode.getTextContent();

            // obsrValue 추출
            NodeList obsrValueNodeList = itemElement.getElementsByTagName("obsrValue");
            Node obsrValueNode = obsrValueNodeList.item(0);
            String obsrValue = obsrValueNode.getTextContent();

            // 각 카테고리에 따라 필드에 할당
            switch (category) {
                case "T1H":
                    weather.setTemp(Double.parseDouble(obsrValue));
                    System.out.println("T1H obsrValue: " + obsrValue);
                    break;
                case "VEC":
                    weather.setWindDir(Integer.parseInt(obsrValue));
                    System.out.println("VEC obsrValue: " + obsrValue);
                    break;
                case "WSD":
                    weather.setWindSpeed(Double.parseDouble(obsrValue));
                    System.out.println("WSD fcstValue: " + obsrValue);
                    break;
                case "REH":
                    weather.setHumidity(Integer.parseInt(obsrValue));
                    System.out.println("REH fcstValue: " + obsrValue);
                    break;
            }
        }
    }
```

Weather 객체에 추가되면 프론트로 전송해줌


```
{
    "success": true,
    "message": "조회 완료",
    "data": [
        {
            "locId": "0490837d-6e15-4d1b-9fa6-e9e0fe7d411b",
            "locName": "서울특별시 중구 세종대로 110",
            "locLatitude": 37.566700969,
```

```
        "locLongitude": 126.97834678,
        "temp": -3.6,
        "windDir": 180,
        "windSpeed": 0.6
    },
    {
        "locId": "07fcd3c8-151f-4921-b6be-fec9e0ba064c",
        "locName": "경상북도 문경시 상신로 115",
        "locLatitude": 36.599759833,
        "locLongitude": 128.187457664,
        "temp": -2.3,
        "windDir": 283,
        "windSpeed": 1.9
    },
    {
        "locId": "446d93a5-b034-447b-87b8-e77d6fd1d039",
        "locName": "서울 강남구 논현로 85 길 46",
        "locLatitude": 37.498246245,
        "locLongitude": 127.034306008,
        "temp": -1.1,
        "windDir": 309,
        "windSpeed": 1.3
    },
    {
        "locId": "a427aa73-c145-4533-86d9-9917cebd83f6",
        "locName": "서울특별시 관악구 신림동 5 가길 31-6",
        "locLatitude": 37.48625087,
        "locLongitude": 126.925579364,
        "temp": -4.1,
        "windDir": 30,
        "windSpeed": 0.6
    },
    {
        "locId": "c8aeb574-2aba-4563-a2c6-9c719bc90428",
        "locName": "서울특별시 강북구 삼양로 139 길 36",
        "locLatitude": 37.64854958,
        "locLongitude": 127.01217755,
        "temp": -4.2,
        "windDir": 56,
        "windSpeed": 1.1
    },
    {
        "locId": "dd2d4642-3cfc-4353-b29f-d2792d27b2f5",
        "locName": "대구광역시 달서구 감삼 1 길 76",
        "locLatitude": 35.846574289,
        "locLongitude": 128.538242346,
        "temp": 1.0,
        "windDir": 332,
        "windSpeed": 1.5
    }
```

```
    ]
}
```

프론트에서는 다음과 같이 출력해주면 됨

```
{selectedLocation && (
                <div className="modal" style={{ position: 'fixed', top: 0, left: 0, width: '100%', height:
'100%', backgroundColor: 'rgba(0, 0, 0, 0.5)', display: 'flex', justifyContent: 'center', alignItems:
'center' }}>
                <div style={{ backgroundColor: 'white', padding: '20px', borderRadius: '10px' }}>
                    <h3>Location Details</h3>
                    <p>장소: {selectedLocation.locName}</p>
                    <p>위도: {selectedLocation.locLatitude}</p>
                    <p>경도: {selectedLocation.locLongitude}</p>
                    <p>기온: {selectedLocation.temp}℃</p>
                    <p>풍향: {selectedLocation.windDir}°</p>
                    <p>풍속: {selectedLocation.windSpeed}m/s</p>
                    <p>습도: {selectedLocation.humidity}%</p>
                    <button onClick={closeModal}>Close</button>
                </div>
                </div>
            )}
```

다음과 같이 데이터가 넘어온 것을 볼 수 있음