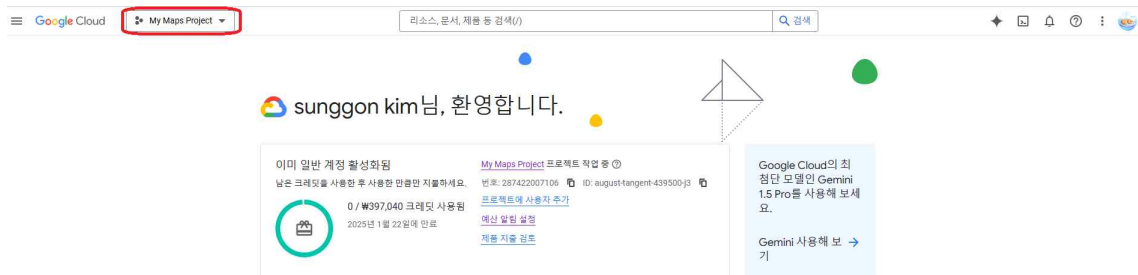
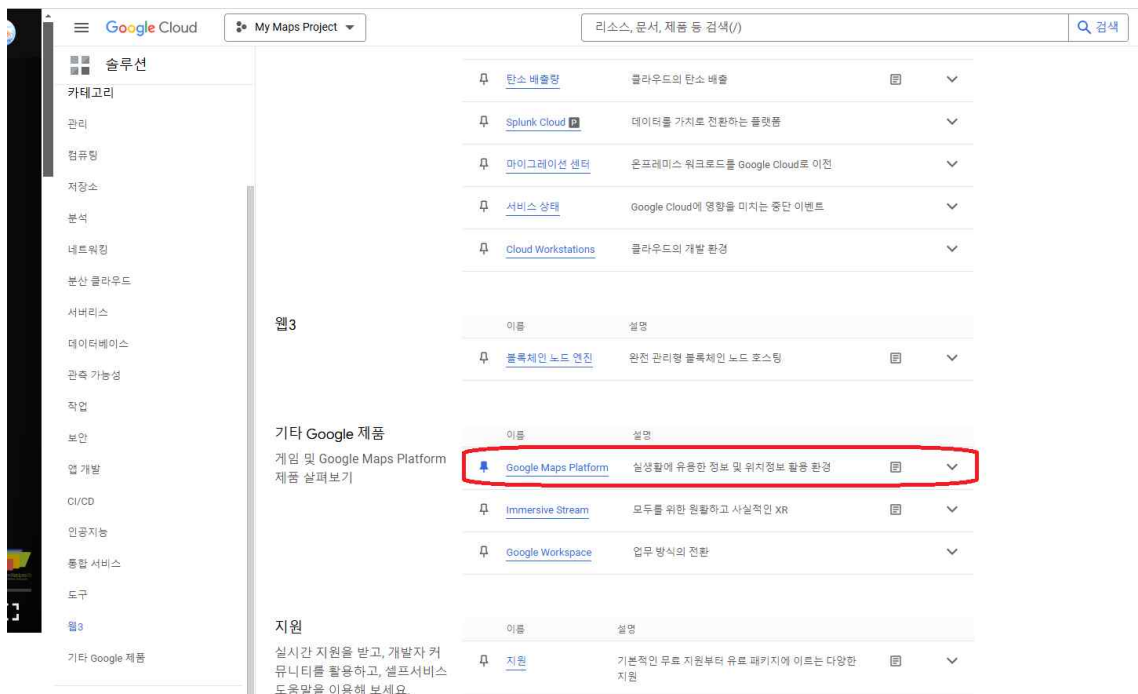


Google Maps Platform 설정

1. google cloud console 검색
2. 사용자 등록 (카드도 필요할지 모름?) / 등록한지 오래되어서...
 - 6개월 프리 / 사용량에 따라 과금
3. 사용자 등록후 프로젝트 목록 클릭



4. 새 프로젝트 생성
5. 프로젝트가 선정된 상태에서... 햄버그 메뉴에서 모든제품보기 선택
6. Google Maps Platform 선택



7. API 및 서비스 선택

Google Cloud

My Maps Project

리소스, 문서, 제품 등 검색(/)

Google Maps Platform

개요

API 및 서비스

측정 항목

발달량

키 및 사용자 인증 정보

지원

솔루션 라이브러리

지도 관리

지도 스타일

데이터 세트

시작하기, sunggon

Google Maps Platform으로 탐색, 학습, 빌드

지도 생성 핵심 요소 알아보기

필과 모바일을 위한 가장 일반적인 API 및 SDK 사용에 대한 단계별 안내

프로젝트 작성

API 사용 설정

API 디리토리

더 보기

환경 맞춤 설정

몇 가지 질문에 답하여 프로젝트에 맞는 유용한 추천을 받으세요.

프로젝트 작성

API 및 서비스

알아보

적합한 지도 제품 찾기

지도와 위치 기반 환경을 변환하는 데 필요한 도구를 살펴보세요.

이 페이지가 도움이 되었나요?

FILTER: Maps Places Routes Environment Enabled Disabled

Air Quality API

ENABLE

Provides air quality data for a specific location with a resolution of 500 x 500 meters.

Environment

Guides

Pollen API

ENABLE

Provides pollen data for a specific location with a resolution of 1x1km

Environment

Guides

Solar API

ENABLE

Advanced imagery and insights to create solar proposals and designs.

Environment

Guides

Aerial View API

ENABLE

Get 3D cinematic videos of places

Maps

Guides

Map Tiles API

ENABLE

2D, 3D and Street View tiles for building immersive visualizations.

Maps

Guides

Maps Datasets API

ENABLE

Use your own geospatial data with Google Maps Platform APIs

Maps

Guides

Maps Elevation API

ENABLE

Elevation data for any point in the world.

Maps

Guides

Maps Embed API

ENABLE

Make places easily discoverable with interactive Google Maps.

Maps

Guides

Maps JavaScript API

DISABLE

Maps for your website

Maps

Keys Metrics Guides

8. 왼쪽 메뉴의 키 및 사용자 인증 정보 클릭

모든 사용자 인증 정보를 보려면 API 및 서비스의 사용자 인증 정보로 이동하세요.

애플리케이션에 대한 정보를 포함하여 OAuth 동의 화면을 구성해야 합니다.

동의 화면 구성

API 키

이름	생성일	제한사항	작업
Maps Platform API Key	2024. 10. 23.	-	키 표시

OAuth 2.0 클라이언트 ID

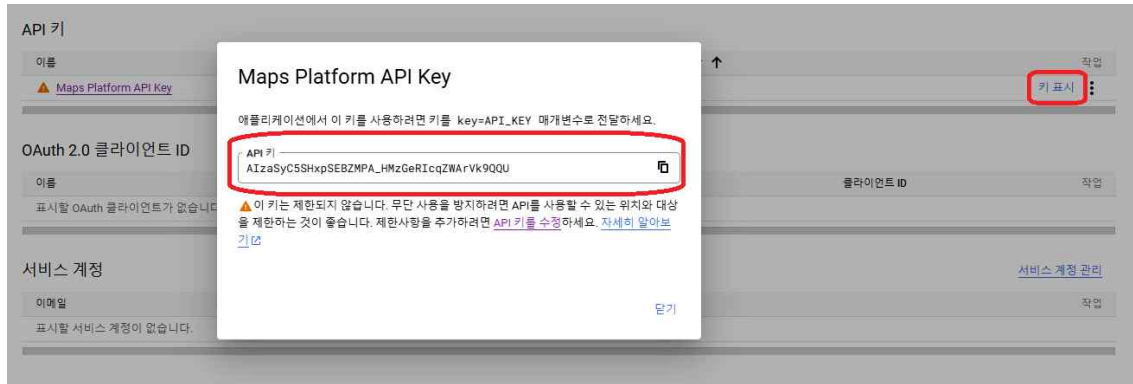
이름	생성일	유형	클라이언트 ID	작업
표시할 OAuth 클라이언트가 없습니다.				

서비스 계정

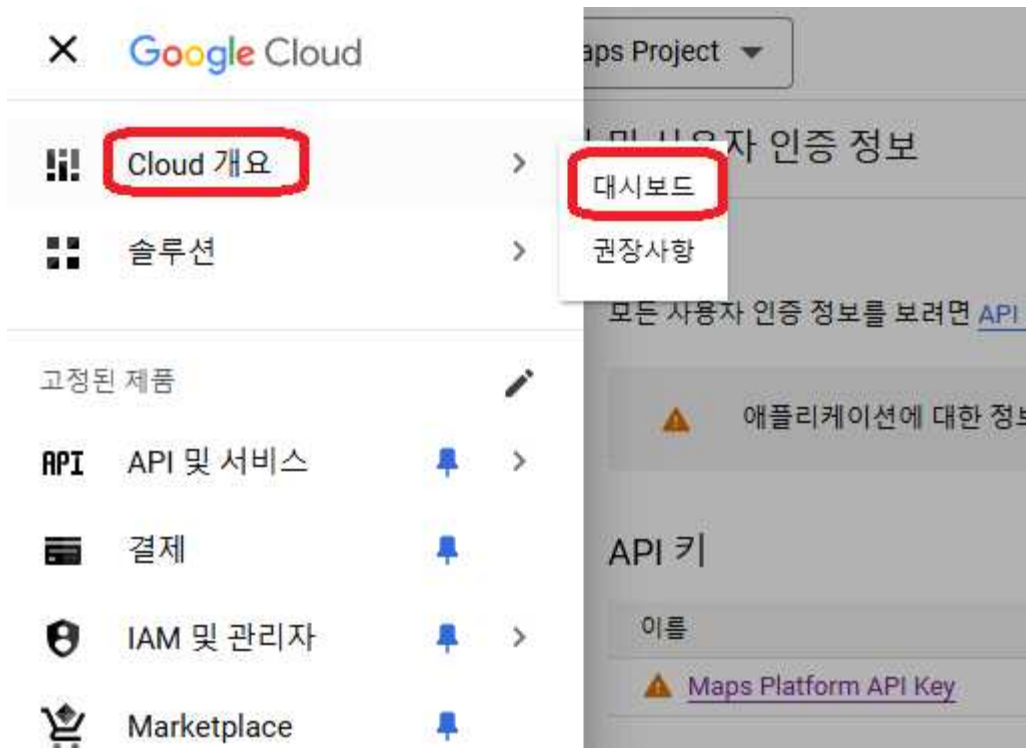
이메일	이름	작업
표시할 서비스 계정이 없습니다.		

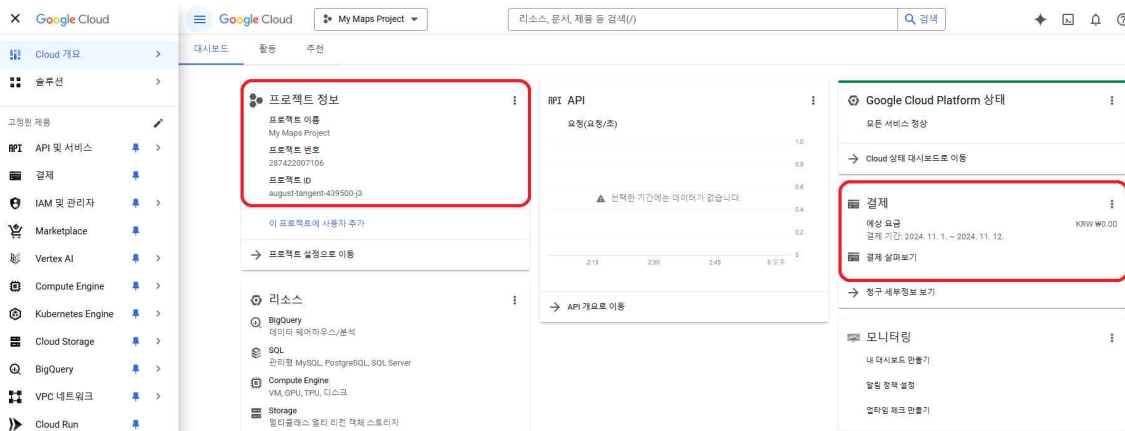
서비스 계정 관리

9. 키 표시를 클릭하면 키 값이 표시된다. 키 값을 잘 복사해서 보관한다.



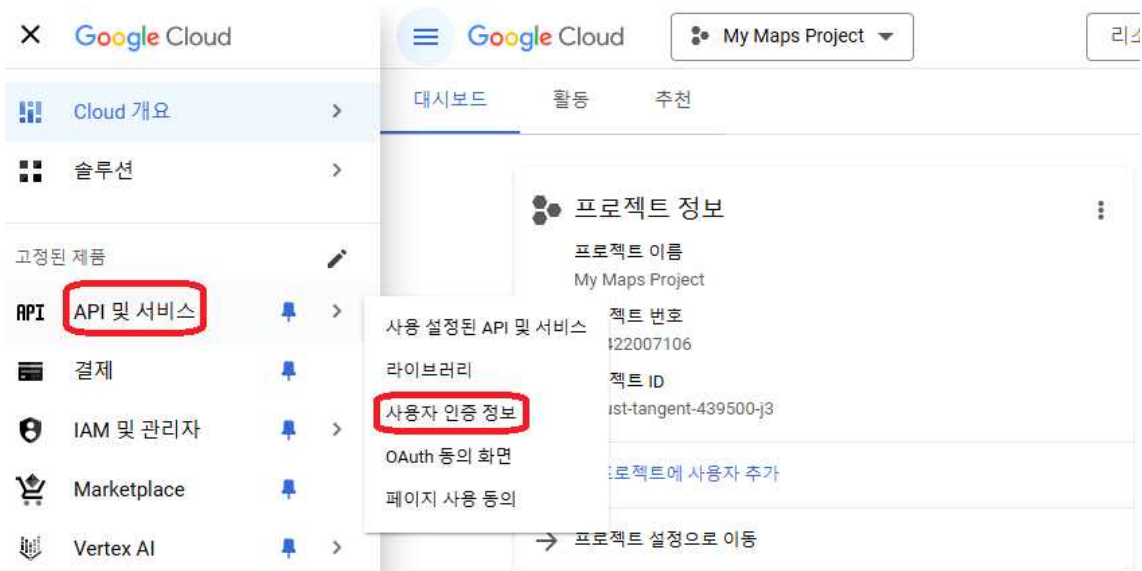
10. 햄버그 메뉴에서 대시보드를 클릭한다.





사용량은 api키를 통해 집계가 되며...
따라서 누출에 주의할 필요가 있고
제한사항의 설정이 필요할 수도 있다.

11. 제한 사항 정보



API 키

<input type="checkbox"/>	이름
<input type="checkbox"/>	⚠ Maps Platform API Key

OAuth 2.0 클라이언트 ID

<input type="checkbox"/>	이름	생성일 ↓
표시할 OAuth 클라이언트가 없습니다.		

이름 *

Maps Platform API Key

키 제한사항



이 키는 제한되지 않습니다. 무단 사용을 방지하려면 API를 사용할 수 있는 위치와 대상을 제한하는 것이 좋습니다. [자세히 알아보기](#)

애플리케이션 제한사항 설정

애플리케이션 제한사항은 API 키 사용을 특정 웹사이트, IP 주소, Android 애플리케이션 또는 iOS 애플리케이션으로 제한합니다. 키별로 애플리케이션 제한사항 1개를 설정할 수 있습니다.

☒ 없음

☐ 웹사이트

☐ IP 주소

☐ Android 앱

☐ iOS 앱

API 제한사항

API 제한사항은 이 키를 호출할 수 있는 사용 설정된 API를 지정합니다.

☒ 키 제한 안함

이 키는 모든 API를 호출할 수 있습니다.

☐ 키 제한

참고: 설정이 적용되는 데 최대 5분이 걸릴 수 있습니다.

저장

취소

웹사이트 제한사항

키 사용량 요청을 지정된 웹사이트로 제한합니다.



비워 두면 API 키가 모든 웹사이트의 요청을 수락합니다.

+ ADD

필터 속성 이름 또는 값 입력



상태


웹사이트

수정

표시할 행이 없습니다.

웹사이트 제한사항

키 사용량 요청을 지정된 웹사이트로 제한합니다.

 비워 두면 API 키가 모든 웹사이트의 요청을 수락합니다.

웹사이트 추가

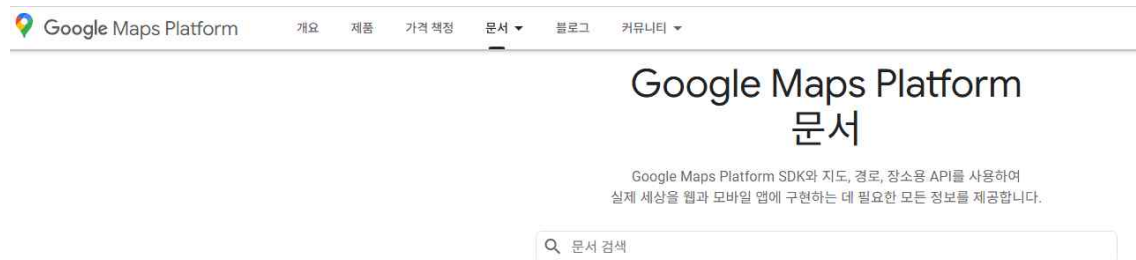
웹사이트 *

[취소](#) [완료](#)

구글 맵 실습

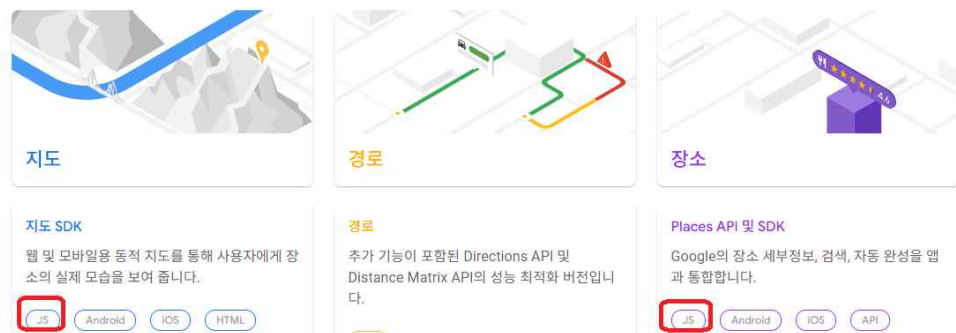
구글 : Google Maps Platform docs 검색

<https://developers.google.com/maps/documentation?hl=ko>

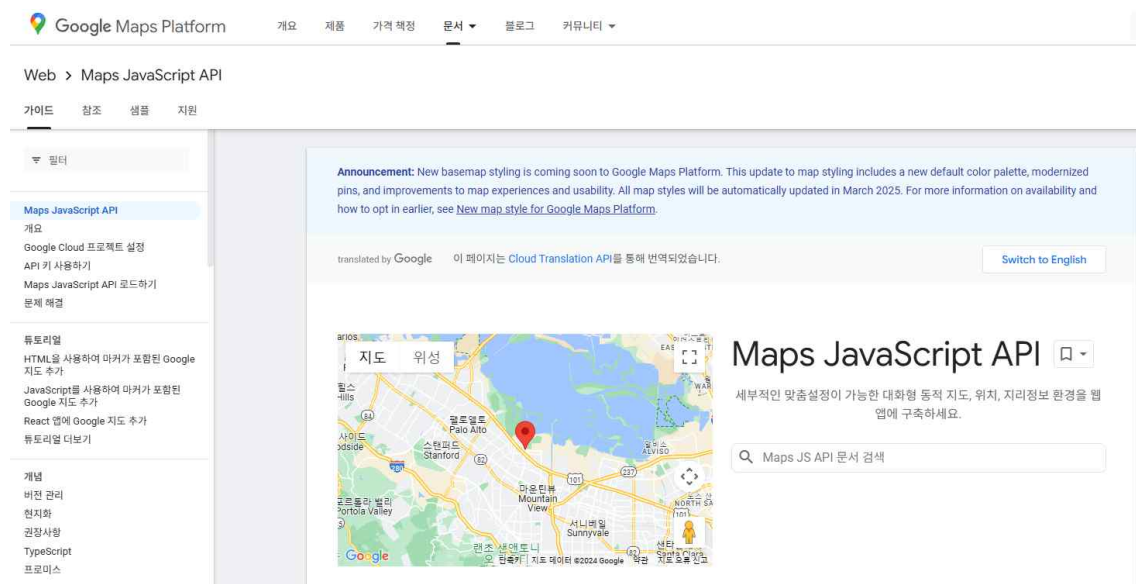


SDK 및 API

포괄적인 문서, 샘플 코드, 예시 앱 등을 살펴보고 학습하며 빌드하세요.

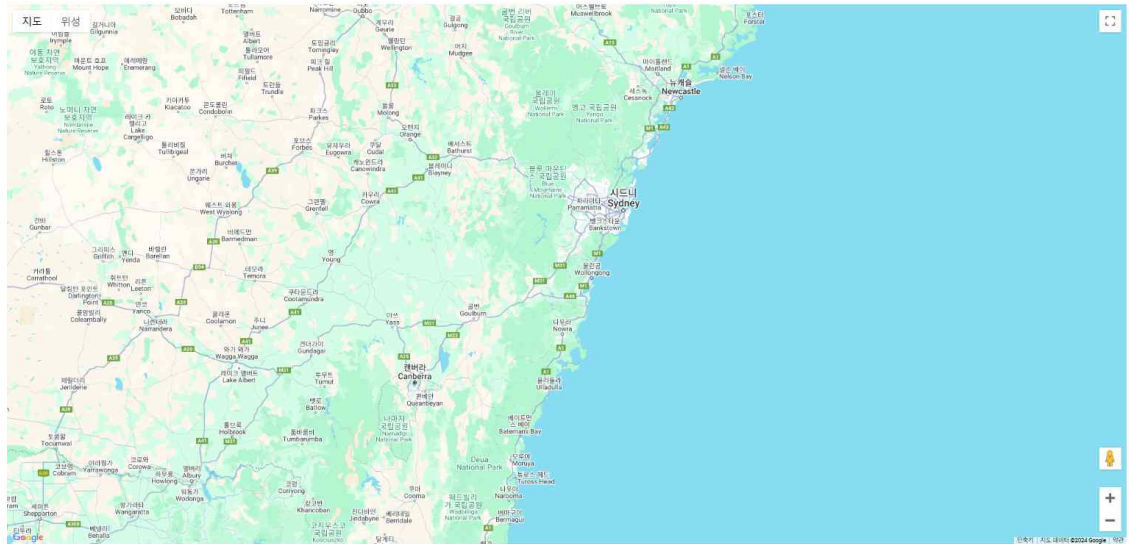


JS를 클릭하면...



간단한 지도 표시하기

Google Maps Platform : JS



```
<!DOCTYPE html>
<html lang="kr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Google Maps Platform : JS</title>
  <link rel="stylesheet" href="/style.css">
  <script src="/google-maps-api-loader.js"> </script>
  <script src="/script.js" defer> </script>
</head>
<body>
  <h1>Google Maps Platform : JS</h1>

  <div id="map"></div>
</body>
</html>
```

script async 와 defer의 차이점

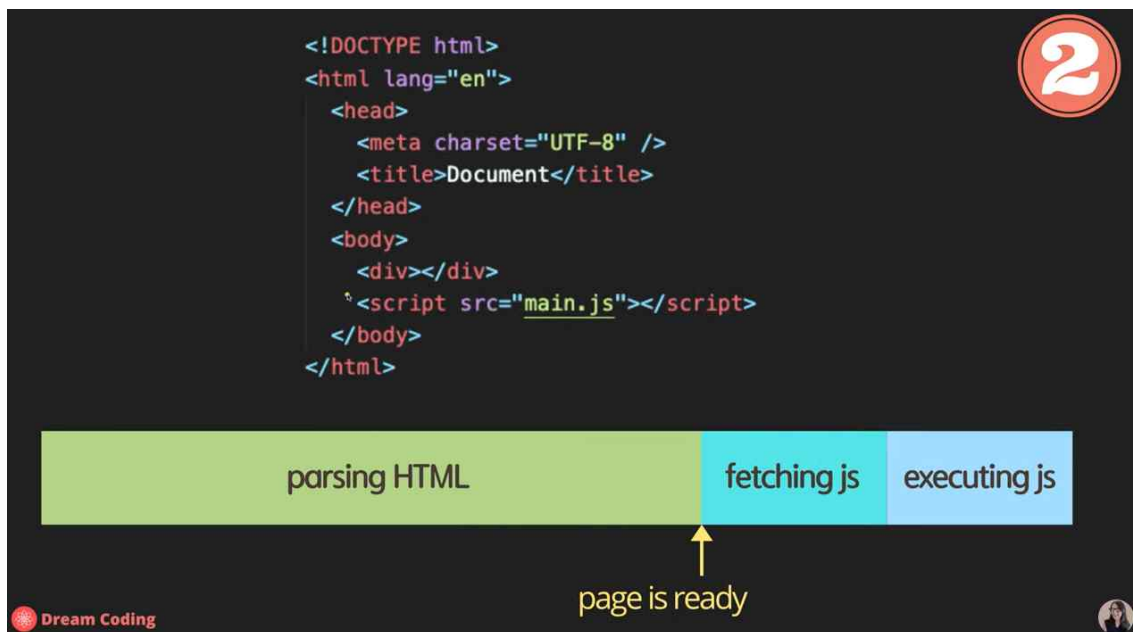


<script>의 위치 확인

DOM에 대한 내용은 설명 생략...

문서 순서대로 해석하다가 main.js를 가져오고 실행하고

단점은?



<script>의 위치는 문서 마지막에 있음

단점은?

웹페이지가 자바스크립트에 크게 의존적인 경우 완전한 화면을 보기까지 시간이 걸린다.

head + async

3

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Document</title>
    <script async src="main.js"></script>
  </head>
  <body>
    <div></div>
  </body>
</html>
```

parsing HTML

blocked

parsing HTML

fetching js

executing js

page is ready

<script>의 위치는 <head>이고 async 속성이 있음

단점은?

문서를 읽다가 js파일을 만나면 html 문서를 읽어 오는것과 js파일을 가져오는 것을 병렬로 처리한다. js문서를 읽어온 후 바로 실행하므로, 문서 뒤쪽의 html을 조작하는 내용이 있을 때 문제가 발생할 수 있다.

head + async

```
<head>
  <meta charset="UTF-8" />
  <title>Document</title>
  <script async src="a.js"></script>
  <script async src="b.js"></script>
  <script async src="c.js"></script>
</head>
```

parsing HTML

blocked

blocked

blocked

parsing HTML

fetching a.js

executing a.js

fetching b.js

executing b.js

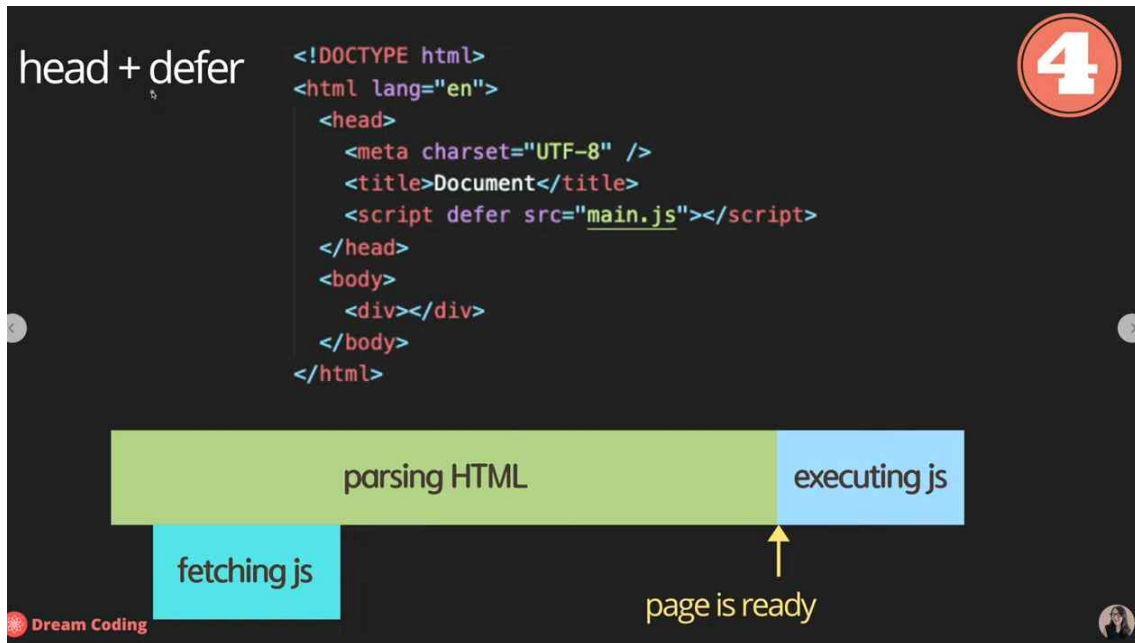
fetching c.js

executing c.js

page is ready

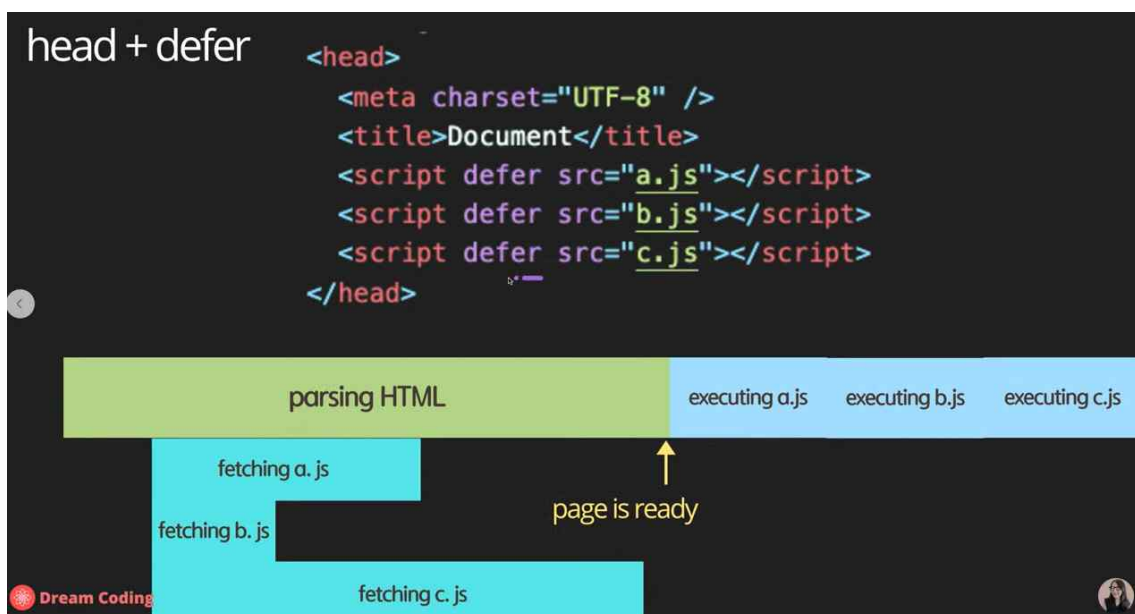
여러개의 js파일이 있을 경우에는 문서의 순서가 아니라 다운로드된 순서대로 실행한다
만약 스크립트가 서로 의존적인 경우에는 문제 발생

가장 좋은 옵션^^



html 문서와 병렬로 가져오는 작업을 하지만 실행은 html 문서를 다 읽어온 후 한다

여러모로 가장 효율적^^



```
/*
 * Always set the map height explicitly to define the size of the div element
 * that contains the map.
 */

#map {
    width: 80%;
    height: 80%;
}

/*
 * Optional: Makes the sample page fill the window.
 */
html,
body {
    height: 100%;
    margin: 0;
    padding: 0;
}
```

```
// script.js

let map;

async function initMap() {
  const { Map } = await google.maps.importLibrary("maps");

  map = new Map(document.getElementById("map"), {
    center: { lat: 35.218973, lng: 129.019035}, //부산폴리텍대학
    zoom: 16,
  });
}

initMap();
```

요약

이 코드는 HTML에서 `id` 가 `"map"` 인 요소에 Google Maps를 표시합니다:

- `Map` 라이브러리를 불러와, 특정 좌표와 줌 수준으로 초기화된 지도를 생성합니다.
- 비동기 방식으로 Google Maps 라이브러리를 로드하여, 로드가 끝나면 지도를 화면에 표시합니다.

이 코드는 Google Maps를 웹페이지에 표시하기 위해 작성된 간단한 초기화 코드입니다. 각 줄을 설명해드릴게요.

코드 설명

1. 전역 변수 `map` 선언:

```
javascript

let map;
```

 코드 복사

`map` 변수를 전역으로 선언하여, 이후에 지도 객체를 이 변수에 저장하고 필요에 따라 다른 함수에서도 접근할 수 있게 합니다.

2. `initMap` 함수 정의:

javascript

 코드 복사

```
async function initMap() { ... }
```

`initMap`이라는 비동기 함수를 정의합니다. 이 함수가 실행되면 Google Maps가 초기화됩니다.

3. Google Maps 라이브러리 로드:

javascript

 코드 복사

```
const { Map } = await google.maps.importLibrary("maps");
```

Google Maps에서 `Map` 클래스를 가져와 사용할 준비를 합니다. 이 과정이 비동기(`await`)로 실행되므로, 라이브러리가 완전히 불러와질 때까지 기다립니다.

이 코드는 Google Maps의 지도 기능을 불러오기 위한 과정입니다.

1. `google.maps.importLibrary("maps")`:

- `google.maps.importLibrary("maps")`는 Google Maps에서 "maps"라는 라이브러리, 즉 지도 관련 기능을 불러오도록 요청하는 코드입니다.
- 이 코드는 비동기 방식이기 때문에 `await`를 사용하여 라이브러리가 다 불러와질 때까지 기다립니다.

2. `const { Map }`:

- 불러온 `maps` 라이브러리에서 `Map`이라는 기능만 가져옵니다.
- `Map`은 지도 객체를 만들고 관리하는 데 필요한 기능을 제공하는 Google Maps의 주요 클래스입니다.

3. 최종 결과:

- 이제 `Map`을 사용할 준비가 되었으므로, 다음 단계에서 지도를 화면에 표시할 수 있습니다.

요약

이 한 줄의 코드는 Google Maps의 지도 표시 기능을 불러오는 것이며, 불러온 후에는 `Map`을 사용하여 실제 지도를 웹페이지에 그릴 수 있습니다.

변수선언에 {}가 들어간 이유?

{ }를 사용하는 이유는 **구조 분해 할당**(destructuring assignment)이라는 자바스크립트 문법을 사용했기 때문입니다. 이 문법을 사용하면 객체에서 특정 속성만 뽑아낼 수 있습니다.

javascript

 코드 복사

```
const { Map } = await google.maps.importLibrary("maps");
```

위 코드에서는 `google.maps.importLibrary("maps")`가 여러 속성을 가진 객체를 반환할 수 있는데, 그중에서 우리는 `Map` 속성만 필요로 하기 때문에 `{ Map }`으로 `Map` 속성만 가져와 `Map`이라는 변수로 사용할 수 있습니다.

구조 분해 할당을 쓰는 이유

1. 코드 간결성: `google.maps.importLibrary("maps")`에서 필요한 `Map`만 뽑아서 `Map` 변수에 담아, 코드를 더 간단하게 만들 수 있습니다.
2. 필요한 것만 가져오기: `Map` 외에 다른 기능이 필요 없다면, 전체 객체를 가져오는 대신 필요한 속성만 가져올 수 있습니다.

예시

예를 들어, 일반적인 객체 할당에서:

javascript

 코드 복사

```
const person = { name: "Alice", age: 25 };
const { name } = person;
console.log(name); // "Alice"
```

이와 같이, `person` 객체에서 `name` 속성만 뽑아 변수로 사용할 수 있습니다.

4. 지도 생성 및 설정:

javascript

코드 복사

```
map = new Map(document.getElementById("map"), {  
  center: { lat: -34.397, lng: 150.644 },  
  zoom: 8,  
});
```

- `new Map(...)` 을 통해 새로운 지도를 생성하고, `map` 변수에 저장합니다.
- `document.getElementById("map")` : HTML에서 `id` 가 `"map"` 인 요소를 찾아 그곳에 지도를 표시합니다.
- `center` : 지도 중심 좌표를 `{ lat: -34.397, lng: 150.644 }` 로 설정합니다.
- `zoom` : 초기 확대 수준을 `8` 로 설정해, 적절한 지역을 보여줍니다.

5. `initMap` 함수 실행:

javascript

코드 복사

```
initMap();
```

정의된 `initMap` 함수를 호출해 Google Maps를 화면에 나타나게 합니다.

```
// google-maps-api-loader.js

const API_KEY = 'AIzaSyC5SHxpSEBZMPA_HMzGeRlcqZWARVk9QQU';
const REGION = 'KR';
const LANG = 'ko';

(g => {
  var h,
    a,
    k,
    p = "The Google Maps JavaScript API",
    c = "google",
    l = "importLibrary",
    q = "__ib__",
    m = document,
    b = window;
  b = b[c] || (b[c] = {});

  var d = b.maps || (b.maps = {}),
    r = new Set,
    e = new URLSearchParams,
    u = () => h || (h = new Promise(async (f, n) => {
      await (a = m.createElement("script")); e.set("libraries", [...r] + "");
      for (k in g) e.set(k.replace(/[A-Z]/g, t => "_" + t.toLowerCase()), g[k]);
      e.set("callback", c + ".maps." + q);
      a.src = `https://maps.${c}apis.com/maps/api/js?` + e; d[q] = f;
      a.onerror = () => h = n(Error(p + " could not load."));
      a.nonce = m.querySelector("script[nonce]")?.nonce || ""; m.head.append(a)
    }));

  d[l] ? console.warn(p + " only loads once. Ignoring:", g) : d[l] = (f, ...n) => r.add(f) && u().then(() => d[l](f, ...n))
})(
  {
    key: API_KEY,
    v: "weekly",
    // Use the 'v' parameter to indicate the version to use (weekly, beta, alpha, etc.).
    // Add other bootstrap parameters as needed, using camel case.
    region: REGION,
    language: LANG
  }
);
```

코드를 별도의 js 파일로 분리한 이유는 다음과 같습니다:

1. 코드의 재사용성: 여러 페이지에서 Google Maps API를 사용할 때 동일한 코드가 필요하므로, 파일로 분리해두면 반복적인 코드 작성 없이 다른 페이지에서 쉽게 불러와 재사용할 수 있습니다.
2. 코드 관리의 편리성: Google Maps API 설정이나 초기화 코드가 독립적으로 관리됩니다. 따라서 API 키를 변경하거나 다른 설정을 조정할 때 이 파일만 수정하면 됩니다.

3. 보안 및 환경 변수 관리: API_KEY와 같은 민감한 정보를 포함하는 코드를 분리해두면 환경에 맞게 보안 처리를 하기가 수월합니다. 예를 들어, .env 파일이나 환경 변수에서 API 키를 관리하도록 쉽게 변경할 수 있습니다.

4. 웹페이지 로딩 최적화: 필요할 때만 이 파일을 로드하여 페이지 로딩 시간을 최적화할 수 있습니다.

이 코드의 요약:

- Google Maps API를 비동기로 불러와, 준비가 완료되면 `google.maps` 네임스페이스에 접근할 수 있도록 해줍니다.
- 이 코드로 여러 페이지에서 일관된 방식으로 Google Maps API를 호출할 수 있고, 페이지 로딩 속도에 영향 없이 필요한 라이브러리만 불러올 수 있습니다.

이 코드는 Google Maps JavaScript API를 웹페이지에 불러오기 위한 코드입니다. 각 부분을 쉽게 설명하자면 다음과 같습니다:

1. API 키 설정:

```
javascript
const API_KEY = 'YOUR_API_KEY';
```

코드 복사

여기서 `API_KEY` 는 Google Maps API에 접근하기 위한 열쇠 같은 역할을 합니다. 이 값은 Google Cloud에서 발급받으며, 개인 키로 대체해야 합니다.

2. 즉시 실행 함수 (IIFE):

```
javascript
(g => { ... })(config);
```

코드 복사

코드는 `g` 라는 변수에 `config` 객체를 전달하여, 내부에서 `API_KEY` 와 `v` (버전 정보)와 같은 설정 정보를 쉽게 사용할 수 있게 합니다.

3. Google 객체 초기화:

```
javascript
b = window;
b = b[c] || (b[c] = {});
```

코드 복사

`window` 객체에 `google` 이라는 네임스페이스를 만들어, `google.maps` 에 접근할 수 있도록 합니다.

4. 비동기 API 로드:

javascript

코드 복사

```
var d = b.maps || (b.maps = {});  
var h = new Promise(...);
```

`h` 라는 Promise 객체를 만들어, 스크립트가 불러와지는 과정이 성공하면 완료(`resolve`) 처리하고, 실패하면 오류(`reject`) 처리하도록 합니다. 이 부분이 비동기적으로 작동해 페이지가 로딩을 기다리지 않고 API가 준비되었을 때 알려줍니다.

5. 라이브러리 설정 및 로드:

javascript

코드 복사

```
e.set("libraries", [...r] + "");
```

`libraries` 라는 매개변수는 Google Maps의 다양한 기능들(library)을 미리 불러오는 설정입니다. 예를 들어, 'places', 'geometry' 등의 추가 기능을 불러올 수 있습니다.

6. 콜백 함수와 스크립트 추가:

javascript

코드 복사

```
e.set("callback", c + ".maps." + q);  
a.src = `https://maps.${c}apis.com/maps/api/js?` + e;
```

여기서 `callback` 설정은 Google Maps API가 준비되었을 때 호출할 함수 이름을 지정합니다. 이 스크립트를 HTML 문서의 `<head>`에 추가하여 Google Maps 스크립트를 실행할 수 있게 합니다.

7. 오류 처리:

javascript

코드 복사

```
a.onerror = () => h = n(Error(p + " could not load."));
```

스크립트가 제대로 불러와지지 않을 때 오류 메시지를 표시하고, Promise가 거부(`reject`)됩니다.

`google-maps-api-loader.js` 파일에서 `region` 과 `language` 옵션을 설정하는 이유는 사용자 경험을 향상시키기 위해 지도에서 제공되는 정보와 서비스가 특정 지역과 언어에 맞게 표시되도록 하는 것입니다.

`region: REGION`

- **설정 목적:** `region` 옵션은 지도가 어떤 지역을 우선적으로 사용할지 설정합니다.
- **영향:**
 - **검색 결과:** 특정 지역을 지정하면 검색할 때 해당 지역을 우선적으로 고려해 결과를 반환합니다. 예를 들어, `KR` (한국)로 설정하면 한국 지역에 맞는 장소 이름이나 정보를 더 정확하게 제공하게 됩니다.
 - **지도 스타일:** 일부 지역 설정에 따라 지도 스타일이나 특정 위치 정보가 변경될 수 있습니다.

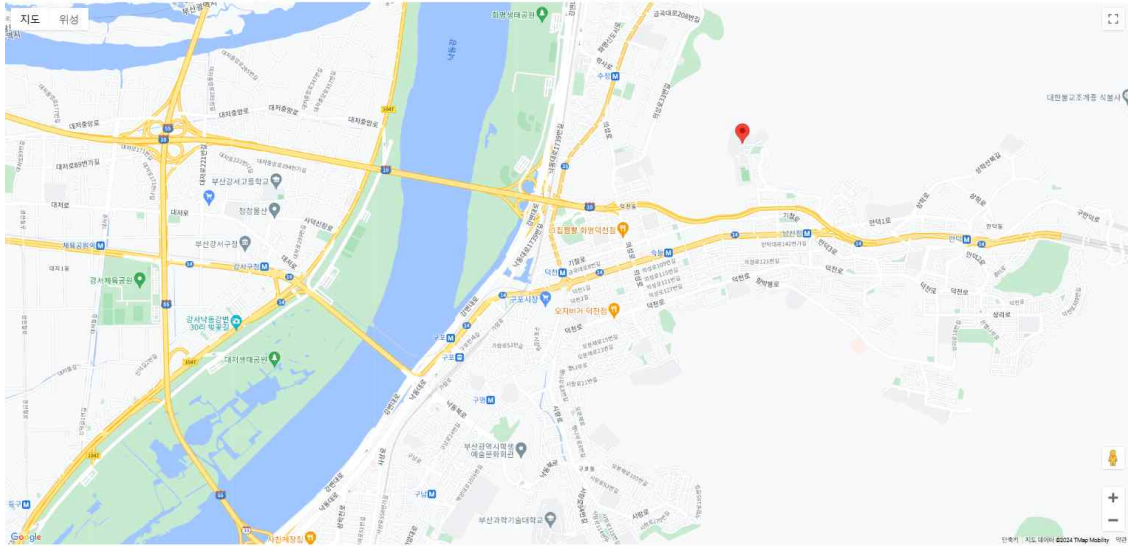
`language: LANG`

- **설정 목적:** `language` 옵션은 지도에서 제공되는 모든 텍스트, 도로명, 장소 이름 등이 지정된 언어로 표시되도록 설정합니다.
- **영향:**
 - **언어:** `ko` (한국어)로 설정하면 지도에 표시되는 텍스트가 한국어로 제공됩니다.
 - **사용자 편의성:** 사용자가 이해하기 쉬운 언어로 정보를 제공하여 사용자 경험을 개선합니다.

예를 들어, `region: "KR"` 과 `language: "ko"` 로 설정하면, Google Maps API는 한국 사용자를 대상으로 지역 및 언어에 맞는 정보가 우선적으로 제공됩니다.

지도에 마커 추가하기

Google Maps Platform : JS



```
let map;
```

```
async function initMap() {
```

```
  // Request needed libraries.
```

```
  const { Map } = await google.maps.importLibrary("maps");
```

```
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");
```

```
  const map = new Map(document.getElementById("map"), {
```

```
    center: { lat: 35.210794, lng: 129.005445 }, // 덕천역(지도중심 설정)
```

```
    zoom: 15,
```

```
    mapId: "4504f8b37365c3d0",
```

```
  });
```

```
  const marker = new AdvancedMarkerElement({
```

```
    map,
```

```
    position: { lat: 35.219016, lng: 129.019002 }, // 부산폴리텍대학(마커 표시)
```

```
  });
```

```
}
```

```
initMap();
```

코드에서 사용된 `mapId` 는 Google Maps에서 특정 스타일의 지도를 불러오기 위해 설정된 고유 ID입니다. 이 `mapId` 는 사용자가 Google Cloud Console에서 미리 정의한 지도 스타일에 대한 식별자입니다.

mapId의 역할

- `mapId: "4504f8b37365c3d0"` 는 Google Maps 플랫폼에서 사용자가 맞춤 설정한 스타일을 불러오기 위해 사용하는 값입니다.
- Google Maps Console에서 특정 테마나 색상, 지도의 요소(예: 도로, 지형, 물의 색상 등)를 커스터마이징하여 생성할 수 있습니다.
- 이 `mapId` 를 설정함으로써 해당 스타일이 적용된 지도를 표시할 수 있습니다.

예시

예를 들어, 기본 지도가 아닌 다크 모드 지도나 단순화된 회색 지도와 같은 맞춤형 스타일을 사용하려면, Google Maps Console에서 해당 스타일을 생성하고, 고유한 `mapId` 를 받아서 코드에 적용할 수 있습니다.

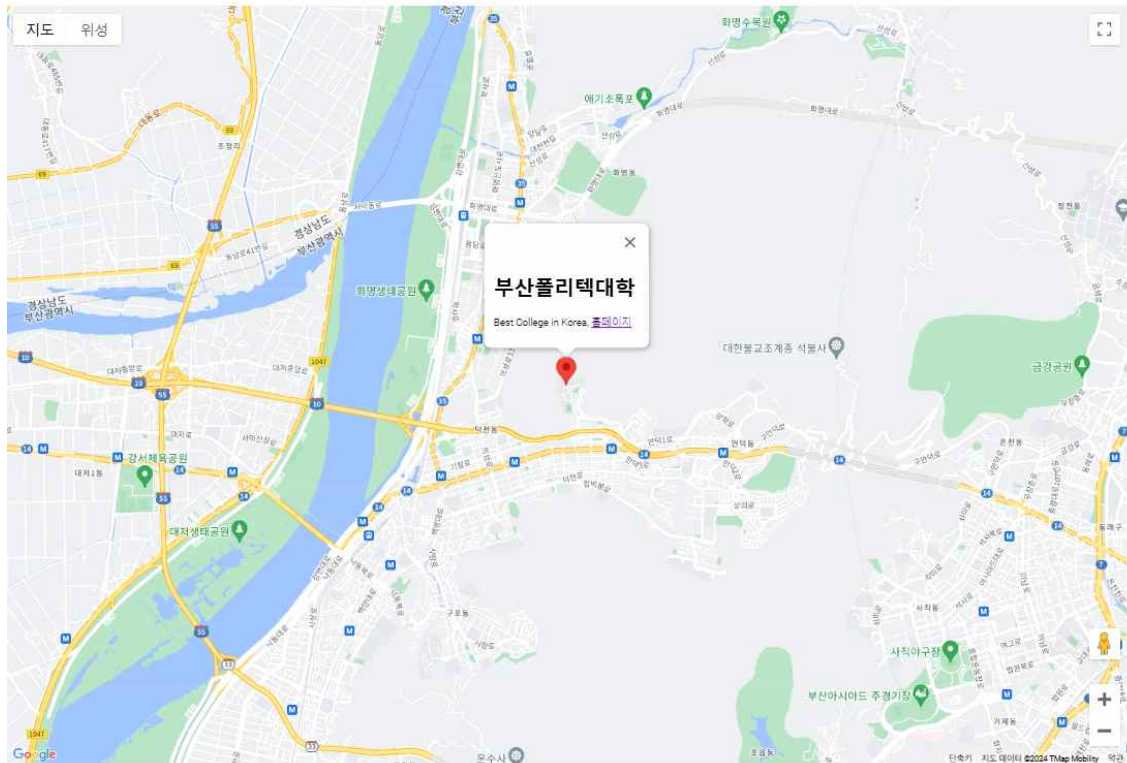
요약

`mapId` 는 Google Maps에서 사용자 맞춤형 지도 스타일을 불러오기 위한 고유 ID입니다. 이를 통해, 기본 스타일이 아닌 원하는 스타일로 지도를 꾸밀 수 있습니다.



정보창 표시하기

Google Maps Platform : JS



```
let map;
```

```
// This example displays a marker at the center of BusanPolytech.
```

```
// When the user clicks the marker, an info window opens.
```

```
async function initMap() {
```

```
  // Request needed libraries.
```

```
  const { Map } = await google.maps.importLibrary("maps");
```

```
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");
```

```
  const BusanPolytech = { lat: 35.219016, lng: 129.019002 }; // 부산폴리텍대학
```

```
  const map = new google.maps.Map(document.getElementById("map"), {
```

```
    center: BusanPolytech,
```

```
    zoom: 14,
```

```
    mapId: "4504f8b37365c3d0"
```

```
  });
```

```
  const marker = new AdvancedMarkerElement({
```

```
    map,
```

```
        position: BusanPolytech, // 부산폴리텍대학
        title: "Busan Polytech Collage"
    });

    const contentString =
        '<h1>부산폴리텍대학</h1>' +
        '<p>Best College in Korea, <a href="https://www.kopo.ac.kr/busan">홈페이지</a></p>'
    const infowindow = new google.maps.InfoWindow({
        content: contentString,
        ariaLabel: "Busan Polytech",
    });

    marker.addListener("click", () => {
        infowindow.open({
            anchor: marker,
            map,
        });
    });
}

initMap();
```

이 코드는 부산폴리텍대학 위치에 마커를 표시하고, 마커를 클릭하면 정보창을 여는 예제입니다. 아래에 각 부분을 쉽게 설명해드릴게요.

코드 설명

1. 전역 변수 `map` 선언:

```
javascript  
  
let map;
```

코드 복사

`map` 변수를 전역으로 선언했습니다. 이 변수는 전역에서 접근할 수 있도록 지도 객체를 저장하는 용도로 사용할 수 있습니다.

2. `initMap` 함수 정의:

```
javascript  
  
async function initMap() { ... }
```

코드 복사

- `initMap` 함수는 **Google Maps API**를 초기화하고 지도를 표시하기 위한 함수입니다.
- 비동기 함수로 설정하여 필요한 라이브러리를 기다려 불러옵니다.

3. Google Maps 라이브러리 불러오기:

```
javascript  
  
const { Map } = await google.maps.importLibrary("maps");  
const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");
```

코드 복사

- `Map` 과 `AdvancedMarkerElement` 라이브러리를 불러와서 Google Maps 기능을 사용할 준비를 합니다.

4. 부산폴리텍대학 위치 설정:

```
javascript  
  
const BusanPolytech = { lat: 35.219016, lng: 129.019002 }; // 부산폴리텍대학
```

코드 복사

- `BusanPolytech` 는 부산폴리텍대학의 위도와 경도를 나타냅니다. 지도 중심과 마커의 위치로 사용됩니다.

5. 지도 생성 및 초기화:

javascript

코드 복사

```
const map = new google.maps.Map(document.getElementById("map"), {
  center: BusanPolytech,
  zoom: 14,
  mapId: "4504f8b37365c3d0"
});
```

- `Map` 객체를 사용해 `id="map"` 인 HTML 요소에 지도를 표시합니다.
- `center: BusanPolytech`: 지도의 중심을 부산폴리텍대학으로 설정합니다.
- `zoom: 14`: 확대 수준을 14로 설정해, 해당 지역을 적절한 크기로 표시합니다.
- `mapId`: 맞춤형 지도 스타일을 설정하기 위한 고유 ID입니다.

6. 마커 생성:

javascript

코드 복사

```
const marker = new AdvancedMarkerElement({
  map,
  position: BusanPolytech,
  title: "Busan Polytech Collage"
});
```

- `AdvancedMarkerElement` 객체를 사용해 마커를 만듭니다.
- `position: BusanPolytech`: 마커의 위치를 부산폴리텍대학으로 지정합니다.
- `title`: 마커에 마우스를 올리면 나타나는 제목입니다.

7. 정보창 콘텐츠 설정:

javascript

코드 복사

```
const contentString =
  '<h1>부산폴리텍대학</h1>' +
  '<p>Best College in Korea, <a href="https://www.kopo.ac.kr/busan">홈페이지</a></p>'
```

- `contentString`에는 정보창에 표시할 HTML 콘텐츠가 들어 있습니다. 제목과 홈페이지 링크를 포함합니다.

8. 정보창 생성:

javascript

코드 복사

```
const infowindow = new google.maps.InfoWindow({
  content: contentString,
  ariaLabel: "Busan Polytech",
});
```

- `InfoWindow` 객체를 사용해 정보창을 만듭니다.
- `content: contentString`: 정보창에 표시할 내용을 `contentString`으로 설정합니다.
- `ariaLabel`: 접근성을 위한 라벨로, 스크린 리더가 읽어줄 텍스트입니다.

9. 마커 클릭 이벤트 추가:

javascript

코드 복사

```
marker.addListener("click", () => {
  infowindow.open({
    anchor: marker,
    map,
  });
});
```

- `marker.addListener("click", ...)`는 마커가 클릭될 때 정보창을 여는 이벤트 리스너입니다.
- `infowindow.open({ anchor: marker, map })`: 클릭 시 마커 위에 정보창을 표시하도록 설정합니다.
- `anchor: marker`는 마커를 기준으로 정보창을 위치시키라는 설정입니다. 이렇게 하면 정보창이 마커 바로 위에 뜹니다.
- `map`: 정보창을 표시할 지도 객체입니다. 이 경우, 이전에 설정된 `map`에 정보창을 표시합니다.


`addEventListener` 와 `addListener` 는 모두 이벤트를 감지하고 특정 작업을 실행하도록 설정하는 메서드지만, 사용되는 환경과 목적에 따라 차이가 있습니다.

1. `addEventListener`

- 기본 자바스크립트 메서드입니다.
- **HTML DOM** 요소에서 이벤트를 감지할 때 사용됩니다. 예를 들어, 버튼 클릭, 키 입력, 마우스 움직임 등과 같은 이벤트를 감지합니다.
- 거의 모든 최신 브라우저에서 지원하며, 표준 자바스크립트 문법입니다.

예시

javascript

 코드 복사

```
document.getElementById("myButton").addEventListener("click", () => {  
  console.log("Button clicked!");  
});
```

위 코드에서는 `myButton` 이라는 HTML 요소에 클릭 이벤트를 추가하여, 사용자가 버튼을 클릭할 때마다 콘솔에 "Button clicked!"를 출력합니다.

2. addListener

- **Google Maps API** 등 라이브러리에서 주로 사용되는 메서드입니다.
- DOM 요소가 아닌, ****Google Maps 객체(예: Marker, Map 등)****에 이벤트를 추가할 때 사용됩니다.
- Google Maps API가 제공하는 고유 메서드이므로, `google.maps.event.addListener` 와 같은 방식으로 사용됩니다.

예시

javascript

 코드 복사

```
const marker = new google.maps.Marker({  
  position: { lat: -25.363, lng: 131.044 },  
  map: map,  
  title: "Hello World!",  
});  
  
marker.addListener("click", () => {  
  console.log("Marker clicked!");  
});
```

위 코드에서는 `marker` 라는 Google Maps 마커 객체에 클릭 이벤트를 추가하여, 사용자가 마커를 클릭할 때마다 콘솔에 "Marker clicked!"를 출력합니다.

9. 마커 클릭 이벤트 추가:

javascript

 코드 복사

```
marker.addListener("click", () => {  
  infowindow.open({  
    anchor: marker,  
    map,  
  });  
});
```

- `marker.addListener("click", ...)` 는 마커가 클릭될 때 정보창을 여는 이벤트 리스너입니다.
- `infowindow.open({ anchor: marker, map })` : 클릭 시 마커 위에 정보창을 표시하도록 설정합니다.

10. `initMap` 함수 실행:

javascript

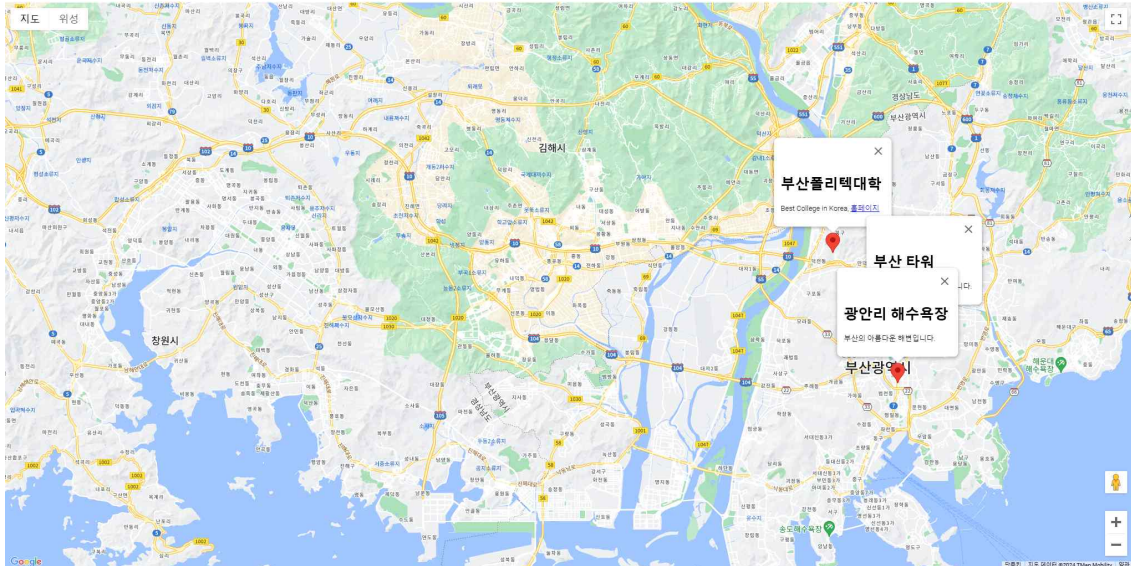
 코드 복사

```
initMap();
```

- `initMap` 함수를 호출하여 지도를 초기화하고 마커와 정보창을 설정합니다.

마커 추가

Google Maps Platform : JS



```
let map;

// Initialize the map and add markers at specified locations
async function initMap() {
  const { Map } = await google.maps.importLibrary("maps");
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");

  // Define locations
  const BusanPolytech = { lat: 35.219016, lng: 129.019002 }; // 부산폴리텍대학
  const BusanTower = { lat: 35.179554, lng: 129.075641 }; // 부산 타워
  const GwanganriBeach = { lat: 35.153093, lng: 129.059028 }; // 광안리 해수욕장

  // Initialize the map centered at BusanPolytech
  const map = new google.maps.Map(document.getElementById("map"), {
    center: BusanPolytech,
    zoom: 14,
    mapId: "4504f8b37365c3d0"
  });

  // Create markers
  const marker1 = new AdvancedMarkerElement({
    map,
    position: BusanPolytech,
```

```

        title: "Busan Polytech College"
    });

    const marker2 = new AdvancedMarkerElement({
        map,
        position: BusanTower,
        title: "Busan Tower"
    });

    const marker3 = new AdvancedMarkerElement({
        map,
        position: GwanganriBeach,
        title: "Gwanganri Beach"
    });

    // Info window content
    const contentString1 = '<h1>부산폴리텍대학</h1><p>Best College in Korea, <a href="https://www.kopo.ac.kr/busan">홈페이지</a></p>';
    const contentString2 = '<h1>부산 타워</h1><p>부산의 유명한 랜드마크입니다.</p>';
    const contentString3 = '<h1>광안리 해수욕장</h1><p>부산의 아름다운 해변입니다.</p>';

    // Create info windows
    const infowindow1 = new google.maps.InfoWindow({ content: contentString1 });
    const infowindow2 = new google.maps.InfoWindow({ content: contentString2 });
    const infowindow3 = new google.maps.InfoWindow({ content: contentString3 });

    // Add click listeners for info windows
    marker1.addListener("click", () => {
        infowindow1.open({ anchor: marker1, map });
    });
    marker2.addListener("click", () => {
        infowindow2.open({ anchor: marker2, map });
    });
    marker3.addListener("click", () => {
        infowindow3.open({ anchor: marker3, map });
    });
}

```

```
initMap();
```

정보창 클릭시 open, 한번더 클릭 close

```
let map;

// Initialize the map and add markers at specified locations
async function initMap() {
  const { Map } = await google.maps.importLibrary("maps");
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");

  // Define locations
  const BusanPolytech = { lat: 35.219016, lng: 129.019002 }; // 부산폴리텍대학
  const BusanTower = { lat: 35.179554, lng: 129.075641 }; // 부산 타워
  const GwangalliBeach = { lat: 35.153093, lng: 129.059028 }; // 광안리 해수욕장

  // Initialize the map centered at BusanPolytech
  const map = new google.maps.Map(document.getElementById("map"), {
    center: BusanPolytech,
    zoom: 14,
    mapId: "4504f8b37365c3d0"
  });

  // Create markers and info windows
  const markers = [
    {
      position: BusanPolytech,
      title: "Busan Polytech College",
      content: '<h1>부산폴리텍대학</h1><p>Best College in Korea, <a href="https://www.kopo.ac.kr/busan">홈페이지</a></p>',
    },
    {
      position: BusanTower,
      title: "Busan Tower",
      content: '<h1>부산 타워</h1><p>부산의 유명한 랜드마크입니다.</p>',
    },
    {
      position: GwangalliBeach,
      title: "Gwangalli Beach",
      content: '<h1>광안리 해수욕장</h1><p>부산의 아름다운 해변입니다.</p>',
    },
  ];
};
```

```

markers.forEach(({ position, title, content }) => {
  const marker = new AdvancedMarkerElement({
    map,
    position,
    title,
  });

  const infowindow = new google.maps.InfoWindow({
    content,
  });

  // isOpen 상태를 추적하는 변수
  let isOpen = false;

  marker.addListener("click", () => {
    if (isOpen) {
      infowindow.close(); // 이미 열려있으면 닫기
    } else {
      infowindow.open({ anchor: marker, map }); // 닫혀있으면 열기
    }
    isOpen = !isOpen; // 상태 반전
  });
});
}

initMap();

```

코드 설명

1. 지도와 마커 위치 정의:

- `BusanPolytech`, `Location1`, `Location2` 변수를 통해 마커 위치로 사용할 위도와 경도를 정의했습니다.
- `map` 객체는 `new google.maps.Map` 으로 초기화하며, 초기 중심 위치는 `BusanPolytech` 로 설정됩니다.

2. `markers` 배열을 이용한 마커 관리:

- `markers` 배열에는 마커와 정보 창에 필요한 **위치, 제목, 내용**이 객체 형태로 정의되어 있습니다.
- 이 배열을 사용하면 반복문을 통해 여러 마커와 정보 창을 효율적으로 생성할 수 있습니다.

3. `isOpen` 변수로 정보 창 상태 추적:

- 각 마커마다 `isOpen` 이라는 변수로 정보 창의 상태(열림 또는 닫힘)를 추적합니다. 처음에는 정보 창이 닫혀 있으므로 `isOpen` 을 `false` 로 설정합니다.
- 마커를 클릭하면 `isOpen` 의 상태를 확인하여:
 - `isOpen` 이 `true` 라면 정보 창이 이미 열려 있는 상태이므로 `infowindow.close()` 를 호출하여 닫습니다.
 - `isOpen` 이 `false` 라면 정보 창이 닫혀 있는 상태이므로 `infowindow.open()` 을 호출하여 엽니다.
- 클릭 후 `isOpen = !isOpen;` 을 사용해 `isOpen` 값을 반전시킵니다. 이로 인해 다음 클릭 시에는 반대 동작을 수행하게 됩니다.

4. 이벤트 리스너 추가:

- 각 마커에 클릭 이벤트 리스너를 추가하여 클릭할 때마다 정보 창의 열림/닫힘 상태가 변경되도록 설정합니다.

`markers.forEach` 구문은 `markers` 배열을 반복하면서 각 객체에 대해 다음과 같은 작업을 수행합니다:

1. `AdvancedMarkerElement` 인스턴스 생성:

- `const marker = new AdvancedMarkerElement(...)` 는 각 위치 객체에 대한 새로운 마커 객체를 생성합니다.
- `map`: 지도 객체를 지정하여 해당 마커가 표시될 지도를 설정합니다.
- `position`: 해당 마커가 나타날 위도와 경도 좌표를 설정합니다.
- `title`: 마커의 제목을 지정하여 마우스를 올릴 때 마커의 제목이 툴팁으로 나타나게 합니다.

2. 각 마커의 결과물:

- `markers` 배열에 있는 각 위치에 대해 `AdvancedMarkerElement` 가 생성되며, 이 결과로 `map` 에 다음과 같은 세 개의 마커가 표시됩니다:
 - `Busan Polytech College` (부산폴리텍대학 위치에 표시)
 - `Busan Tower` (부산 타워 위치에 표시)
 - `Gwangalli Beach` (광안리 해수욕장 위치에 표시)

이렇게 생성된 마커는 지도 위에 지정된 위치에 표시되며, 제목이 툴팁으로 표시됩니다.

지도의 중심을 BusanPolytech으로 고정

지도의 중심을 특정 위치(예: BusanPolytech)로 고정하려면 지도가 확대나 축소될 때마다 center_changed 이벤트를 사용하여 중심을 다시 BusanPolytech 위치로 설정할 수 있습니다.

아래와 같이 center_changed 이벤트 리스너를 추가하여 지도의 중심이 변경될 때마다 중심을 BusanPolytech로 되돌리도록 설정할 수 있습니다.

```
let map;

// Initialize the map and add markers at specified locations
async function initMap() {
  const { Map } = await google.maps.importLibrary("maps");
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");

  // Define locations
  const BusanPolytech = { lat: 35.219016, lng: 129.019002 }; // 부산폴리텍대학
  const BusanTower = { lat: 35.179554, lng: 129.075641 }; // 부산 타워
  const GwanganriBeach = { lat: 35.153093, lng: 129.059028 }; // 광안리 해수욕장

  // Initialize the map centered at BusanPolytech
  const map = new google.maps.Map(document.getElementById("map"), {
    center: BusanPolytech,
    zoom: 14,
    mapId: "4504f8b37365c3d0"
  });

  // Add center_changed event listener to keep map centered
  map.addListener("center_changed", () => {
    // Set a slight delay before resetting center to avoid immediate flicker
    setTimeout(() => {
      map.setCenter(BusanPolytech);
    }, 300); // 300ms 딜레이 후 중심을 고정
  });

  // Create markers and info windows
  const markers = [
    {
      position: BusanPolytech,
      title: "Busan Polytech College",
```

```

        content: '<h1>부산폴리텍대학</h1><p>Best College in Korea, <a
href="https://www.kopo.ac.kr/busan">홈페이지</a></p>',
    },
    {
        position: BusanTower,
        title: "Busan Tower",
        content: '<h1>부산 타워</h1><p>부산의 유명한 랜드마크입니다.</p>',
    },
    {
        position: GwanganriBeach,
        title: "Gwangalli Beach",
        content: '<h1>광안리 해수욕장</h1><p>부산의 아름다운 해변입니다.</p>',
    },
];

markers.forEach(({ position, title, content }) => {
    const marker = new AdvancedMarkerElement({
        map,
        position,
        title,
    });

    const infowindow = new google.maps.InfoWindow({
        content,
    });

    // isOpen 상태를 추적하는 변수
    let isOpen = false;

    marker.addListener("click", () => {
        if (isOpen) {
            infowindow.close(); // 이미 열려있으면 닫기
        } else {
            infowindow.open({ anchor: marker, map }); // 닫혀있으면 열기
        }
        isOpen = !isOpen; // 상태 반전
    });
});
}

```



```
initMap();
```

코드 설명

1. `center_changed` 이벤트 리스너:

- `map.addListener("center_changed", ...)` 부분은 지도의 중심이 변경될 때마다 호출됩니다.
- `setTimeout` 을 사용하여 300밀리초의 딜레이를 준 후에 `map.setCenter(BusanPolytech);` 를 호출하여 지도의 중심을 다시 `BusanPolytech` 으로 설정합니다. 이로 인해 사용자 조작에 따른 중심 이동이 지연 없이 `BusanPolytech` 으로 다시 고정됩니다.

2. 딜레이 적용 이유:

- 딜레이를 주지 않으면 지도 중심이 고정되는 과정에서 사용자가 줌을 할 때마다 미세한 깜빡임이 발생할 수 있어, 이를 줄이기 위해 딜레이를 설정했습니다.

이제 확대/축소 등의 작업을 하더라도 지도의 중심은 항상 `BusanPolytech` 위치로 고정됩니다.

Geocoding API vs BOM(navigator.geolocation)

지오코딩 API와 BOM(Browser Object Model)의 네비게이터 API는 위치 정보를 얻는 데 사용될 수 있지만, 각기 다른 방식과 용도를 가지고 있습니다.

1. Google Geocoding API

- **목적:** 주소를 위도와 경도로 변환하거나, 위도와 경도를 통해 주소를 얻기 위한 API입니다.
- **입력:** 주로 텍스트 형태의 주소를 입력으로 사용하여 해당 위치의 좌표(위도, 경도)를 반환합니다.
- **결과:** 사용자가 입력한 특정 주소의 위도와 경도를 반환합니다. 반대로 위도와 경도를 주소로 변환할 수도 있습니다.
- **사용 상황:** 사용자가 특정 위치를 주소로 입력하거나 검색했을 때, 해당 주소의 위도와 경도를 알고 싶은 경우.
- **제한:** Google Cloud에서 제공하는 서비스이므로 **API 키**가 필요하며, 일정 한도 이상 사용 시 과금됩니다.

2. BOM의 navigator.geolocation

- **목적:** 브라우저나 기기를 통해 현재 사용자의 실제 위치(위도와 경도)를 얻기 위한 API입니다.
- **입력:** 사용자의 허가(권한)가 필요하며, 위치 정보는 기기 **GPS** 또는 **네트워크** 정보를 통해 자동으로 수집됩니다.
- **결과:** 사용자가 현재 있는 장소의 위도와 경도 좌표를 반환합니다.
- **사용 상황:** 실시간으로 사용자의 현재 위치를 알고자 할 때 유용합니다. 예를 들어 날씨 앱에서 현재 위치의 날씨 정보를 보여주는 기능에 사용될 수 있습니다.
- **제한:** 사용자의 허가가 필요하며, 정확도는 기기와 네트워크 상태에 따라 달라질 수 있습니다. 특히, 데스크탑에서는 위치가 부정확할 수 있습니다.

주요 차이점

- **위치 결정 방식:** 지오코딩 API는 사용자가 입력한 주소를 기준으로 위치를 제공하고, 네비게이터는 사용자의 실제 현재 위치를 기반으로 좌표를 제공합니다.
- **용도:** 지오코딩 API는 주소 데이터를 좌표로 변환하는 데 적합하며, 네비게이터는 실시간 위치 기반 서비스에 적합합니다.
- **제한 및 과금:** 지오코딩 API는 Google의 유료 서비스로, 무료 한도 초과 시 과금됩니다. 네비게이터는 무료이지만, 사용자의 위치 접근 권한을 요구합니다.

따라서, 사용자가 특정 주소의 위도와 경도를 찾고자 할 때는 지오코딩 API를, 사용자의 현재 위치를 기반으로 서비스를 제공할 때는 BOM의 `navigator.geolocation` 을 사용하는 것이 적합합니다.

지오로케이션도 로컬호스트에서 작동하는 거지?

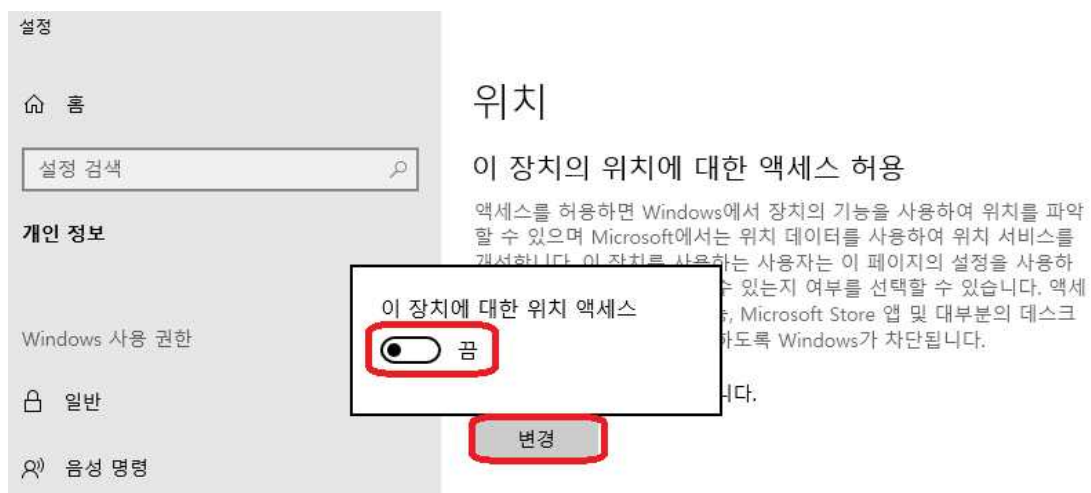
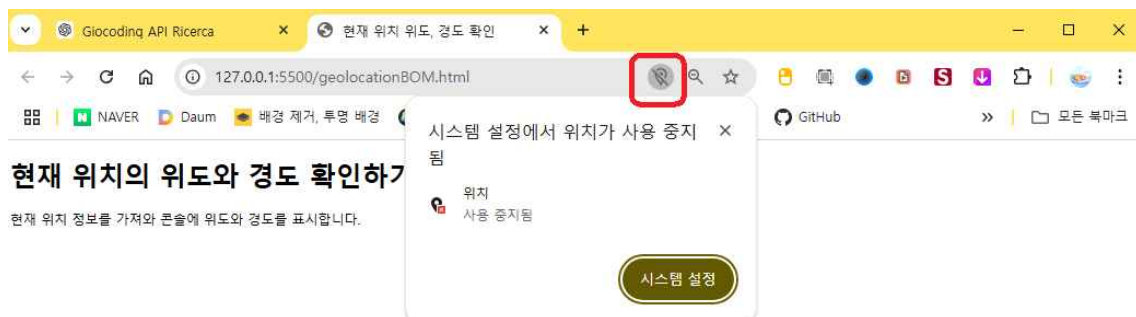
네, `navigator.geolocation` 은 로컬호스트에서도 작동합니다. 다만, 위치 정보를 요청할 때 브라우저에서 사용자의 위치 권한을 허용해야 위치 데이터를 얻을 수 있습니다.

로컬 환경에서 주의할 점

- **HTTPS 권장:** 대부분의 브라우저에서는 보안이 중요한 위치 정보를 요청할 때 HTTPS 프로토콜을 요구합니다. 그러나 개발 환경에서 **localhost**는 예외적으로 **HTTP**에서도 허용됩니다.
- **브라우저 권한 설정:** 위치 권한 요청이 뜨지 않거나, 허용이 안 되는 경우 브라우저의 설정에서 사이트별로 위치 접근 권한을 확인해야 할 수 있습니다.

따라서 로컬 개발 환경에서는 위치 권한을 허용하기만 하면, `navigator.geolocation` 을 사용하여 테스트할 수 있습니다.

이전에 실습하면서 위치접근허용을 안함으로 해 놓은 경우



위치

이 장치의 위치에 대한 액세스 허용

액세스를 허용하면 Windows에서 장치의 기능을 사용하여 위치를 파악할 수 있으며 Microsoft에서는 위치 데이터를 사용하여 위치 서비스를 개선합니다. 이 장치를 사용하는 사용자는 이 페이지의 설정을 사용하여 앱에서 위치에 액세스할 수 있는지 여부를 선택할 수 있습니다. 액세스를 거부하면 Windows 기능, Microsoft Store 앱 및 대부분의 데스크톱 앱에 위치를 제공하지 못하도록 Windows가 차단됩니다.

이 장치의 위치가 켜져 있습니다.

변경

앱에서 사용자의 위치에 액세스하도록 허용

액세스를 허용하면 이 페이지의 설정을 통해 장치의 정확한 위치 및 위치 기록에 액세스할 수 있는 앱을 선택하여 길찾기 날씨 같은 위치 기반 환경을 사용할 수 있습니다. 이 장치에서 Microsoft 계정을 사용하여 로그인한 경우 마지막으로 알려진 위치가 클라우드에 저장되고 Microsoft 계정으로 로그인한 다른 장치와 공유됩니다. 액세스를 거부하면 이 페이지에 나열된 앱만 사용자의 위치에 액세스하지 못하도록 차단됩니다.

 켜

이 페이지의 설정이 꺼져 있는 경우에도 일부 데스크톱 앱에서 사용자의 위치를 확인할 수 있습니다. [이유를 확인하세요](#)

앱에서 위치를 사용 중이면 이 위치 사용 중 아이콘이 표시됩니다. ●

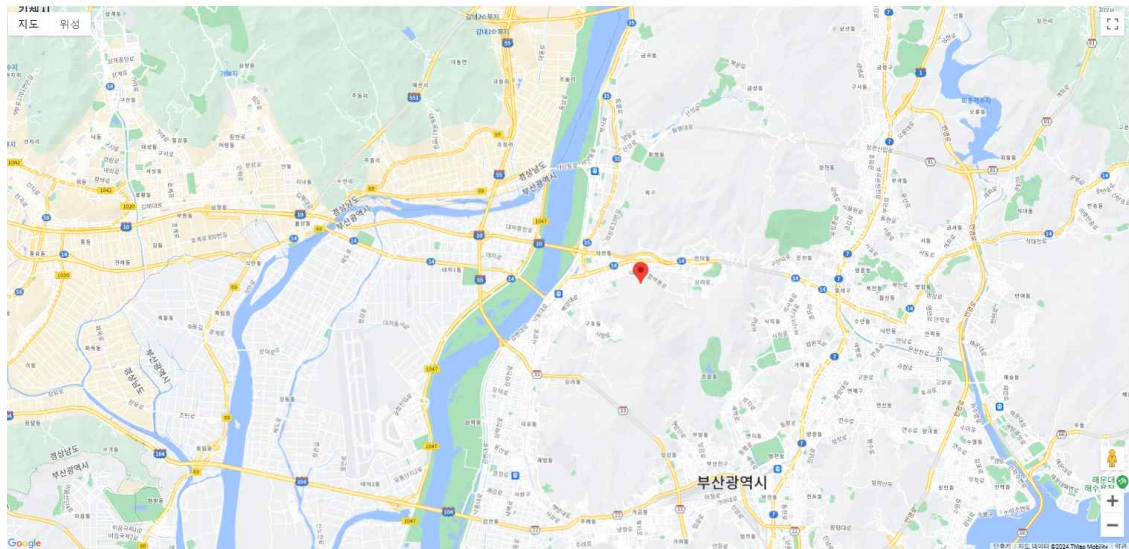
geolocation 기본 예제

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>현재 위치 위도, 경도 확인</title>
</head>
<body>
  <h1>현재 위치의 위도와 경도 확인하기</h1>
  <p>현재 위치 정보를 가져와 콘솔에 위도와 경도를 표시합니다.</p>

  <script>
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(
        (position) => {
          const latitude = position.coords.latitude;
          const longitude = position.coords.longitude;
          console.log("현재 위치의 위도:", latitude);
          console.log("현재 위치의 경도:", longitude);
          alert("현재 위치의 위도: " + latitude + "\n현재 위치의 경도: " +
longitude);
        },
        (error) => {
          console.error("위치 정보를 가져올 수 없습니다:", error.message);
          alert("위치 정보를 가져올 수 없습니다: " + error.message);
        }
      );
    } else {
      console.log("이 브라우저는 위치 정보를 지원하지 않습니다.");
      alert("이 브라우저는 위치 정보를 지원하지 않습니다.");
    }
  </script>
</body>
</html>
```

Google Maps와 연동

Google Maps Platform : JS



```
let map;
```

```
async function initMap() {  
  const { Map } = await google.maps.importLibrary("maps");  
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");  
  
  // Check if geolocation is available  
  if (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(  
      (position) => {  
        // Get user's current position  
        const userLocation = {  
          lat: position.coords.latitude,  
          lng: position.coords.longitude,  
        };  
  
        console.log(userLocation.lat);  
        console.log(userLocation.lng);  
  
        // Initialize the map centered at user's location  
        map = new google.maps.Map(document.getElementById("map"), {  
          center: userLocation,  
          zoom: 14,  
          mapId: "4504f8b37365c3d0",  
        });  
      }  
    );  
  }  
}
```

```

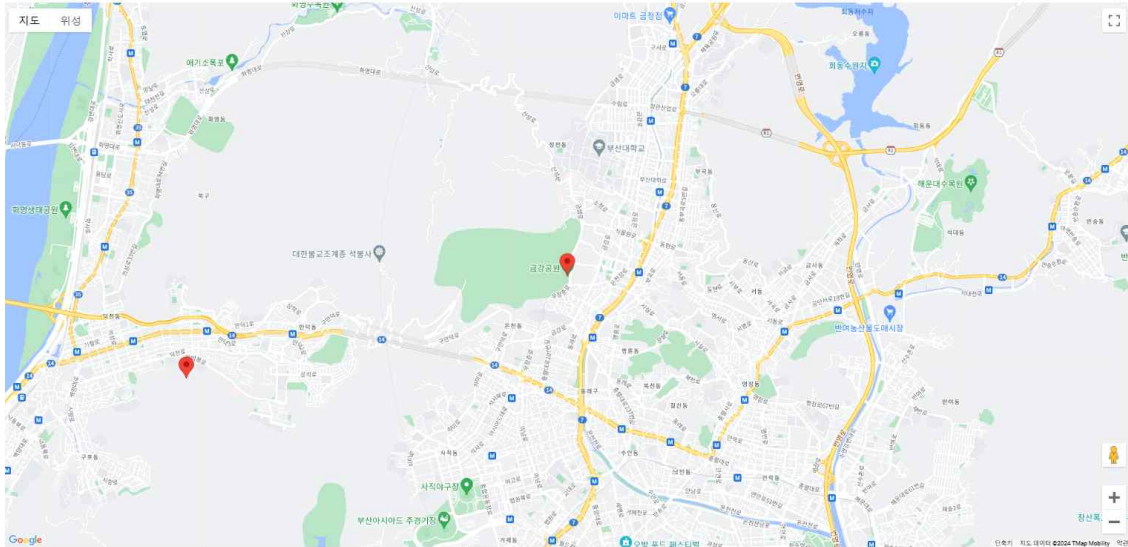
        // Add a marker at the user's location
        new AdvancedMarkerElement({
            map,
            position: userLocation,
            title: "Your Location",
        });
    },
    () => {
        alert("현재 위치를 찾을 수 없습니다.");
        // 기본 중심으로 설정 (예: 부산 폴리텍 대학)
        const defaultLocation = { lat: 35.219016, lng: 129.019002 };
        map = new google.maps.Map(document.getElementById("map"), {
            center: defaultLocation,
            zoom: 14,
            mapId: "4504f8b37365c3d0",
        });
    }
);
} else {
    alert("이 브라우저에서는 위치 기능을 지원하지 않습니다.");
    const defaultLocation = { lat: 35.219016, lng: 129.019002 };
    map = new google.maps.Map(document.getElementById("map"), {
        center: defaultLocation,
        zoom: 14,
        mapId: "4504f8b37365c3d0",
    });
}
}

initMap();

```


클릭한 곳으로 지도 중심 이동하고 새로운 마커 생성

Google Maps Platform : JS



```
let map;
```

```
async function initMap() {  
  const { Map } = await google.maps.importLibrary("maps");  
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");  
  
  // Check if geolocation is available  
  if (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(  
      (position) => {  
        // Get user's current position  
        const userLocation = {  
          lat: position.coords.latitude,  
          lng: position.coords.longitude,  
        };  
  
        // Initialize the map centered at user's location  
        map = new google.maps.Map(document.getElementById("map"), {  
          center: userLocation,  
          zoom: 14,  
          mapId: "4504f8b37365c3d0",  
        });  
      });  
  }  
}
```



```

        // Add a marker at the user's location
        new AdvancedMarkerElement({
            map,
            position: userLocation,
            title: "Your Location",
        });

        // Map click event listener to move center
        map.addListener("click", (event) => {
            const clickedLocation = {
                lat: event.latLng.lat(),
                lng: event.latLng.lng(),
            };
            map.setCenter(clickedLocation);

            // Optional: Add a marker at the clicked location
            new AdvancedMarkerElement({
                map,
                position: clickedLocation,
                title: "Clicked Location",
            });
        });
    },
    () => {
        alert("현재 위치를 찾을 수 없습니다.");
        const defaultLocation = { lat: 35.219016, lng: 129.019002 };
        map = new google.maps.Map(document.getElementById("map"), {
            center: defaultLocation,
            zoom: 14,
            mapId: "4504f8b37365c3d0",
        });
    }
);
} else {
    alert("이 브라우저에서는 위치 기능을 지원하지 않습니다.");
    const defaultLocation = { lat: 35.219016, lng: 129.019002 };
    map = new google.maps.Map(document.getElementById("map"), {
        center: defaultLocation,
        zoom: 14,
    });
}

```

```
        mapId: "4504f8b37365c3d0",
    });
}
}

initMap();
```

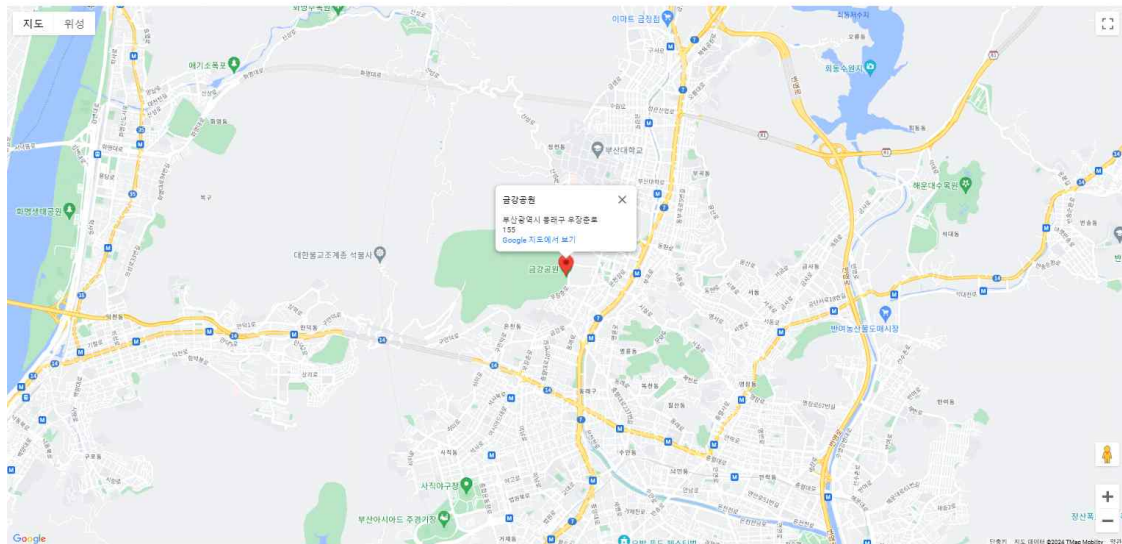
추가된 기능 설명

1. `map.addListener("click", ...)`: 사용자가 지도를 클릭하면 발생하는 이벤트 리스너를 추가합니다.
2. `event.latLng`: 클릭한 위치의 위도와 경도를 가져옵니다.
3. `map.setCenter(clickedLocation)`: 지도의 중심을 클릭한 위치(`clickedLocation`)로 이동시킵니다.
4. 선택사항 - 마커 추가: 클릭한 위치에 마커를 추가하여 클릭 위치를 표시할 수도 있습니다.

이 코드를 실행하면 지도를 클릭할 때마다 지도의 중심이 클릭한 위치로 이동하게 됩니다.

중심이동후 새로운 마커가 생성되면 이전 마커는 삭제

Google Maps Platform : JS



```
let map;

let currentMarker = null; // 현재 마커를 추적할 변수

async function initMap() {
  const { Map } = await google.maps.importLibrary("maps");
  const { AdvancedMarkerElement } = await google.maps.importLibrary("marker");

  // Check if geolocation is available
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      (position) => {
        const userLocation = {
          lat: position.coords.latitude,
          lng: position.coords.longitude,
        };

        // Initialize the map centered at user's location
        map = new google.maps.Map(document.getElementById("map"), {
          center: userLocation,
          zoom: 14,
          mapId: "4504f8b37365c3d0",
        });

        // Add a marker at the user's location
```

```

        currentMarker = new AdvancedMarkerElement({
            map,
            position: userLocation,
            title: "Your Location",
        });

        // Map click event listener to move center and add a new marker
        map.addListener("click", (event) => {
            const clickedLocation = {
                lat: event.latLng.lat(),
                lng: event.latLng.lng(),
            };

            // Remove the previous marker if it exists
            if (currentMarker) {
                currentMarker.map = null; // 지도에서 기존 마커 제거
            }

            // Add a new marker at the clicked location
            currentMarker = new AdvancedMarkerElement({
                map,
                position: clickedLocation,
                title: "Clicked Location",
            });

            // Move the map center to the clicked location
            map.setCenter(clickedLocation);
        });
    },
    () => {
        alert("현재 위치를 찾을 수 없습니다.");
        const defaultLocation = { lat: 35.219016, lng: 129.019002 };
        map = new google.maps.Map(document.getElementById("map"), {
            center: defaultLocation,
            zoom: 14,
            mapId: "4504f8b37365c3d0",
        });
    }
};

} else {

```

```
        alert("이 브라우저에서는 위치 기능을 지원하지 않습니다.");
        const defaultLocation = { lat: 35.219016, lng: 129.019002 };
        map = new google.maps.Map(document.getElementById("map"), {
            center: defaultLocation,
            zoom: 14,
            mapId: "4504f8b37365c3d0",
        });
    }
}

initMap();
```

코드 설명

1. **currentMarker** 변수 추가: 현재 지도에 표시된 마커를 저장하기 위한 변수입니다.
2. 기존 마커 제거: `map.addListener("click", ...)` 내부에서 새로운 마커를 추가하기 전에, `if (currentMarker)` 조건을 확인하여 이전 마커를 지도에서 제거합니다. `currentMarker.map = null;` 을 통해 기존 마커가 지도에서 제거됩니다.
3. 새로운 마커 추가: 클릭한 위치에 새로운 마커를 추가하고 `currentMarker` 변수에 저장합니다.
4. 중심 이동: 지도의 중심을 클릭한 위치로 이동합니다.

이제 새로운 위치를 클릭할 때마다 이전 마커가 제거되고, 클릭한 위치에 새로운 마커가 표시됩니다.

`let currentMarker = null;`의 초기값을 `null`로 설정한 이유는 처음에는 마커가 없음을 명시하기 위해서입니다. 이를 통해 코드가 처음 실행될 때 `currentMarker`가 비어 있음을 확실히 하고, 이후 조건문을 통해 마커가 있는지 없는지를 쉽게 확인할 수 있습니다.

자세한 설명

1. 초기 상태 명시:

- 코드가 처음 실행될 때 `currentMarker`는 마커가 없으므로, `null`로 초기화하여 "현재 마커가 없다"는 상태를 명확하게 나타냅니다.

2. 조건문에서의 존재 확인:

- 클릭 이벤트가 발생할 때마다 `if (currentMarker)` 조건문을 통해 `currentMarker`가 `null`이 아닌 경우에만 이전 마커를 제거합니다.
- `null`로 초기화하지 않으면 `currentMarker`가 `undefined` 상태가 되어 조건문이 의도한 대로 동작하지 않을 수 있습니다.

3. 일관된 값 설정:

- `null`은 코드에서 "값이 아직 설정되지 않음"을 표현하는 명확한 방법입니다. 초기값이 없을 경우 나중에 변수가 마커 객체를 가리킬지 `undefined`일지 불명확해질 수 있기 때문에 `null`로 설정하는 것이 좋습니다.

이렇게 `null`로 초기화하면 코드가 더 안정적이고 예측 가능하게 동작하게 됩니다.

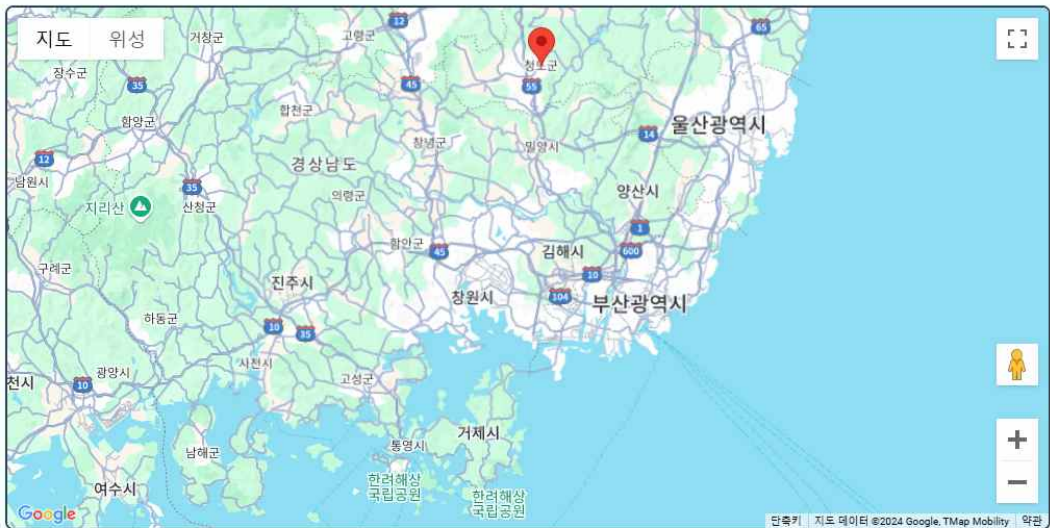
현재 날씨를 나의 위치기반으로 표시하고, 구글 지도도 현재 위치를 마
커하고, 구글지도의 위치를 클릭하였을때 클릭한 곳의 날씨가 표시

위치 기반 날씨와 구글 지도

알고 싶은 지역의 날씨를 지도에서 클릭하세요^^

Gyeongsan-si의 현재 온도: 12.94°C

날씨: 온흐림



```
// google-maps-api-loader.js

const API_KEY = 'AIzaSyC5SHxpSEBZMPA_HMzGeRlcqZWARVk9QQU';
const REGION = 'KR';
const LANG = 'ko';

(g => {
  var h,
    a,
    k,
    p = "The Google Maps JavaScript API",
    c = "google",
    l = "importLibrary",
    q = "__ib__",
    m = document,
    b = window;
  b = b[c] || (b[c] = {});

  var d = b.maps || (b.maps = {}),
    r = new Set,
    e = new URLSearchParams,
    u = () => h || (h = new Promise(async (f, n) => {
      await (a = m.createElement("script")); e.set("libraries", [...r] + "");
      for (k in g) e.set(k.replace(/[A-Z]/g, t => "_" + t[0].toLowerCase()), g[k]);
      e.set("callback", c + ".maps." + q);
      a.src = `https://maps.${c}apis.com/maps/api/js?` + e; d[q] = f;
      a.onerror = () => h = n(Error(p + " could not load."));
      a.nonce = m.querySelector("script[nonce]")?.nonce || ""; m.head.append(a)
    }));

  d[l] ? console.warn(p + " only loads once. Ignoring:", g) : d[l] = (f, ...n) => r.add(f) &&
  u().then(() => d[l](f, ...n))
})){
  key: API_KEY,
  v: "weekly",
  // Use the 'v' parameter to indicate the version to use (weekly, beta, alpha, etc.).
  // Add other bootstrap parameters as needed, using camel case.
  region: REGION,
  language: LANG
};
```



```
<- weather.html ->
```

```
<!DOCTYPE html>
```

```
<html lang="kr">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

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

```
  <title>위치 기반 날씨 정보와 구글 지도</title>
```

```
  <link rel="stylesheet" href="/weather.css">
```

```
  <script src="/google-maps-api-loader.js"> </script>
```

```
  <script src="/weather.js" defer> </script>
```

```
</head>
```

```
<body>
```

```
  <h1>내 위치 기반 날씨와 구글 지도</h1>
```

```
  <button id="getWeather">현재 위치 날씨 가져오기</button>
```

```
  <div id="weather"></div>
```

```
  <div id="map"></div>
```

```
</body>
```

```
</html>
```

```
<- weather.css ->
```

```
/* 전체 레이아웃 설정 */
```

```
body {
```

```
  font-family: Arial, sans-serif;
```

```
  text-align: center;
```

```
  background-color: #f0f8ff;
```

```
  color: #333;
```

```
  margin: 0;
```

```
  padding: 0;
```

```
}
```

```
/* 제목 스타일 */
```

```
h1 {
```

```
  color: #2c3e50;
```

```
  margin-top: 20px;
```

```
}
```

```
h3 {
```

```
  color: #34495e;
```

```
  margin-bottom: 20px;
```

```
font-weight: 400;
}

/* 지도 스타일 */
#map {
    height: 500px;
    width: 80%;
    max-width: 1000px;
    margin: 20px auto;
    border: 2px solid #2c3e50;
    border-radius: 8px;
}

/* 날씨 정보 스타일 */
#weather {
    background-color: #fff;
    padding: 15px;
    margin: 20px auto;
    width: 60%;
    max-width: 500px;
    border: 1px solid #dcdcdc;
    border-radius: 8px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    font-size: 1.2em;
    color: #333;
}

#weather p {
    margin: 10px 0;
    font-size: 1em;
}
```

```

// weather.js

// HTML 요소 가져오기
const weatherDiv = document.getElementById('weather');
const getWeatherBtn = document.getElementById('getWeather');

// API 키와 URL 설정
const openWeatherApiKey = 'a546e42e9c1e18a07c48cd26d92f67ef'; //
// OpenWeatherMap API 키를 입력
const weatherApiUrl = 'https://api.openweathermap.org/data/2.5/weather';
let map, marker;

// 날씨 정보 가져오기 함수
function fetchWeather(latitude, longitude) {
    const url =
` ${weatherApiUrl}?lat=${latitude}&lon=${longitude}&appid=${openWeatherApiKey}&units=metric&lang=kr`;

    fetch(url)
        .then(response => response.json())
        .then(data => {
            const { main, weather, name } = data;
            const temp = main.temp;
            const description = weather[0].description;
            const city = name;

            weatherDiv.innerHTML = `
<p>${city ? city : '선택한 위치'}의 현재 온도: ${temp}°C</p>
<p>날씨: ${description}</p>`;
        })
        .catch(error => {
            console.error('날씨 정보를 가져오는 중 오류가 발생했습니다:', error);
            weatherDiv.textContent = '날씨 정보를 가져오는 중 오류가 발생했습니다.';
        });
}

// 사용자의 위치를 가져오는 함수
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(

```

```

        (position) => {
            const { latitude, longitude } = position.coords;
            fetchWeather(latitude, longitude);
            initMap(latitude, longitude); // 지도를 현재 위치로 초기화
        },
        (error) => {
            console.error('위치를 가져오는 중 오류가 발생했습니다:', error);
            weatherDiv.textContent = '위치를 가져오는 중 오류가 발생했습니다.';
        }
    );
} else {
    weatherDiv.textContent = '위치 서비스를 지원하지 않는 브라우저입니다.';
}
}

// 구글 지도 초기화 함수
async function initMap(latitude, longitude) {
    // 위치 설정
    const myPosition = { lat: latitude, lng: longitude };

    // 지도 생성
    // Request needed libraries.
    const { Map } = await google.maps.importLibrary("maps");
    const map = new google.maps.Map(document.getElementById("map"), {
        center: myPosition,
        zoom: 14,
    });

    // 기본 마커 설정
    const marker = new google.maps.Marker({
        position: myPosition,
        map,
        title: "현재 위치",
    });

    // 지도 클릭 이벤트 추가
    map.addListener("click", (event) => {
        const clickedLat = event.latLng.lat();
        const clickedLng = event.latLng.lng();
    });
}

```

```
// 마커 위치 업데이트
marker.setPosition({ lat: clickedLat, lng: clickedLng });
marker.setTitle("선택한 위치");

// 클릭한 위치의 날씨 가져오기
fetchWeather(clickedLat, clickedLng);
});
}

// 버튼 클릭 시 현재 위치 기반 날씨 정보 가져오기
getWeatherBtn.addEventListener('click', getLocation);
```

도시이름 한국어로 바꾸기

- 일부 대도시(서울 부산 인천)은 영문으로 표기됨???

영어 도시 이름을 한국어로 바꾸기 위해 다음과 같은 방법을 사용할 수 있습니다:

1. **OpenWeatherMap**에서 `lang=kr` 파라미터 사용: 이미 `fetchWeather` 함수의 API 호출에 `lang=kr` 을 추가해 한국어 날씨 설명을 가져오고 있지만, 일부 도시 이름은 여전히 영어로 제공될 수 있습니다. 이 경우 Google Maps Geocoding API 또는 Kakao Local API 같은 지오코딩 서비스를 활용하는 방법이 좋습니다.
2. **Google Maps Geocoding API** 사용하기:
 - 사용자가 지도를 클릭할 때 Google Maps Geocoding API를 호출하여, 클릭한 위치의 위도와 경도를 기반으로 한국어 주소 정보를 가져옵니다.
 - `address_components` 에서 `locality` 필드를 참조하여 도시 이름을 추출할 수 있습니다.

일부 대도시(예: 서울, 부산, 인천)의 경우, 오픈웨더 또는 구글 지오코딩 API에서 영어 도시 이름으로 반환되는 이유는 다음과 같은 원인일 수 있습니다.

1. **API 데이터 제한:** 일부 API는 특정 대도시에 대해서만 데이터베이스에 영어 이름을 저장하거나, 지역별 기본 언어 설정과 관계없이 영어 이름을 반환할 수 있습니다. 이는 API 제공자가 국제적으로 더 널리 알려진 영어 이름을 반환하는 방식으로 설계한 경우입니다.
2. **지오코딩 API의 언어 설정 제한:** 구글 지오코딩 API의 경우, `language=ko` 파라미터가 있어도 정확히 일치하는 한국어 도시명을 반환하지 않을 수 있습니다. 구글은 일부 주요 도시에 대해 영어 이름을 우선하여 보여주는 정책이 있을 수 있습니다.
3. **API의 캐싱 및 데이터 갱신 문제:** 오픈웨더 또는 구글의 데이터베이스가 특정 지역의 언어 설정에 대한 갱신이 늦어질 수 있습니다. 이에 따라 최신 한국어 명칭 정보가 반영되지 않아 영어로 표시되는 경우가 있습니다.

해결 방안

1. **직접 매핑:** 한국 주요 도시(예: 서울, 부산, 인천 등)의 영어 이름을 한국어로 매핑하는 로컬 데이터베이스를 만들어, 영어 이름이 반환될 경우 해당 이름을 한국어로 변환해 보여줄 수 있습니다.
2. **대체 지오코딩 서비스 사용:** 한국에서는 카카오나 네이버의 지오코딩 API가 더 정확하게 한국어 주소를 반환할 수 있습니다. 이를 통해 한국어 도시 이름을 가져오는 방법도 고려할 수 있습니다.

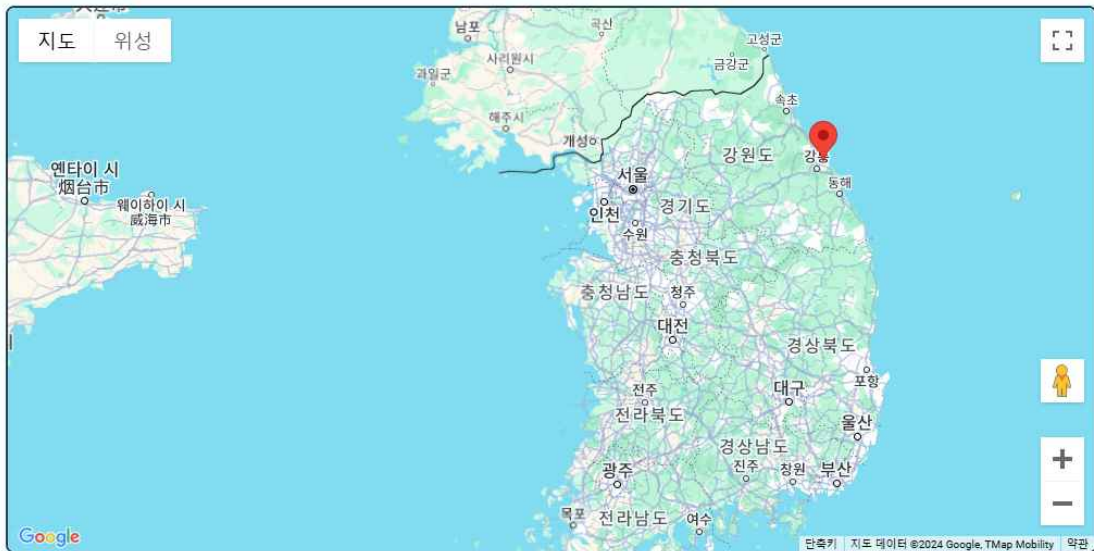
특히 한국 내 사용자를 대상으로 하는 서비스라면 카카오 API 등을 활용하는 것이 더 정확한 도시 이름을 제공할 수 있는 좋은 대안이 될 수 있습니다.

위치 기반 날씨와 구글 지도

알고 싶은 지역의 날씨를 지도에서 클릭하세요^^

강릉시의 현재 온도: 16.19°C

날씨: 온흐림



```
// HTML 요소 가져오기
```

```
const weatherDiv = document.getElementById('weather');
```

```
const getWeatherBtn = document.getElementById('getWeather');
```

```
// API 키와 URL 설정
```

```
const openWeatherApiKey = 'a546e42e9c1e18a07c48cd26d92f67ef'; //
```

```
OpenWeatherMap API 키를 입력
```

```
const weatherApiUrl = 'https://api.openweathermap.org/data/2.5/weather';
```

```
let map, marker;
```

```
// 날씨 정보 가져오기 함수
```

```
// 수정된 fetchWeather 함수(도시이름 한국어로)
```

```
function fetchCityNameInKorean(latitude, longitude) {
```

```
    const geocodingUrl =
```

```
`https://maps.googleapis.com/maps/api/geocode/json?latlng=${latitude},${longitude}&key=${API_KEY}&language=ko`;
```

```
    return fetch(geocodingUrl)
```

```
        .then(response => response.json())
```

```
        .then(data => {
```

```

        const addressComponents = data.results[0].address_components;
        const cityComponent = addressComponents.find(component =>
            component.types.includes("locality")
        );
        return cityComponent ? cityComponent.long_name : null;
    })
    .catch(error => {
        console.error("도시 이름을 가져오는 중 오류가 발생했습니다:", error);
        return null;
    });
}

function fetchWeather(latitude, longitude) {
    fetchCityNameInKorean(latitude, longitude).then(cityNameInKorean => {
        const url =
` ${weatherApiUrl}?lat=${latitude}&lon=${longitude}&appid=${openWeatherApiKey}&units=metric&lang=kr`;

        fetch(url)
            .then(response => response.json())
            .then(data => {
                const { main, weather } = data;
                const temp = main.temp;
                const description = weather[0].description;
                const city = cityNameInKorean || data.name;

                weatherDiv.innerHTML = `
                <p>${city}의 현재 온도: ${temp}°C</p>
                <p>날씨: ${description}</p>`;
            })
            .catch(error => {
                console.error("날씨 정보를 가져오는 중 오류가 발생했습니다:", error);
                weatherDiv.textContent = "날씨 정보를 가져오는 중 오류가 발생했습니다.";
            });
    });
}

// 사용자의 위치를 가져오는 함수

```



```

function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      (position) => {
        const { latitude, longitude } = position.coords;
        fetchWeather(latitude, longitude);
        initMap(latitude, longitude); // 지도를 현재 위치로 초기화
      },
      (error) => {
        console.error('위치를 가져오는 중 오류가 발생했습니다:', error);
        weatherDiv.textContent = '위치를 가져오는 중 오류가 발생했습니다.';
      }
    );
  } else {
    weatherDiv.textContent = '위치 서비스를 지원하지 않는 브라우저입니다.';
  }
}

// 구글 지도 초기화 함수
async function initMap(latitude, longitude) {
  // 위치 설정
  const myPosition = { lat: latitude, lng: longitude };

  // 지도 생성
  // Request needed libraries.
  const { Map } = await google.maps.importLibrary("maps");
  const map = new google.maps.Map(document.getElementById("map"), {
    center: myPosition,
    zoom: 11,
  });

  // 기본 마커 설정
  const marker = new google.maps.Marker({
    position: myPosition,
    map,
    title: "현재 위치",
  });

  // 지도 클릭 이벤트 추가
  map.addListener("click", (event) => {

```

```
const clickedLat = event.latLng.lat();
const clickedLng = event.latLng.lng();

// 마커 위치 업데이트
marker.setPosition({ lat: clickedLat, lng: clickedLng });
marker.setTitle("선택한 위치");

// 클릭한 위치의 날씨 가져오기
fetchWeather(clickedLat, clickedLng);
});
}

// 버튼 클릭 시 현재 위치 기반 날씨 정보 가져오기
// getWeatherBtn.addEventListener('click', getLocation);
getLocation();
```

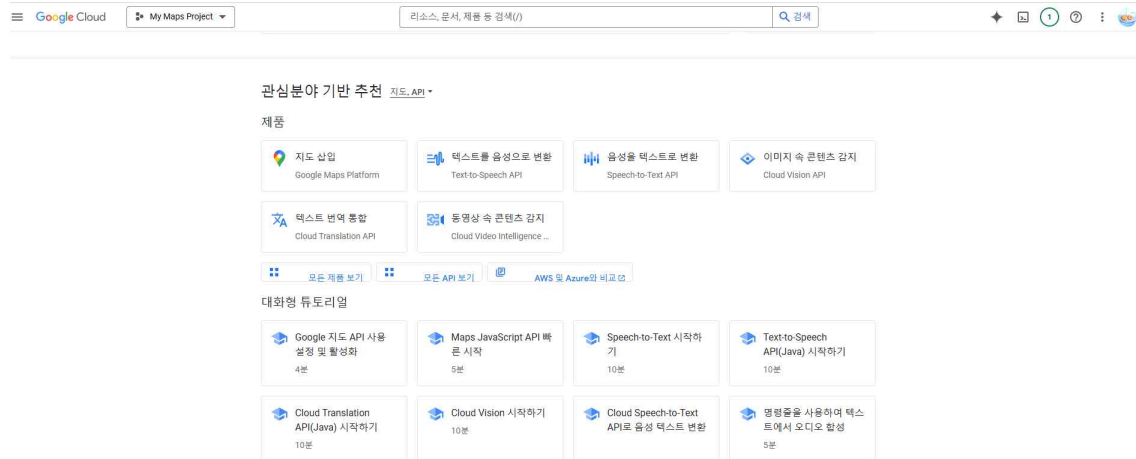
Google Maps Platform으로 할 수 있는 작업 알아보기

<https://console.cloud.google.com/google/maps-apis/build?project=august-tangent-439500-j3>

몇 분 만에 위치 기반 환경을 맞춤설정하고 실행합니다.

관심분야 기반 추천 : 지도 API

<https://console.cloud.google.com/welcome/new?project=august-tangent-439500-j3>



℃