최종보고서

2021년 12월 14일

2021년 2학기 소프트웨어시스템실습

담당교수: 최성종

김민호

2017440018@office.uos.ac.kr

서울시립대학교 전자전기컴퓨터공학부

차 례

[1. 요약 1](#_Toc90354252)

[2. 요구사항 도출 2](#_Toc90354253)

[2.1. 전국 현황 숫자로 출력 2](#_Toc90354254)

[2.2. 지역별 문자 확인 3](#_Toc90354255)

[2.3. 광역지자체별 현황 확인 4](#_Toc90354256)

[2.4. 비기능적 요구사항 5](#_Toc90354257)

[3. 설계 6](#_Toc90354258)

[3.1. 사용 기술 6](#_Toc90354259)

[3.2. 구조도 7](#_Toc90354260)

[3.3. 흐름도 7](#_Toc90354261)

[4. 구현 10](#_Toc90354262)

[4.1. 스타일 지정(8~120) 10](#_Toc90354263)

[4.2. 라이브러리 호출 및 전역 변수 지정(120~132) 11](#_Toc90354264)

[4.3. 지역코드 API 호출, setLoading() 함수(134~158) 12](#_Toc90354265)

[4.4. 전국 단위 문자 현황 요청(160~231) 13](#_Toc90354266)

[4.5. 지역 단위 문자 현황 요청(233~324) 15](#_Toc90354267)

[4.6. deleteMarker() 함수(326~334) 17](#_Toc90354268)

[4.7. 화면에 표시할 div 객체(337~369) 17](#_Toc90354269)

[4.8. 처음에 자동으로 수행되는 함수(370~387) 18](#_Toc90354270)

[4.9. 네이버 지도 출력(389~424) 20](#_Toc90354271)

[4.10. findCenter() 함수(426~453) 21](#_Toc90354272)

[4.11. makerEvent() 함수(455~490) 22](#_Toc90354273)

[4.12. search() 함수(402~510) 23](#_Toc90354274)

[5. 토론 24](#_Toc90354275)

[5.1. 요구사항의 완성도 24](#_Toc90354276)

[5.2. 변수와 함수의 통일 24](#_Toc90354277)

[5.3. 경상남도 창원시 25](#_Toc90354278)

[5.4. 전체적인 디자인 25](#_Toc90354279)

[6. 결론 26](#_Toc90354280)

[7. 전체 소스코드 27](#_Toc90354281)

[8. 참고자료 44](#_Toc90354282)

요약

전국단위, 지역단위로 재난문자 발령 현황을 출력하는 웹사이트를 작성한다. 네이버 지도 API를 사용하여 화면에 지도를 출력한다. 실시간 정보인 재난문자 현황은 XML 파일로 불러와 사용하고, 마커의 위치 정보는 JSON 파일로서 개인 페이지에 저장된 파일을 다운받아 사용한다. 구현 결과, 기능적 요구사항을 모두 만족하였다. 하지만 비기능적 요구사항 몇 가지는 미흡한 부분이 있었다. 향후 미흡한 부분을 보완하여 개발을 해야 할 것이다.

요구사항 도출

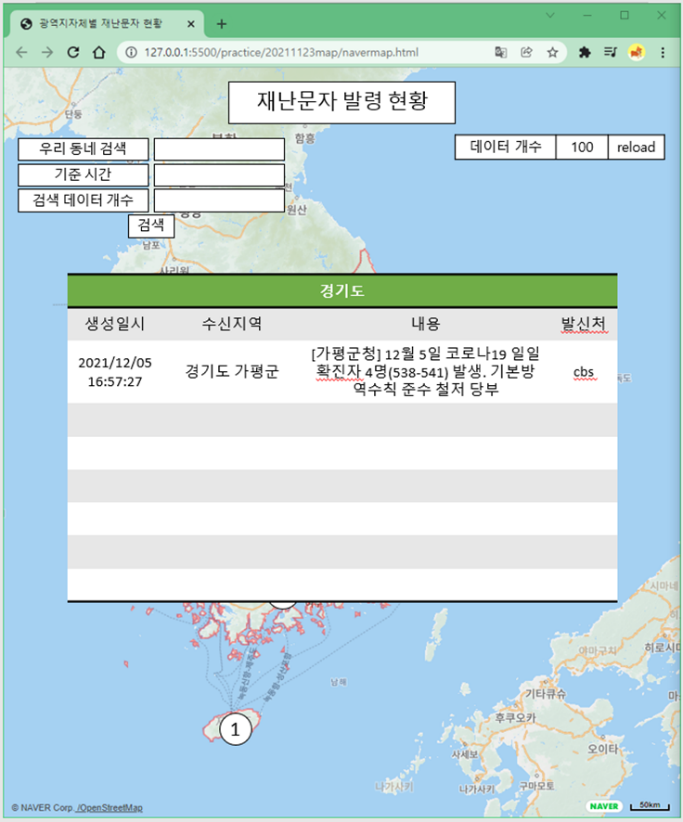
전국 현황 숫자로 출력

지도이(가) 표시된 사진

자동 생성된 설명

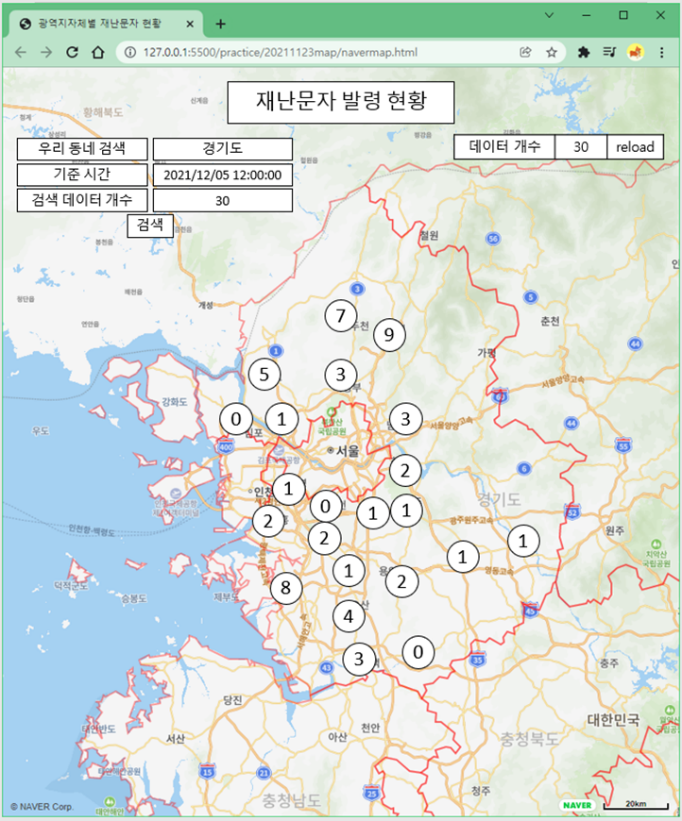
HTML을 처음 실행시키면 위와 같이 전국 지도가 한 눈에 보이도록 화면을 설정한다. 그리고 전국 단위 재난 문자 현황을 받아온다. 받아 온 재난 문자는 지역별로 분류하여 각 지역에 몇 개의 문자가 있는지를 해당 지역 위에 출력해야 한다.

지역별 문자 확인



위에서 문자 개수를 표시하는 동그라미를 클릭하면 위와 같이 해당 지역에 발령된 문자의 상세 정보를 출력한다. 화면 가운데에 표를 띄운다. 표는 화면을 가리지 않기 위해 최대 크기가 정해져 있으며, 만약 데이터가 많다면 스크롤을 통해 모든 정보를 확인할 수 있도록 해야 한다.

광역지자체별 현황 확인



좌측 상단에 특정 지역, 시간, 데이터 개수를 설정하여 해당 지역의 재난문자 발령 현황을 확인할 수 있도록 하는 검색 창이 있다. 이 부분에 정보를 입력한 후 ‘검색’버튼을 누르면 화면이 해당 지역을 확대 해 보여준다. 이후 해당 지역단위 재난문자 발령 현황을 불러와 시 또는 군별로 각 지역에 몇 개의 문자가 발령되었는지를 표시한다. 앞선 전국 단위와 마찬가지로, 문자 개수를 표시하는 동그라미를 클릭하면 해당 지역에 발령된 문자를 확인할 수 있는 표가 화면에 떠야 한다.

비기능적 요구사항

재난문자를 불러오는 것은 시간이 오래 소요될 수 있다. 그러면 사용자에게 현재 데이터를 불러오는 중이라는 것을 알려야 한다. 또한 그동안은 문자 정보를 알 수 없으니 문자 개수를 확인하는 동그라미가 화면에 출력되지 말아야 한다. 또는 동그라미를 클릭하더라도 정보가 담기지 않은 테이블을 띄워서는 안된다.

특정 지역에 대해 검색하기 위해 화면이 이동 할 때에는 부드럽게 화면이 이동하여 위화감을 최대한 줄이도록 한다.

어느 지역에 문자가 많이 발령되었는지 한 눈에 알아볼 수 있도록 동그라미의 바탕 색깔을 조절한다. 많은 문자가 발령된 지역의 동그라미는 진한 바탕을 가져 한 눈에 문자 발령 현황의 분포를 쉽게 알 수 있도록 한다.

전체적인 인터페이스는 깔끔해야 하며, 사용함에 있어 불편이 없어야 한다.

설계

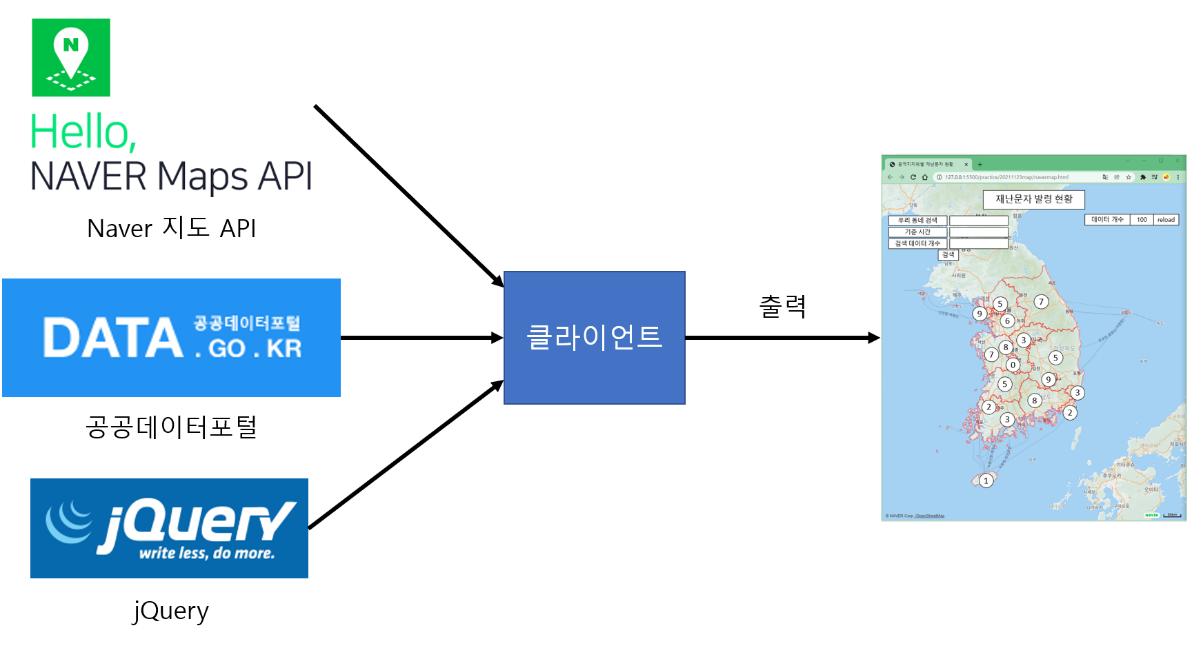
사용 기술

네이버 지도 API를 사용하여 화면에 지도를 출력한다. 네이버 지도 API는 JavaScript 형태로 제공되는 네이버의 지도 플랫폼으로, 웹 서비스 또는 애플리케이션에 지도를 구현할 수 있도록 한다. 프로젝트의 핵심 기술로, 화면 출력, 마커 출력 등 대부분의 핵심 요소가 네이버 지도 API를 통해 이루어진다.

AJAX를 이용하여 서버에 요청을 보내고 XML 파일을 받아온다. 재난문자 정보는 실시간으로 바뀌는 데이터로, 공공데이터포털의 서버에 저장되어 있다. 이를 불러오기 위해 서버에 요청을 보내고 XML 형식의 파일을 받아온다. XML 형식은 클라이언트와 서버, 클라이언트와 클라이언트 등 네트워크에서 데이터를 주고받을 때 사용되는 일반적인 형식이다. 따라서 실시간 정보인 재난문자 현황을 XML 파일로 불러와 사용하게 된다.

동그라미가 위치할 좌표는 JSON 파일에 저장되어 있다. JSON은 정보를 저장하고 이동시키기 위한 텍스트 형식이다. 키-값 형식의 구조로 되어있으며, 오직 텍스트 형식으로만 구성되어 있다. 따라서 프로그래밍 언어에 의존되지 않으며 다른 컴퓨터 간에 쉽게 정보를 보낼 수 있다. JSON 라이브러리의 parse() 함수를 사용하면 텍스트 형식의 JSON 파일을 자바스크립트에서 사용할 수 있는 객체로 만들 수 있다. 지역의 좌표에 대한 정보는 실시간으로 바뀌는 정보가 아니다. 따라서 서버에게 요청하는 방식이 아니라, 파일을 개인 페이지에 저장 해 놓고 문서 실행시에 이를 다운 받아 사용하도록 할 것이다.

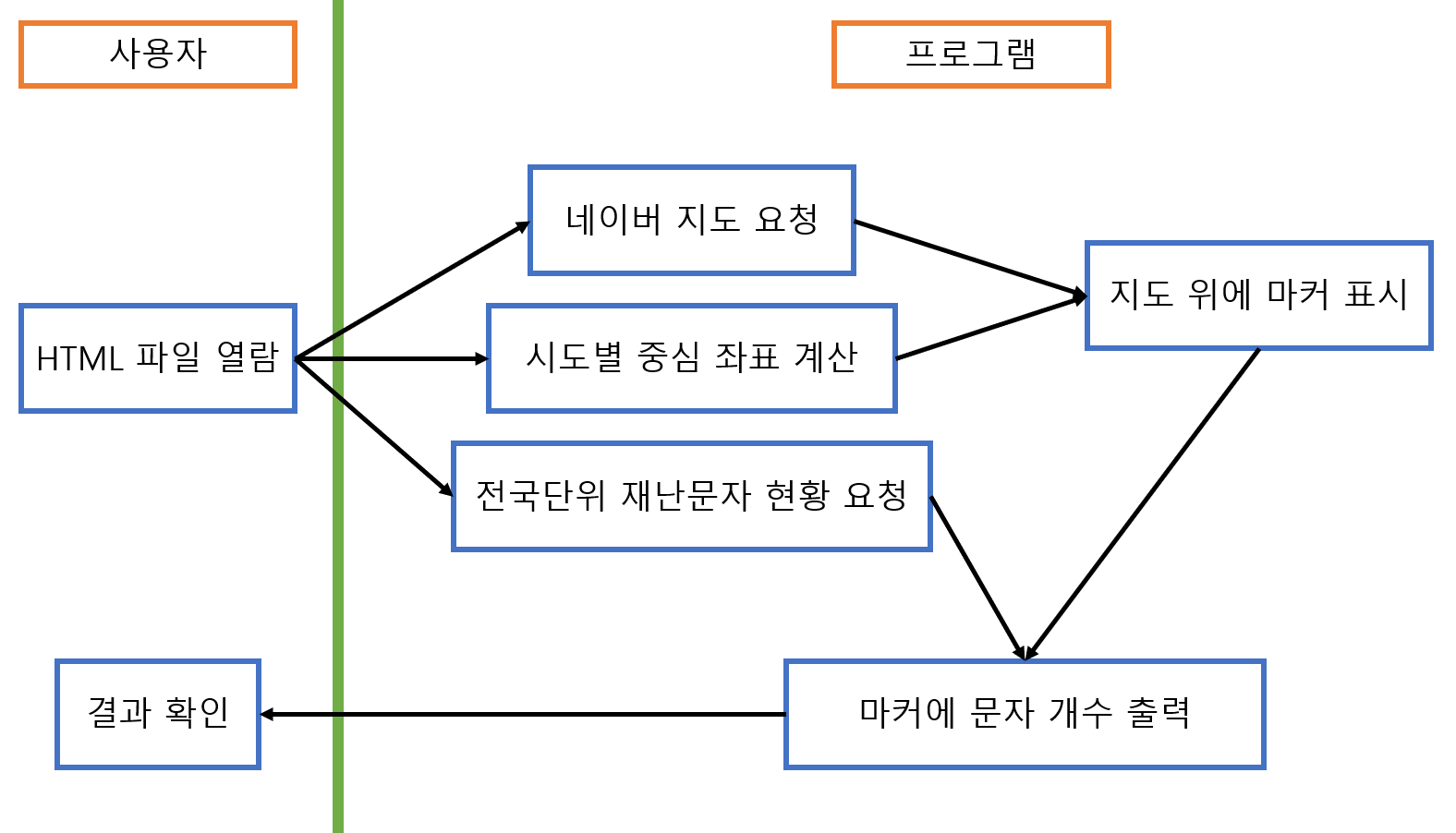
구조도



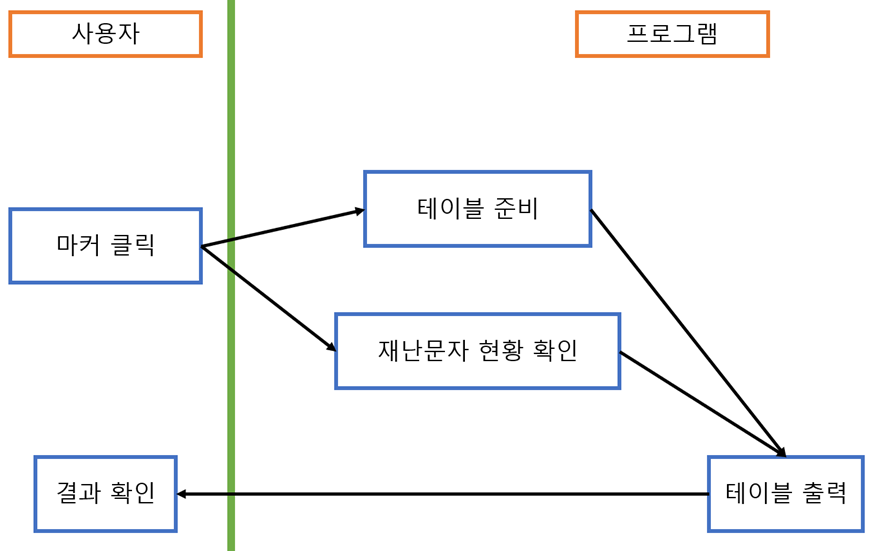
소프트웨어의 구조는 위와 같다. 외부에서 받아 올 정보는 네이버 지도 API, 공공데이터포털의 재난문자 정보, 지역별 재난문자 정보, 재난문자의 지역코드 API, 그리고 jQuery이다.

흐름도

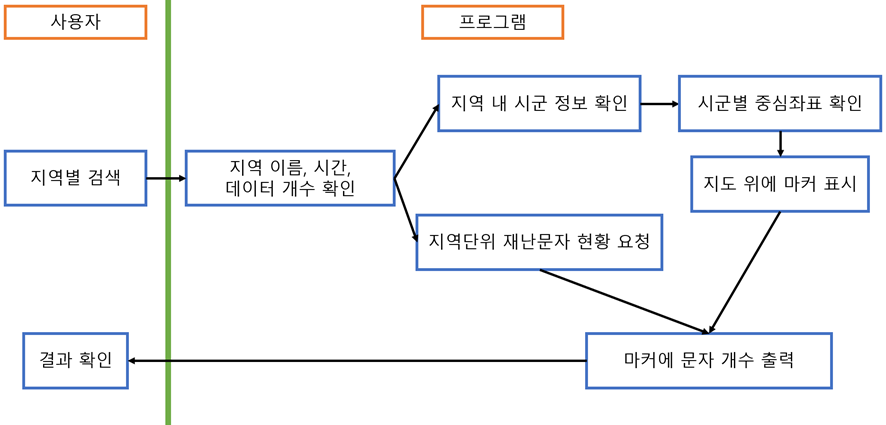
2. 요구사항 도출에 의하면, 사용자의 입력으로 인해 시작되는 시나리오는 크게 3가지이다. 각 시나리오별 프로그램의 흐름을 설계한다.



사용자가 최초에 HTML파일을 열람하면, 네이버 지도를 요청하여 화면에 꽉 차게 띄운다. 시도별 중심좌표 정보를 얻어 해당 위치에 마커를 띄운다. 동시에, 전국단위 재난문자 현황을 요청하여 해당 정보를 받는다. 재난문자 현황에 대한 응답이 도착하면, 그 정보를 분류하여 지역별 마커에 문자 개수를 출력하면 사용자가 화면으로 정보를 확인할 수 있게 된다.



두 번째로 사용자가 마커를 클릭했을 경우이다. 마커를 클릭하면 받아온 재난문자 현황을 테이블로 화면에 출력해야 한다. 이를 위해 테이블이 들어있는 div를 만들어야 하며, 이 div에 문자 정보를 입력하는 함수가 필요 할 것이다. 테이블에 문자 정보가 입력되면 테이블을 화면상에 출력하여 사용자가 화면으로 정보를 확인할 수 있게 된다.



마지막으로 사용자가 화면 좌측 상단의 지역별 검색을 실행 한 경우이다. 프로그램은 지역의 이름, 시간, 요청할 데이터 개수 정보를 받는다. 이를 이용하여 지역단위 재난문자 현황을 서버에 요청한다. 동시에, 선택한 지역 안의 시군을 알아내야 한다. 시군 목록을 알아낸 후에는 각각의 중심좌표를 얻어 그 위치에 마커를 표시한다. 지역단위 재난문자 현황에 대한 응답을 받으면 이를 분류하여 마커에 문자 개수를 출력하면 사용자가 화면으로 정보를 확인할 수 있게 된다.

구현

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 모니터, 스크린샷, 검은색이(가) 표시된 사진

자동 생성된 설명스타일 지정(8~120)

문서에서 사용할 element들의 스타일을 지정한다. 화면에 스크롤 없이 꽉 차게 출력하기 위해 body의 margin을 0px로 하였다. 표의 각 행은 bgColors라는 클래스로 지정되는데, 이 중에서 짝수 행의 바탕색을 다르게 하기 위해 .bgColors:nth-child(even)의 형식을 통해 스타일을 지정하였다. 화면에 표시되는 마커들의 클래스는 circle로 지정할 것이다. border-radius를 50%로 하여 원형으로 표현하였다.

라이브러리 호출 및 전역 변수 지정(120~132)

텍스트이(가) 표시된 사진

자동 생성된 설명

121~126줄에서는 프로그램에서 사용 할 라이브러리를 호출한다. 네이버 지도 API, jQuery, 그리고 시군구 중심좌표를 저장한 JSON파일을 호출한다. 시군구 중심좌표를 저장한 JSON의 경우 개인 GitHub page에 저장 해 놓았고, 이를 불러오고 있다.

128~132줄은 프로그램에서 사용할 전역 변수들이다. raw\_Massage는 재난문자 정보 요청에 대한 응답을 저장 할 변수이다. massage는 2차원배열로 사용할 것인데, 각 지역단위별 메시지가 들어갈 것이다. globalName은 광역지자체 이름, localName은 선택한 광역지자체의 시, 군 이름이다. regionCode는 다음에 호출할 재난문자에 사용되는 지역코드에 대한 정보가 들어간다.

지역코드 API 호출, setLoading() 함수(134~158)

텍스트이(가) 표시된 사진

자동 생성된 설명

getRegionCode() 함수는 공공데이터포털에 지역코드에 대한 오픈 API요청을 한다. 요청이 완료되어 응답이 오면 그 반환값은 XML 형식이지만 사용하고자 하는 부분은 문자열로 온다. 따라서 JSON.parse()함수를 이용하여 JSON 형식으로 바꾸어 주었다. 이를 전역변수인 regionCode에 저장한다.

setLoading() 함수는 화면에 로딩중임을 알리는 div인 loading을 조작하는 함수이다. 입력으로 들어 온 값이 true라면 display값을 inline-block으로 하여 div를 화면에 출력하고, false라면 display값을 none으로 하여 보이지 않도록 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명전국 단위 문자 현황 요청(160~231)

loadGlobal() 함수는 공공데이터포탈에 재난문자 발령 현황을 요청한다. 우선, 요청을 시작하기 전에 setLoading(true)를 통해 로딩중임을 알린다. 입력으로 받은 num만큼 row를 호출 할 것이다. 호출이 종료되고 정상 응답이 돌아오면 얻어온 정보를 raw\_Massage에 저장한 뒤 setLoading(false)를 통해 로딩이 끝났음을 알린다.

다음으로 makeGlobal‌Marker()를 통해 지도상에 마커를 표시한다. 이때, 위치 정보는 regionGeo‌Json에 저장되어 있다. 이 값은 네이버 지도 호출시에 받아오는 정보로 광역지자체별 지적 정보를 담고 있다. 중앙 좌표는 findCenter()함수를 사용하여 얻는다. 이 함수는 아래에서 설명 할 것이다. 마커는 네이버 API의 Marker 클래스를 사용한다. 마커의 모양은 HtmlIcon 형식으로, HTML 요소를 아이콘으로 사용할 수 있다. 지역 이름을 id값으로 갖는 circle 클래스의 div 객체를 넣어 주었다. 마커들은 markerList라는 리스트에 저장된다.

마커를 지도에 표시한 후에는 setGlobalNumber() 함수를 통해 마커에 숫자를 입력한다. 우선, raw\_Massage의 각 데이터들에 접근한다. 그리고 globalName을 통해 반복문을 수행하는데, 각 지역 이름을 이용하여 raw\_Massage의 각 데이터의 지역 이름에 현재 선택된 지역 이름이 포함된 수를 globalNumber에 저장하고, 그런 경우 massage 리스트에 해당 데이터를 넣는 방식이다. 모든 반복이 종료되면 이번에는 지역별로 반복을 수행한다. 각 지역별 마커에 접근하여 globalNumber 값만큼 innerHTML을 수정한다. 그리고 정보가 많으면 바탕색이 진하게 출력되도록 스타일을 지정하였다.

setGlobalNumber()함수로 마커에 문자 개수를 넣어 준 후에는 마커에 이벤트를 달아주어야 한다. 마커를 클릭하면 표가 출력되도록 하는 이벤트이다. 이는 markerEvent() 함수로 수행하는데, 아래에서 설명 할 것이다.

텍스트, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명지역 단위 문자 현황 요청(233~324)

텍스트이(가) 표시된 사진

자동 생성된 설명loadLocal() 함수의 진행은 위에서 설명한 loadGlobal() 함수의 진행과 유사하다. 우선 공공데이터포털을 통해 지역별 재난문자 발령 데이터를 받아온다. 그 정보를 raw\_Massage에 저장한다.

이후 setLocalName()함수를 호출한다. loadGlobal()의 경우 표시해야 할 마커가 광역지자체이기 때문에 정해져 있다. 따라서 사전에 globalName 리스트를 만들어 사용할 수 있었지만 이번에는 그때그때 이 리스트를 만들어야 한다. 이를 위한 함수가 setLocalName()이다. 지역 코드 API를 저장했던 region‌Code를 사용하여 “발송지역(시도)” 값이 선택한 광역지자체인 레코드들의 “발송지역(시군구)”값을 불러와 리스트로 만든다. 이를 전역변수인 localName에 저장한다.

localName을 만든 후에는 loadGlobal()과 유사하게 makeLocalMarker() 함수를 통해 마커를 지도위에 표시한 뒤 setLocalNumber() 함수를 통해 마커에 숫자를 입력한다. 마지막으로 markerEvent() 함수를 통해 마커에 이벤트를 달아주면 지역단위 재난문자 호출이 마무리된다.

deleteMarker() 함수(326~334)

텍스트이(가) 표시된 사진

자동 생성된 설명

이 함수는 화면에 표시된 모든 circle 클래스 객체를 지운다. 즉, 모든 마커를 지우는 함수이다. 새로 마커를 표시하기 전에 항상 이 함수를 실행해야 한다.

화면에 표시할 div 객체(337~369)

텍스트이(가) 표시된 사진

자동 생성된 설명

id가 title인 div객체는 화면 상단 중앙부에 위치하는 제목이다.

id가 loading인 div객체는 데이터를 불러오는 동안 표시될 로딩 안내이다.

id가 search인 div 객체는 화면 좌상단에 위치하는 검색창이다. 우선, 검색 지역으로 광역지자체 17가지와 전국을 합하여 18가지 중 하나를 선택하는 select객체를 통해 입력을 받는다. 기준 시간은 <input type=’text’>를 이용하여 문자열을 받는다. 검색 데이터 개수 또한 <input type=’text’>를 이용하여 문자열을 받고, 기본값은 100이다.

id가 map인 div 객체는 화면 전체에 표시될 지도 객체가 들어가는 곳이다.

id가 table인 div 객체는 문자의 상세 정보를 출력하는 객체로, 문서 상단의 style에서 기본으로 display값을 none으로 하여 처음에는 보이지 않도록 하였다.

id가 close인 div 객체는 table을 끄는 버튼을 포함한다.

처음에 자동으로 수행되는 함수(370~387)

텍스트이(가) 표시된 사진

자동 생성된 설명

371줄에서는 처음에 호출한 시군구 중심좌표가 저장된 JSON 데이터를 전역변수 center에 넣는다. 이 center는 아래의 findCenter() 함수에서 사용 할 것이다.

373~375줄은 현재 시간정보를 검색창의 시간 검색 부분에 기본값으로 넣기 위한 부분이다.

377~380줄은 id가 close인 객체에 이벤트를 추가한다. 이 이벤트 함수는 자신과 id가 table인 객체의 display값을 none으로 수정하는 것으로, 화면에 팝업 형식으로 출력된 table을 끄는 역할을 수행하게 된다.

382~384줄은 id가 submit인 객체에 search() 함수를 이벤트로 추가한다. search() 함수는 아래에서 설명할 것이다.

386줄의 getRegionCode()함수는 4.3.에서 설명한 함수이다. 지역 코드를 API로 불러와 regionCode라는 전역변수에 저장한다.

네이버 지도 출력(389~424)

텍스트이(가) 표시된 사진

자동 생성된 설명

네이버 지도 API를 이용하여 화면상에 지도를 출력하는 부분이다. 396~399줄에서 실질적인 지도 호출이 이루어진다. zoom은 한반도가 한 눈에 보이도록 7로 하였고, 중심좌표는 한반도의 정중앙인 양구의 배꼽마을의 좌표로 하였다.

402~424줄은 지도가 처음 켜진 뒤 발생하는 이벤트에 대해 설정하고 있다. 우선 총 17개의 광역지자체에 대한 지적 정보를 불러와 regionGeoJson에 저장하고, 이것이 모두 완료되면 localGlobal() 함수를 호출하도록 하였다. 즉, 지도가 화면에 출력된 후에 전국단위 재난문자를 호출하는 방식이다.

findCenter() 함수(426~453)

텍스트이(가) 표시된 사진

자동 생성된 설명

center를 이용하여 입력된 지역의 중심좌표를 리턴하는 함수이다. 한편, center의 데이터 중 SIG\_KOR\_NM은 ‘용인시 기흥구’, ‘용인시 수지구’와 같이 행정동이 있는 시의 경우 같은 시+구의 형식으로 저장되어 있다. 하지만 필요한 것은 용인시의 중심 좌표이기 때문에 용인시에 해당하는 좌표들의 x평균값과 y평균값을 리턴 하도록 하였다. 한편, ‘중구’와 같은 이름은 전국에 여러 곳에 존재한다. 따라서 location\_code를 입력으로 받는다. 이 값은 SIG\_CD값이 생성될 때 앞에 붙는 부분으로, 원하는 시의 중구 좌표를 얻을 수 있도록 한다.

makerEvent() 함수(455~490)

텍스트이(가) 표시된 사진

자동 생성된 설명

마커를 클릭하면 테이블에 정보를 입력하고 테이블과 id가 close인 div객체를 화면에 표시하는 함수이다. 마커는 markerList에 저장되어 있으므로 이 리스트에 접근하여 모든 마커에 이벤트를 추가한다.

출력할 문자 정보는 massage에 저장되어 있는데, globalName 또는 localName을 통해 몇 번째 지역인지를 알아낸 후, 이 인덱스를 이용해 massage에 접근하면 현재 클릭한 마커의 지역에 해당하는 메시지들을 얻을 수 있다. 이를 localMassage에 저장한다.

테이블의 제목은 마커가 가지고 있는 title값이 된다.

localMassage의 값들에 접근하면서 생성일시, 수신지역, 내용, 발신처를 테이블에 추가한다.

테이블 구성이 완료되면 테이블이 담긴 div 객체와 id가 close인 div 객체의 스타일의 display값을 inline-block으로 하여 화면에 출력될 수 있도록 한다. 객체들의 위치는 화면의 중앙부에 미리 설정 해 두었으므로 마치 팝업처럼 화면 중앙에 테이블이 출력 된다.

search() 함수(402~510)

텍스트, 스크린샷, 모니터, 노트북이(가) 표시된 사진

자동 생성된 설명

화면의 좌상단에 위치한 검색창에서 정보를 받아와 지역별 검색을 진행하는 함수이다. 494~496줄에서 검색에 필요한 정보를 받아온다. 이때, 만약 선택한 지역이 전국이라면 loadGlobal() 함수를 사용할 것이다. 선택한 지역이 전국이 아닌 값이라면 loadLocal() 함수를 통해 지역별 재난문자 현황을 받아온다.

재난문자현황을 받아 오기 전, 지도의 포커스를 변경한다. 전국이 선택된 경우 최초 화면을 출력한 세팅으로 돌아오고, 그렇지 않다면 regionGeoJson을 이용하여 panToBounds()함수에 넣음으로써 지도를 이동시키게 된다.

토론

요구사항의 완성도

최초 제시한 요구사항 중 기능적 요구사항인 전국 현황 출력, 클릭 시 표 출력, 지역별 현황 출력인 이 3가지는 성공적으로 완성시켰다.

비기능적 요구사항 중 데이터를 받아오는 동안 이에 대한 정보를 안내하는 부분과 마커의 바탕 색을 가중치를 이용하는 부분은 완수하였다.

하지만 화면의 부드러운 이동은 미흡한 부분이 있었다. 지역별 이동시에는 panToBounds() 함수를 이용하기에 부드럽게 이동하지만 전국을 대상으로 이동하면 setCenter() 함수와 setZoom()함수를 사용하기 때문에 순간적으로 바뀐다. 전국 단위의 bound 정보를 얻어 panToBounds()함수를 사용하거나 setCenter() 실행 시 약간의 딜레이를 부여하여 이동에 시간이 소요되도록 하여 문제를 해결할 수 있을 것으로 보인다.

변수와 함수의 통일

코드를 보면, 사실 불필요한 함수와 변수들이 다수 존재한다. 대부분의 변수와 함수는 비슷한 역할을 하지만 대상이 전국인지 지역별인지에 따라 local과 global로 나뉘게 된다. 처음부터 두 방향 모두 고려하여 코드를 작성하였다면 불필요한 함수나 변수를 줄일 수 있었을 것이다. 기능적으로는 크게 문제가 되지 않겠지만 불필요한 메모리를 사용한다는 점과 유지보수에 어려움이 있다는 점에서 개선해야 할 점이라 생각된다.

경상남도 창원시

경상남도를 대상으로 지역별 검색을 수행하면, 창원시의 marker가 제대로 작동하지 않는 문제를 발견하였다. 클릭하면 table은 정상적으로 출력되지만 문자의 개수가 나오지 않으며 바탕색 또한 바뀌지 않았다. 원인을 찾아보니 지역별 코드 API의 결과값에 창원시가 여러 번 중복 되어있다는 것을 알 수 있었다. 창원시가 통합창원시로 개편되면서 지역 정보가 정리되지 않았기 때문이 아닐까 생각된다. 따라서 화면에 마커가 제대로 출력되지 않는 문제는 사실 여러 개의 마커가 겹쳐져 있기 때문에 빈 마커가 정보를 담은 마커를 덮고 있는 상황인 것이다. 따라서 창원시에 대해서만 특별히 오류를 수정하는 special coding이 들어가면 문제를 해결 할 수 있을 것이다.

전체적인 디자인

기능적 요구사항은 대부분 완수하였지만 비기능적 요구사항 중 가장 중요한 요소인 디자인적 문제에서는 상당히 미흡하다. 대부분의 객체들이 div로 이루어져 있고, marker 정도를 제외하면 직사각형 박스로 되어 단순한 형태를 취하고 있다. 인터넷 검색을 통해 쉽게 얻을 수 있는 무료 템플릿을 사용하는 등의 방법으로 디자인을 향상시킬 수 있을 것이다.

결론

요구사항 도출에서 제시한 주요 기능적 요구사항과 시나리오를 모두 완수하였다. 이 프로그램을 이용하여 전국의 재난문자 현황은 물론, 지역별 재난문자 또한 확인할 수 있다. 하지만 몇몇 오류가 존재하며 디자인적으로 미흡한 부분이 존재한다. 또한 코드가 전체적으로 중구난방으로 구성된 ‘더티 코드’이다. 이를 통해 얻게 된 점은 더욱 구체적이고 자세한 설계과정을 통해 함수와 함수간, 나아가서는 변수와 변수 간의 관계를 명확히 하여 불필요한 점을 줄여야 한다. 또한 디자인적 요소 역시 신경 써서 작업해야 할 것이다.

전체 소스코드

<!DOCTYPE html>

<html lang="ko">

<head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>재난문자 발령 현황</title>

    <style>

        body {

            width: 100%;

            height: 100vh;

            margin: 0px;

        }

        #title {

            border: 1px solid black;

            background-color: white;

            display: inline-block;

            position: fixed;

            left: 50%;

            top: 5%;

            transform: translate(-50%, 0%);

            z-index: 1;

        }

        #loading {

            border: 1px solid black;

            background-color: white;

            display: inline-block;

            position: fixed;

            left: 50%;

            top: 10%;

            transform: translate(-50%, 0%);

            z-index: 1;

        }

        #search {

            display: inline-block;

            position: fixed;

            left: 3%;

            top: 15%;

            width: max-content;

            z-index: 1;

        }

        .info {

            border: 1px solid black;

            background-color: white;

            display: inline-block;

            width: 9em;

            text-align: center;

        }

        #submit {

            display: inline-block;

            margin-left: 45%;

        }

        #table {

            background-color: white;

            display: none;

            position: fixed;

            width: 75%;

            height: fit-content;

            max-height: 70%;

            left: 50%;

            top: 50%;

            overflow: scroll;

            transform: translate(-50%, -50%);

            z-index: 2;

        }

        table{

            text-align: center;

            background-color: white;

            width: 100%;

            height: 100%;

            border-bottom: 2px solid black;

            border-collapse: collapse;

        }

        table > caption {

            background-color: #9cd476;

            border-top: 2px solid black;

            border-bottom: 2px solid black;

            font-size: larger;

            font-weight: bold;

        }

        .bgColors:nth-child(even) {

            background-color: #e7e7e7;

        }

        th, td {

            border-collapse: collapse;

            padding: 5px;

        }

        td.date {

            width: 16.75%

        }

        td.location\_name {

            width: 25.43%

        }

        td.msg {

            width: 45.62%

        }

        td.send {

            width: 12.20%

        }

        #close {

            display: none;

            position: fixed;

            left: 50%;

            bottom: 10%;

            transform: translate(-50%, -50%);

            z-index: 3;

        }

        .circle {

            border: 1px solid black;

            width: 30px;

            height: 30px;

            border-radius: 50%;

            background-color: rgb(255,255,255);

            display: table-cell;

            text-align: center;

            vertical-align: middle;

            transform: translate(-50%, -50%);

            z-index: 1;

        }

    </style>

    <!-- naver map -->

    <script type="text/javascript" src="https://openapi.map.naver.com/openapi/v3/maps.js?ncpClientId=7d7py3r4ly"></script>

    <!-- jquery -->

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

    <!-- 시군구 중심좌표 Json -->

    <script type="text/javascript" src="https://grapediget.github.io/project/database/center\_sigungu.json"></script>

    <script>

        var raw\_Massage;

        var massage = [];

        var globalName = ['강원도', '경기도', '경상남도', '경상북도', '광주광역시', '대구광역시', '대전광역시', '부산광역시', '서울특별시', '울산광역시', '인천광역시', '전라남도', '전라북도', '제주특별자치도', '충청남도', '충청북도', '세종특별자치시'];

        var localName = [];

        var regionCode;

        function getRegionCode() {

            var xhr = new XMLHttpRequest();

            var url = 'http://api.odcloud.kr/api/15066113/v1/uddi:b72a7ab6-9c27-44ad-937e-93188c67c53d'; /\*URL\*/

            var queryParams = '?' + encodeURIComponent('serviceKey') + '='+'wXG%2FXtYUrxKSkNDwa0yojj4kt6qQeqxElf8MLbn1SJz1QFS5jexLxeDY%2B%2FqycFa6ROuboaa%2FC91NCvmQBMK%2Fxw%3D%3D'; /\*Service Key\*/

            queryParams += '&' + encodeURIComponent('page') + '=' + encodeURIComponent('1'); /\*\*/

            queryParams += '&' + encodeURIComponent('perPage') + '=' + encodeURIComponent(300); /\*\*/

            queryParams += '&' + encodeURIComponent('returnType') + '=' + encodeURIComponent('xml'); /\*\*/

            xhr.onreadystatechange = function() {

                if (this.readyState == 4 && this.status == 200) {

                    regionCode = JSON.parse(this.response).data;

                }

            };

            xhr.open('GET', url + queryParams);

            xhr.send('');

        }

        function setLoading(state) {

            loading = document.getElementById("loading");

            if(state)

                loading.style.display = "inline-block";

            else

                loading.style.display = "none";

        }

        function loadGlobal(num) {

            setLoading(true);

            var xhr = new XMLHttpRequest();

            var url = 'http://apis.data.go.kr/1741000/DisasterMsg3/getDisasterMsg1List'; /\*URL\*/

            var queryParams = '?' + encodeURIComponent('serviceKey') + '='+'wXG%2FXtYUrxKSkNDwa0yojj4kt6qQeqxElf8MLbn1SJz1QFS5jexLxeDY%2B%2FqycFa6ROuboaa%2FC91NCvmQBMK%2Fxw%3D%3D'; /\*Service Key\*/

            queryParams += '&' + encodeURIComponent('pageNo') + '=' + encodeURIComponent('1'); /\*\*/

            queryParams += '&' + encodeURIComponent('numOfRows') + '=' + encodeURIComponent(num); /\*\*/

            queryParams += '&' + encodeURIComponent('type') + '=' + encodeURIComponent('xml'); /\*\*/

            xhr.onreadystatechange = function() {

                if (this.readyState == 4 && this.status == 200) {

                    raw\_Massage = this.responseXML.getElementsByTagName("row");

                    setLoading(false);

                    makeGlobalMarker();

                    setGlobalNumber();

                    markerEvent(true);

                }

            };

            xhr.open('GET', url + queryParams);

            xhr.send('');

        }

        function makeGlobalMarker() {

            for (var i = 0; i < regionGeoJson.length; i++) {

                region = regionGeoJson[i].features[0];

                position = findCenter(region.properties.area1, NaN)

                var marker = new naver.maps.Marker({

                    position: new naver.maps.LatLng(position[1], position[0]),

                    map: map,

                    title: region.properties.area1,

                    icon: {

                        content: [

                                    '<div class="circle" id=',region.properties.area1,'>',

                                    '</div>'

                                ].join('')

                    }

                });

                markerList[i] = marker;

            }

        }

        function setGlobalNumber() {

            var globalNumber = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

            massage = []

            for(var j = 0; j < globalName.length; j++) {

                massage.push([])

            }

            for(var i=0; i < raw\_Massage.length; i++) {

                var location = raw\_Massage[i].getElementsByTagName("location\_name")[0].innerHTML;

                for(var j = 0; j < globalName.length; j++) {

                    if(location.includes(globalName[j])){

                        globalNumber[j] += 1;

                        massage[j].push(raw\_Massage[i])

                    }

                }

            }

            for(var i = 0; i < globalName.length; i++) {

                document.getElementById(globalName[i]).innerHTML = globalNumber[i]

                if (globalNumber[i] < 50) {

                    red\_blue = 255-globalNumber[i]/50\*255

                    green = 255-globalNumber[i]/50\*128

                }

                else {

                    red\_blue = 0

                    green = 128

                }

                document.getElementById(globalName[i]).style.backgroundColor = 'rgb('+red\_blue+','+green+','+red\_blue+')';

            }

        }

        function loadLocal(location\_name, time, num) {

            setLoading(true)

            var xhr = new XMLHttpRequest();

            var url = 'http://apis.data.go.kr/1741000/DisasterMsg4/getDisasterMsg2List'; /\*URL\*/

            var queryParams = '?' + encodeURIComponent('serviceKey') + '='+'wXG%2FXtYUrxKSkNDwa0yojj4kt6qQeqxElf8MLbn1SJz1QFS5jexLxeDY%2B%2FqycFa6ROuboaa%2FC91NCvmQBMK%2Fxw%3D%3D'; /\*Service Key\*/

            queryParams += '&' + encodeURIComponent('pageNo') + '=' + encodeURIComponent('1'); /\*\*/

            queryParams += '&' + encodeURIComponent('numOfRows') + '=' + encodeURIComponent(num); /\*\*/

            queryParams += '&' + encodeURIComponent('type') + '=' + encodeURIComponent('xml'); /\*\*/

            queryParams += '&' + encodeURIComponent('create\_date') + '=' + encodeURIComponent(time); /\*\*/

            queryParams += '&' + encodeURIComponent('location\_name') + '=' + encodeURIComponent(location\_name); /\*\*/

            xhr.onreadystatechange = function() {

                if (this.readyState == 4 && this.status == 200) {

                    raw\_Massage = this.responseXML.getElementsByTagName("row");

                    setLoading(false);

                    setLocalName(location\_name);

                    makeLocalMarker(location\_name);

                    setLocalNumber();

                    markerEvent(false);

                }

            };

            xhr.open('GET', url + queryParams);

            xhr.send('');

        }

        function setLocalName(location\_name) {

            localName = new Array();

            if(location\_name == '세종특별자치시') {

                localName.push("세종특별자치시")

                return;

            }

            for(var i=0; i < regionCode.length; i++) {

                if(regionCode[i]["발송지역(시도)"] == location\_name) {

                    if(regionCode[i]["발송지역(시군구)"] != '전체')

                        localName.push(regionCode[i]["발송지역(시군구)"]);

                }

            }

        }

        function makeLocalMarker(location\_name) {

            for (var i = 0; i < localName.length; i++) {

                region = localName[i];

                idx = globalName.indexOf(location\_name)

                location\_code = parseInt(regionGeoJson[idx].features[0].properties.admcode) / 100000000

                position = findCenter(region, location\_code);

                var marker = new naver.maps.Marker({

                    position: new naver.maps.LatLng(position[1], position[0]),

                    map: map,

                    title: region,

                    icon: {

                        content: [

                                    '<div class="circle" id=', region,'>',

                                    '</div>'

                                ].join('')

                    }

                });

                markerList[i] = marker;

            }

        }

        function setLocalNumber() {

            var localNumber = [];

            massage = [];

            for(var j = 0; j < localName.length; j++) {

                localNumber.push(0)

                massage.push([])

            }

            for(var i = 0; i < raw\_Massage.length; i++) {

                var location = raw\_Massage[i].getElementsByTagName("location\_name")[0].innerHTML;

                for(var j = 0; j < localName.length; j++) {

                    if(location.includes(localName[j])){

                        localNumber[j] += 1;

                        massage[j].push(raw\_Massage[i])

                    }

                }

            }

            for(var i = 0; i < localName.length; i++) {

                document.getElementById(localName[i]).innerHTML = localNumber[i];

                if (localNumber[i] < 50) {

                    red\_blue = 255-localNumber[i]/50\*255

                    green = 255-localNumber[i]/50\*128

                }

                else {

                    red\_blue = 0

                    green = 128

                }

                document.getElementById(localName[i]).style.backgroundColor = 'rgb('+red\_blue+','+green+','+red\_blue+')';

            }

        }

        function deleteMarker() {

            for(var i = 0; i < markerList.length; i++) {

                var target = document.getElementsByClassName('circle')[0];

                target.parentNode.removeChild(target);

            }

            markerList = [];

        }

    </script>

</head>

<body>

    <div id="title">재난문자 발령 현황</div>

    <div id="loading">재난문자를 검색중입니다.</div>

    <form id="search">

        <div class = "info">검색 지역</div> <select id = 'search\_location' style="width:9em; text-align: center" value="">

            <option value="전국">전국</option>

            <option value="서울특별시">서울특별시</option>

            <option value="부산광역시">부산광역시</option>

            <option value="대구광역시">대구광역시</option>

            <option value="인천광역시">인천광역시</option>

            <option value="광주광역시">광주광역시</option>

            <option value="대전광역시">대전광역시</option>

            <option value="울산광역시">울산광역시</option>

            <option value="세종특별자치시">세종특별자치시</option>

            <option value="경기도">경기도</option>

            <option value="강원도">강원도</option>

            <option value="충청북도">충청북도</option>

            <option value="충청남도">충청남도</option>

            <option value="전라북도">전라북도</option>

            <option value="전라남도">전라남도</option>

            <option value="경상북도">경상북도</option>

            <option value="경상남도">경상남도</option>

            <option value="제주특별자치도">제주특별자치도</option>

        </select>

            <br>

        <div class = "info">기준 시간</div> <input id = 'search\_time' type="text" style="width:9em; text-align: center" value=""> <br>

        <div class = "info">검색 데이터 개수</div> <input id = 'search\_num' type="text" style="width:9em; text-align: center" value="100"> <br>

        <div id = 'submit'><button type="button">검색</button></div>

    </form>

    <div id="map" style="width: 100%; height: 100%; position: relative; overflow: hidden"></div>

    <div id="table"></div>

    <div id="close">

        <form><button type="button">X</button></form>

    </div>

    <script>

        let center = JSON.parse(JSON.stringify(centerCoordinates));

        let now = new Date();

        let create\_date = now.getFullYear()+'/'+now.getMonth()+'/'+now.getDate()+' '+now.getHours()+':'+now.getMinutes()+':'+now.getSeconds();

        document.getElementById("search\_time").value = create\_date;

        document.getElementById("close").addEventListener("click", function() {

            this.style.display = "none";

            document.getElementById("table").style.display = "none";

        })

        document.getElementById("submit").addEventListener("click", function() {

            search();

        })

        getRegionCode();

    </script>

    <script>

        var HOME\_PATH = "https://navermaps.github.io/maps.js/docs",

            urlPrefix = HOME\_PATH + "/data/region",

            urlSuffix = ".json",

            regionGeoJson = [],

            loadCount = 0,

            markerList = [];

        var map = new naver.maps.Map(document.getElementById("map"), {

            zoom: 7,

            mapTypeId: "normal",

            center: new naver.maps.LatLng(36.53375, 128.05025),

        });

        naver.maps.Event.once(map, 'init\_stylemap', function () {

            for (var i = 1; i < 18; i++) {

                var keyword = i +'';

                if (keyword.length === 1) {

                    keyword = '0'+ keyword;

                }

                $.ajax({

                    url: urlPrefix + keyword + urlSuffix,

                    success: function(idx) {

                        return function(geojson) {

                            regionGeoJson[idx] = geojson;

                            loadCount++;

                            if (loadCount === 17) {

                                loadGlobal(100);

                            }

                        }

                    }(i - 1)

                });

            }

        });

        function findCenter(region\_name, location\_code) {

            var coordinates = [];

            if(isFinite(location\_code))

                center.features.forEach(function(reg) {

                    if (reg.SIG\_KOR\_NM.includes(region\_name)) {

                        if (parseInt(reg.SIG\_CD / 1000) == location\_code) {

                            coordinates.push([reg.X, reg.Y]);

                        }

                    }

                });

            else

                center.features.forEach(function(reg) {

                    if (reg.SIG\_KOR\_NM.includes(region\_name)) {

                        coordinates.push([reg.X, reg.Y]);

                    }

                });

            if(coordinates.length == 1)

                return coordinates[0];

            else {

                var sum\_x = 0,

                    sum\_y = 0;

                for(var i = 0; i < coordinates.length; i++) {

                    sum\_x += coordinates[i][0];

                    sum\_y += coordinates[i][1];

                }

                return [sum\_x/coordinates.length, sum\_y/coordinates.length]

            }

        }

        function markerEvent(region) {

            markerList.forEach(function(marker) {

                marker.addListener("click", function(e) {

                    if(region)

                        var localMassage = massage[globalName.indexOf(marker.title)];

                    else

                        var localMassage = massage[localName.indexOf(marker.title)];

                    table = document.getElementById("table");

                    var content = ['<table>',

                        '<caption>', marker.title, '</caption>',

                        '<thead class="bgColors">',

                            '<tr><th>생성일시</th><th>수신지역</th><th>내용</th><th>발신처</th></tr>',

                        '</thead>',

                        '<tbody>'];

                    for (var j = 0; j < localMassage.length; j++) {

                        let location\_name = localMassage[j].getElementsByTagName("location\_name")[0].innerHTML;

                        if(location\_name.length > 30) {

                            location\_name = location\_name.slice(0, 40) + '...'

                        }

                        content.push(['<tr class="bgColors"><td class="date">',

                            localMassage[j].getElementsByTagName("create\_date")[0].innerHTML,

                            '</td><td class="location\_name">',

                            location\_name,

                            '</td><td class="msg">',

                            localMassage[j].getElementsByTagName("msg")[0].innerHTML,

                            '</td><td class="send">',

                            localMassage[j].getElementsByTagName("send\_platform")[0].innerHTML,

                            '</td></tr>'].join(''))

                    }

                    content.push('</tbody></table>');

                    table.innerHTML = content.join('');

                    table.style.display = 'inline-block';

                    document.getElementById("close").style.display = "inline-block";

                });

            });

        }

        function search() {

            deleteMarker();

            var location = document.getElementById("search\_location").value;

            var time = document.getElementById("search\_time").value;

            var num = document.getElementById("search\_num").value;

            if(location == "전국") {

                map.setCenter(new naver.maps.LatLng(36.53375, 128.05025));

                map.setZoom(7);

                makeGlobalMarker();

                loadGlobal(num);

            }

            var feature = regionGeoJson[globalName.indexOf(location)].features[0];

            var minPoint = new naver.maps.Point(feature.bbox[0], feature.bbox[1]);

            var maxPoint = new naver.maps.Point(feature.bbox[2], feature.bbox[3]);

            var bound = new naver.maps.PointBounds(minPoint, maxPoint);

            map.panToBounds(bound);

            loadLocal(location, time, num);

        }

    </script>

</body>

</html>

참고자료

1. “행정안전부\_재난문자방송 발령현황(지역별)”, 공공데이터포털,

https://www.data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=15091495

2. “행정안전부\_재난문자방송 발령현황”, 공공데이터포털,

https://www.data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=3058822

3. “행정안전부\_재난문자 발령현황 API 지역코드”, 공공데이터포털,

https://www.data.go.kr/tcs/dss/selectFileDataDetailView.do?publicDataPk=15066113#tab-layer-openapi

4. “NAVER 지도 API v3”, NAVER CLOUD FATFORM,

https://navermaps.github.io/maps.js.ncp/docs/index.html

5. “시군구 중심 좌표 추출”, 브런치, https://brunch.co.kr/@data/29

6. “CSS 원하는 순서 및 홀수, 짝수에만 스타일”, IOTINFRA’S BLOG,

http://blog.iotinfra.net/?p=2213