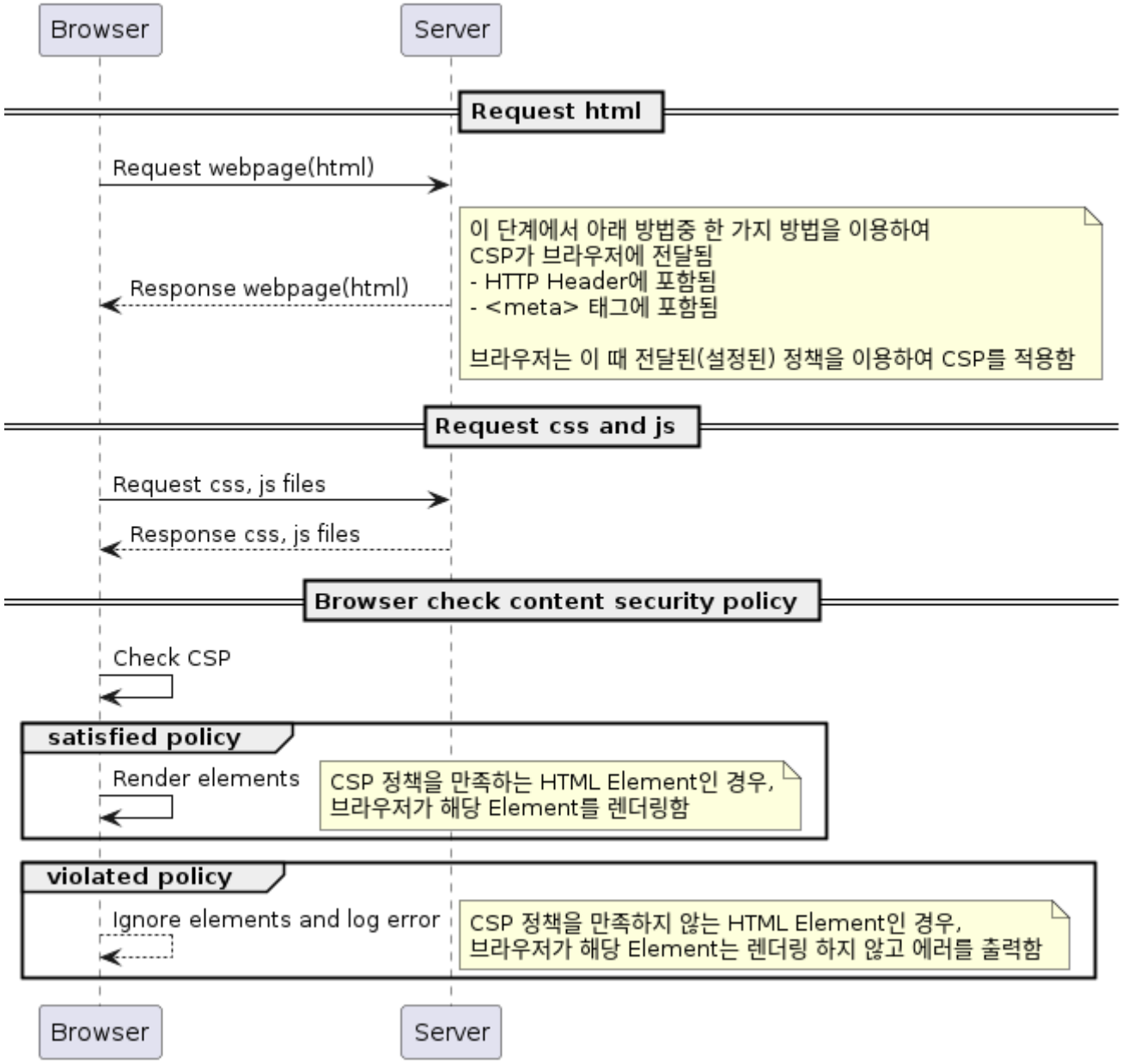
Content Security Policy

개요

Content-Security-Policy(이하 CSP)에 대한 글입니다.

CSP의 목적, 종류, 적용 방법 등 CSP에 대한 전반적인 내용을 NodeJS 라이브러리 관련 내용을 최대한 제외하고 작성하였습니다.



간략화한 CSP 적용 sequence diagram. CORS와 유사하게 동작한다.

> ℹ 일부 Javascript / HTML은 ReactJS를 기반으로 설명합니다.

CSP의 목적

- 브라우저를 타겟으로 하는 다양한 종류의 공격을 방어하기 위한 ***추가적인*** 보안 레이어
  - 공격의 예시
    - Cross-Site Scripting(XSS)
    - data injection attack
- `<script>` `<div>` 등 태그 단위로 Element 실행 여부를 결정할 수 있음
  → 파일(`sha-256`), 태그(`nonce-…`) 단위로 정책 설정 가능

> `sha-256` (hash)는 파일에서의 md5와 유사합니다.
> 예를 들어, `alert('Hello, World.');` 의 sha-256 값은
> `qznLcsROx4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng=` 입니다.
> 해당 script를 허용하려면 다음과 같이 `<meta>` 태그를 설정합니다.
>
> ```
> <html>
>   <head>
>     <meta
>       http-equiv="Content-Security-Policy"
>       content="script-src 'sha256-
> qznLcsROx4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng='"
>     />
>     <script>alert('Hello, world.');</script>
>   </head>
> </html>
> ```
>
> sha-256은 아래와 같은 방법으로 생성 가능합니다.
>
> ```
> echo -n "alert('Hello, world.');" | openssl dgst -sha256 -binary |
> openssl enc -base64
> ```

XSS

- `Cross-Site Scripting`의 약자
- HTML의 javascript event handler에 악의적인 스크립트를 삽입, 해당 스크립트를 실행

코드 예시

```
// 이 경우, 이미지가 로딩되지 않으므로 1이 포함된 팝업이 출력됨
var app = document.querySelector('#app');
app.innerHTML = '<img src="x" onerror="alert(1)">';
```

HTML의 `Element.innerHTML` 또는 react의 `dangerouslySetInnerHTML`로 `<script>` 태그를 삽입한 경우, 해당 스크립트는 실행되지 않습니다.
`<script>` 태그를 HTML에 삽입하려는 스크립트 부분이 이미 파싱이 완료되어 실행까지 완료된 상태이기 때문입니다.
따라서 아래와 같은 코드는 동작하지 않습니다.

```
// This won't execute
var div = document.querySelector('#some-div');
div.innerHTML = '<script>alert("XSS Attack");</script>';
```

예방 방법

- DomPurify 등을 이용하여, 문자열의 HTML과 CSS를 다음과 같이 sanitize 처리
  sanitization을 거치면 마크업, 스크립트가 아닌 plain string 형태로 출력됨

```
app.innerHTML = '&#60;img src&#61;&#34;x&#34;
onerror&#61;&#34;alert&#40;1&#41;&#34;&#62;';
```

CSP의 종류

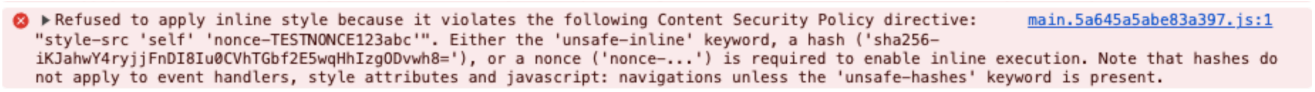
Directives

Directive name	적용되는 리소스	비고
`default-src`	아래 resource fetching에 대한 기본 정책을 정의 - Javascript - Images - CSS - Fonts - AJAX requests - Frames - HTML5 Media	- 모든 directive가 default-src로 fallback 되는 것은 아님
`script-src`	- Javascript	- `<script>`
`style-src`	- stylesheet - CSS	- `<style type="text/css">` - `<link rel="stylesheet">`
`connect-src`	- `XLMHttpRequest` (AJAX) - `WebSocket` - `fetch()` - `<a ping>` - `EventSource`	- CSP에서 차단된 경우, 브라우저 자체적으로 HTTP code `400`을 리턴 (If not allowed the browser emulates a `400` HTTP status code.)
`font-src`	- `@font-face` in CSS	N/A
`media-src`	- `<audio>` - `<video>`	N/A

- 일부 directive는 HTTP header에 선언된 경우에만 동작합니다.
  즉, `<meta>` 태그를 이용한 경우에는 동작하지 않습니다.
  - 예) `report-to`과 같은 CSP 위반 사항을 서버로 보고하는 기능
- 모든 directive 리스트를 확인하시려면 CSP Level 2 W3C Recommendation#directives를 확인해 주시기 바랍니다.

Values

Value	예시	비고
`*`	`img-src *`	- 와일드카드 - `data:`, `blob:`, `filesystem:`, `schemes:`를 제외한 모든 URL을 허용함
`'none'`	`object-src 'none'`	- 어떠한 리소스도 허용하지 않음
`'self'`	`script-src 'self'`	- 동일한 origin(동일 scheme, host, port)에 대해서만 리소스 허용
`data:`	`img-src 'self' data:`	- data scheme(예 : base64 인코딩 이미지)에 대해서만 리소스 허용
`domain.example.com`	`script-src domain.example.com`	- `domain.example.com`가 origin인 script에 대해서만 실행 허용
`*.example.com`	`script-src *.example.com`	- `*.example.com`가 origin인 script에 대해서만 실행 허용
`https:`	`script-src https:`	- `https`로 연결된 origin의 script에 대해서만 실행 허용
`'unsafe-inline'`	`script-src 'unsafe-inline`	- HTML tag의 `style`, `onclick` 또는 inline script를 허용
`'unsafe-eval'`	`script-src 'unsafe-eval'`	- javascript의 `eval()` 함수와 같이, 안전하지 않은 동적 코드 생성 함수 허용
`'unsafe-hashes'`	`script-src 'unsafe-hashes' 'sha256-abc…'`	- event handler(예 : `onclick`)에 연결된 스크립트 허용 - `javascript:`, inline `<script>`에는 적용되지 않음
`'sha256-'`	`script-src 'sha256-xyz…'`	- CSP에 설정한 해시값과 script / CSS파일의 해시값이 일치하는 경우, inline script 또는 CSS를 실행 - **event handler에 bind된 inline-script 또는 style에는 적용되지 않음. 이 경우에는 `nonce`를 사용**
`'nonce-'`	`script-src 'nonce-rAnd0m'`	- `'sha256-'`과 동일



SHAs(hash)는 event handler, style attribute, javascripts:에 적용되지 않는다.

⭐ Value의 경우, URL, `data:`, `blob:`, `filesystem:`, `schemes:`를 제외한 아래의 값은 **따옴표로 감싸줘야 합니다.**

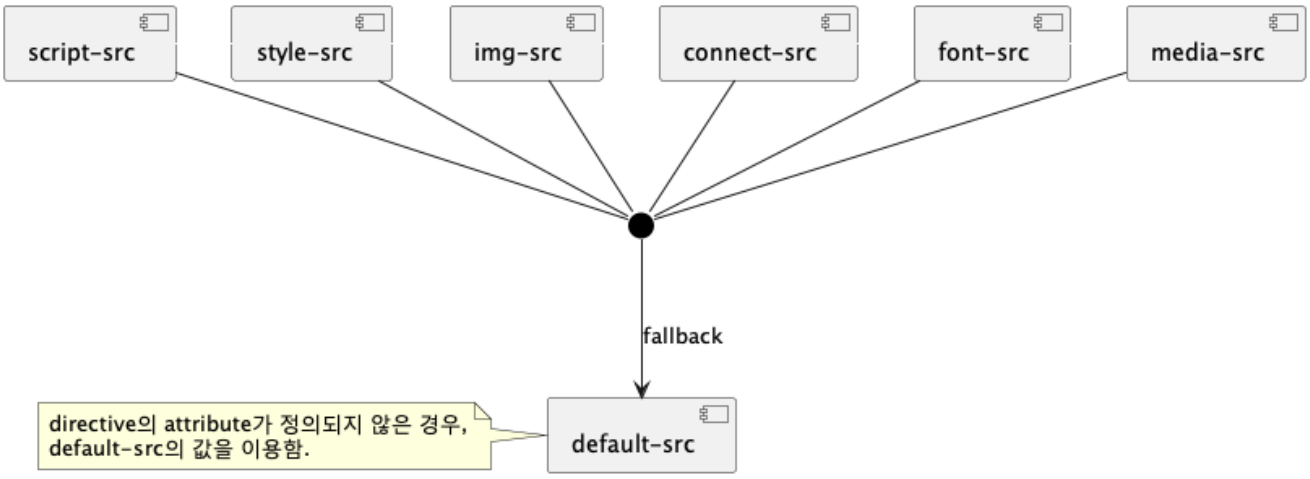
- `'none'`
- `'self'`
- `'unsafe-inline'`
- `'unsafe-eval'`
- `'sha256-'`
- `'nonce-'`
- `'unsafe-hashes'`

❗ 아래 항목은 보안상의 이유로 사용하는 것을 권장하지 않습니다.

~~괜히 `unsafe`가 붙은게 아닙니다.~~

- `'unsafe-inline'`
- `'unsafe-eval'`
- `'unsafe-hashes'`

Fallback



위 사진은 예시로써, 위에서 나열된 6개의 Directive만 default-src로 fallback되는 것을 의미하지 않습니다.

예시

- 아래와 같이 CSP를 설정한 경우, `image-src`, `connect-src`, `font-src`, `media-src`는 `default-src`의 정책을 상속(fallback)받음

```
Content-Security-Policy:
  default-src https:;
  script-src https: 'unsafe-inline';
  style-src https: 'unsafe-inline';
```

```
# 위의 코드는 아래와 같음
Content-Security-Policy:
  default-src https:;
  script-src https: 'unsafe-inline';
  style-src https: 'unsafe-inline';
  # 아래와 같이 명시적으로 작성하지 않은 directive는 default-src의 값을 상속받음
  image-src https:;
  connect-src https:;
  font-src https:;
  media-src https:;
```

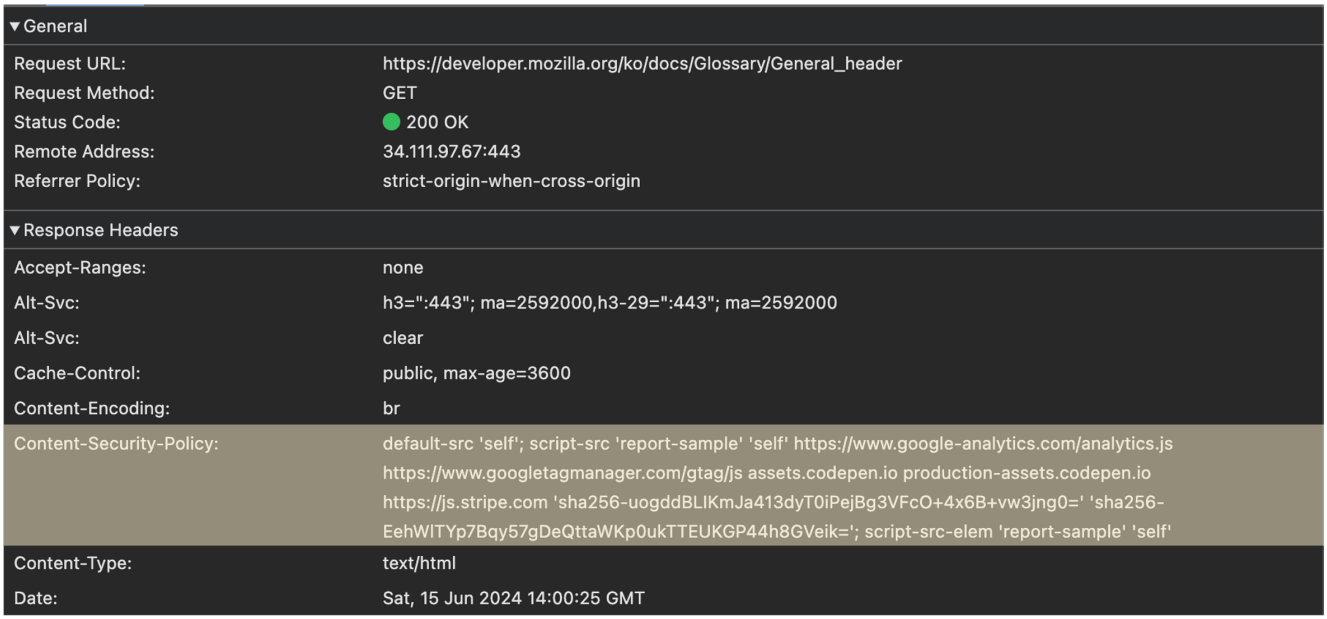
편의상 들여쓰기 및 줄바꿈을 이용하였습니다. 실제 CSP 설정시에는 줄바꿈 및 들여쓰기를 하지 않습니다.
공백 및 세미콜론으로 각 directive 및 value를 구분합니다.

CSP 활성화 방법

Add on HTTP Header

`Content-Security-Policy`를 HTTP header에 추가함

예시

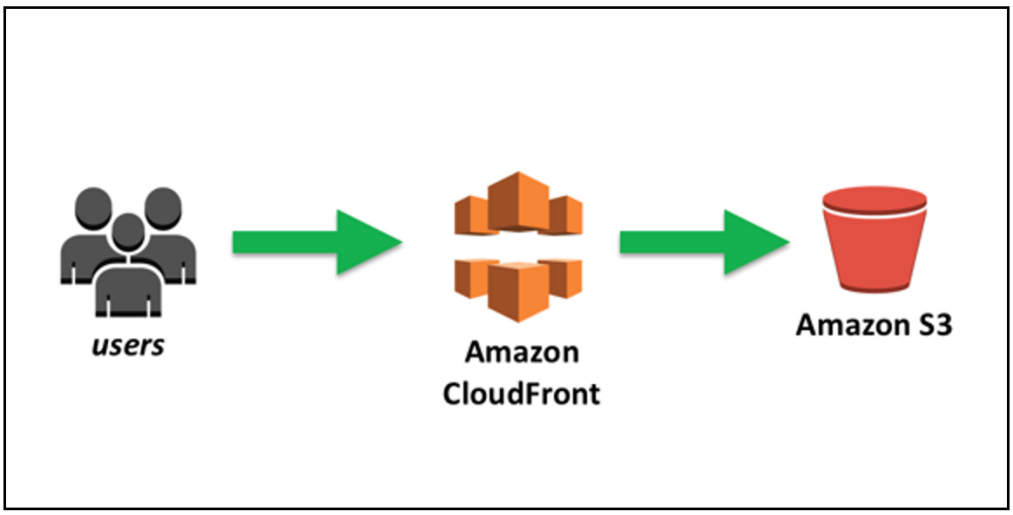


HTTP header에 Content-Security-Policy가 설정된 예시(Mozilla Web Docs)

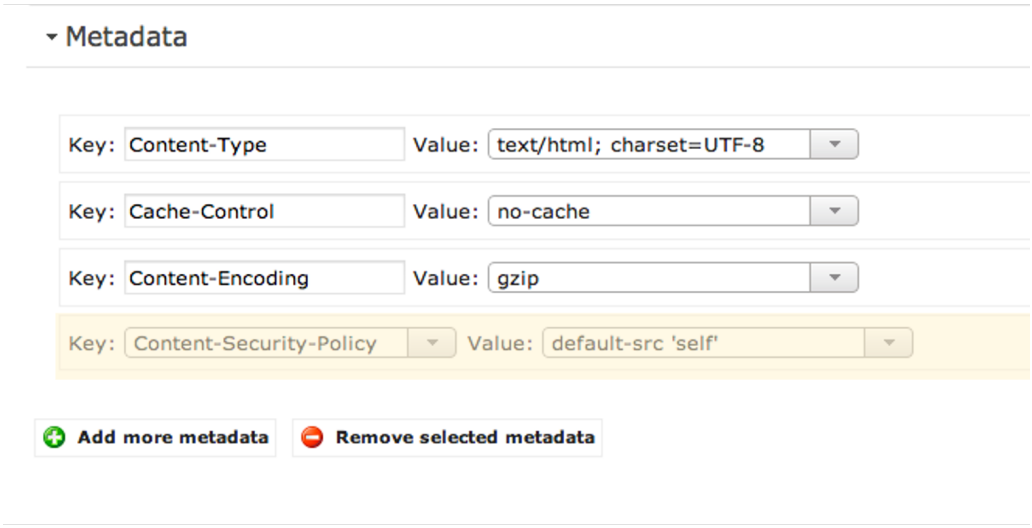
구현방법

AWS CloudFront

- e-Centric Manager가 초반에 사용했던 방법
- 빌드 / 번들링 완료된 html파일을 S3에 업로드하고, 사용자는 AWS CloudFront를 거쳐서 html 파일에 접근
- AWS CloudFront에서 HTTP header를 설정



S3에 올린 html파일을 AWS CloudFront를 이용하여 사용자가 접근 시나리오



AWS CloudFront에서 Content-Security-Policy를 설정하는 부분

express.js 이용

- 만약 html파일을 express.js를 이용하여 사용자에게 전달하는 경우, 다음과 같이 response를 설정

```
res.set("Content-Security-Policy", "default-src 'self'");
```

기타 설정 방법

아래와 같은 서버에서 사용하는 어플리케이션 레이어 레벨의 프로그램에서도 CSP를 설정할 수 있음

- Apache - HTTP application server
- Nginx - proxy server
- NextJS - Server Side Rendering(SSR)

> **Deprecated** : `X-Content-Security-Policy` header는 더 이상 사용하지 않습니다.

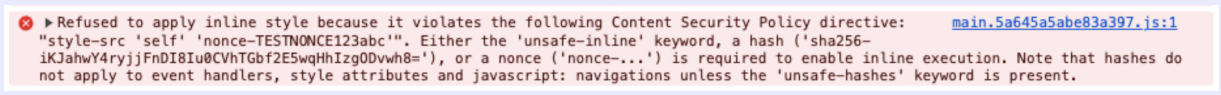
Add `<meta>` element

- 브라우저에서 실행할 html파일에 다음과 같이 `<meta>`를 추가함
- 후술할 내용은 모두 `<head>`에 `<meta>` 태그를 추가하는 방법임. 다만, 태그를 추가하는 stage가 다름

```
<html>
  <head>
    <!-- 중간 내용 생략 -->
    <meta
      http-equiv="Content-Security-Policy"
      content="default-src 'self'; script-src 'self' *.google.com
'sha256-
1e57a452a094728c291bc42bf2bc7eb8d9fd8844d1369da2bf728588b46c4e75';"
    />
  </head>
  <body></body>
</html>
```

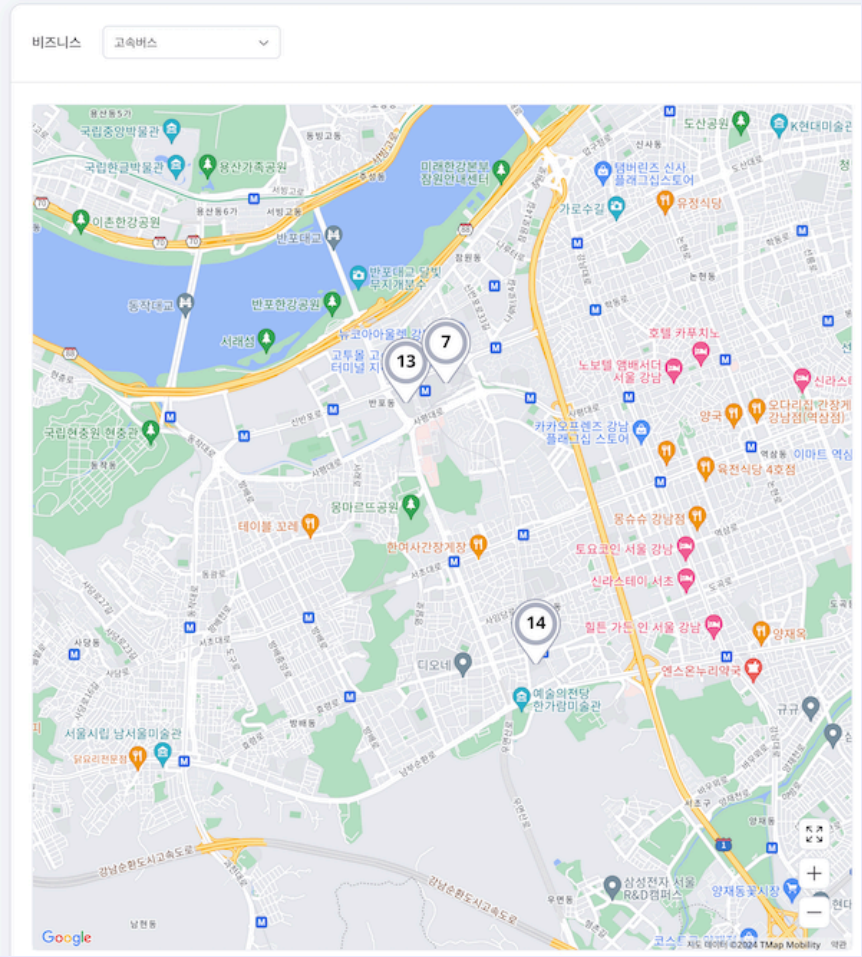
빌드 단계별 CSP 생성 방법(ReactJS 기준)

Build stage	적용 방법	비고
Pre-build stage	`public/index.html`에 직접 `<meta>` 태그 생성	- `'nonce-…'` 사용 불가
Build stage	`html-webpack-plugin(csp-html-webpack-plugin)` 이용 (`webpack` 사용 시)	- `'nonce-…'` 사용 불가
Runtime stage	`react-helmet-async` 이용, 런타임에 CSP 생성	- `'nonce-…'` 사용 불가



SHAs(hash)는 event handler, style attribute, javascripts:에 적용되지 않는다.

"CSP의 종류" 섹션에서 언급했듯이, `'nonce-…'`만 event handler에 bind된 inline-style 또는 script에도 적용됩니다.



e-Centric Manager의 충전지도에서 사용하는 마커가 이러한 event handler에 bind된 경우이다.

즉, event handler에 bind된 inline-style, script가 없다면, Pre-build / Build stage의 방법을 사용하는 것이 훨씬 간단합니다.

⭐ [Content Security Policy] Issue with inline-style - material-ui github issue처럼, 유명한 라이브러리인 경우에도 inline-style을 사용하여 `'nonce-…'` 사용이 강제되는 경우가 있습니다.
사용하는 라이브러리가 inline-style 또는 inline-script를 사용하는지 확인이 필요할 수 있습니다.

- e-Centric Manager의 경우, Jodit Editor의 Element constructor가 `Element.innerHTML`과 inner-style을 사용하여 `'nonce-…'` 사용이 강제되었습니다.

문서에 의하면, CSR에서 *일반적인 방법으로는* 동적으로 Content Security Policy를 설정할 수 있는 방법은 없습니다.

> Note: Modifications to the `content` attribute of a `meta` element after the element has been parsed will be ignored.

참고 : Content Security Policy Level 2 W3C Recommendation#HTML meta Element

~~e-Centric Manager에서 `'nonce-...'` 사용이 어떻게 가능했는지는 논의가 더 필요해 보입니다.~~

→ 디버그 결과, Content-Security-Policy가 포함된 `meta` 태그가 HTML에 렌더링 되기 전에 inline-style이 적용된 컴포넌트가 먼저 렌더링되어 CSP 적용이 되지 않는 것으로 확인되었습니다.

기타

CORS와의 차이점

항목	역할
CORS(Cross-Origin Resource Sharing)	- URL 단위의 whitelist 관리
CSP(Content-Security-Policy)	- URL 단위의 whitelist 관리 예) `http:` `*.com` `'self'` - 파일 단위의 whitelist 관리 예) `'sha256-xyz…'` - event handler에 bind된 style / script 관리 예) `'nonce-rAnd0m'`

CSP, 그래서...해치웠나?

예) Jodit editor를 CDN(unpkg)에서 로드하는 경우

아래와 같이 `<head>` 태그 하위에 `stylesheet`와 `script`를 로드하도록 설정

```
<head>
  <!-- 중간 생략 -->
  <link
    rel="stylesheet"
    href="https://unpkg.com/jodit@4.0.1/es2021/jodit.min.css"
  />
  <script src="https://unpkg.com/jodit@4.0.1/es2021/jodit.min.js">
</script>
</head>
```

위와 같이 CDN에서 라이브러리를 가져오는 경우, 일반적으로 CSP는 다음과 같이 설정함

```
<meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self' https://unpkg.com/jodit@4.0.1/es2021;"
/>
```

그러나 아래와 같이 사용하는 라이브러리가 **다른 라이브러리를 추가로 가져오는** 경우도 존재함

```
// config.js in Jodit
Config.prototype.sourceEditorCDNUrlsJS = [
    'https://cdnjs.cloudflare.com/ajax/libs/ace/1.4.2/ace.js'
];

Config.prototype.beautifyHTMLCDNUrlsJS = [
    'https://cdnjs.cloudflare.com/ajax/libs/js-
beautify/1.14.4/beautify.min.js',
    'https://cdnjs.cloudflare.com/ajax/libs/js-beautify/1.14.4/beautify-
html.min.js'
];
```

이러한 경우, `https://cdnjs.cloudflare.com`은 CSP 정책에 의해 차단되고, 오류 발생함

References

- Content-Security-Policy
  - Content Security Policy(CSP) - mdn web docs
  - Content Security Policy Level 2 W3C Recommendation
  - 콘텐츠 보안 정책(web.dev)
  - Content Security Policy (CSP) Quick Reference Guide)
- Content-Origin Resource Sharing
  - Same-origin policy - mdm web docs
  - CORS란 무엇인가? - velog.io
- XSS
  - Preventing cross-site scripting attacks when using innerHTML in vanilla JavaScript
  - How to sanitize third-party content with vanilla JS to prevent cross-site scripting (XSS) attacks
- Libraries
  - material ui
  - Jodit
- ETC
  - material-ui github issue - [Content Security Policy] Issue with inline-style #19938