

18. 최신 기술 동향

18.1 Web Components (Shadow DOM, Custom Elements)

✓ 개요

Web Components는 HTML/CSS/JavaScript 기반의 **재사용 가능한 UI 컴포넌트**를 만들기 위한 브라우저 표준 기술이다. 이 기술을 활용하면 프레임워크 없이도 자체 태그를 정의하고, 캡슐화된 구조를 만들 수 있다.

Web Components는 크게 3가지 핵심 기술로 구성된다:

1. **Custom Elements** - 사용자 정의 HTML 태그 생성
2. **Shadow DOM** - DOM 구조, 스타일, 이벤트를 캡슐화
3. **HTML Templates** - 반복 구조 및 초기 콘텐츠 선언

📦 1. Custom Elements

사용자가 **새로운 HTML 태그**를 정의할 수 있게 해주는 API.

문법:

```
1 class MyComponent extends HTMLElement {
2   connectedCallback() {
3     this.innerHTML = `<p>Hello, web component!</p>`;
4   }
5 }
6
7 customElements.define('my-component', MyComponent);
```

사용:

```
1 <my-component></my-component>
```

- `connectedCallback()`: 요소가 DOM에 추가될 때 실행
- `disconnectedCallback()`: 요소가 제거될 때
- `attributeChangedCallback()`: 속성 변경 감지

기본적으로 `HTMLElement`를 상속하지만, `<button>` 등의 내장 요소를 확장하려면 `HTMLButtonElement`와 `is="..."`를 사용해야 함.

🌑 2. Shadow DOM

Shadow DOM은 **호스트 요소 내부에 "비공개 DOM 트리"**를 만들고 외부로부터 스타일이나 구조를 **캡슐화**할 수 있는 기능이다.

문법:

```
1 const shadow = this.attachShadow({ mode: 'open' });
2 shadow.innerHTML = `
3   <style>
4     p { color: red; }
5   </style>
6   <p>This is Shadow DOM</p>
7 `;
```

- `mode: "open"`: 외부 JS에서 `.shadowRoot` 접근 가능
- `mode: "closed"`: 외부 JS에서 접근 불가

예시 전체 코드:

```
1 class RedText extends HTMLElement {
2   constructor() {
3     super();
4     const shadow = this.attachShadow({ mode: 'open' });
5     shadow.innerHTML = `
6       <style>
7         p { color: red; }
8       </style>
9       <p>This is Shadow DOM</p>
10    `;
11  }
12 }
13
14 customElements.define('red-text', RedText);
```

```
1 <red-text></red-text>
```

Shadow DOM 안의 스타일은 외부 CSS의 영향을 받지 않으며, 반대로 외부에 영향을 주지도 않음.

✖ 3. HTML Template

`<template>` 태그는 실행되지 않고 DOM에 보이지 않지만, JS로 불러와 사용할 수 있는 재사용 가능한 마크업 블록이다.

```
1 <template id="my-card">
2   <style>
3     .card { border: 1px solid #ccc; padding: 10px; }
4   </style>
5   <div class="card">
6     <slot name="title"></slot>
7     <slot name="content"></slot>
8   </div>
9 </template>
```

```

1 class MyCard extends HTMLElement {
2   constructor() {
3     super();
4     const shadow = this.attachShadow({ mode: 'open' });
5     const template = document.getElementById('my-card').content.cloneNode(true);
6     shadow.appendChild(template);
7   }
8 }
9 customElements.define('my-card', MyCard);

```

```

1 <my-card>
2   <span slot="title"> 📌 제목</span>
3   <p slot="content">이건 콘텐츠입니다.</p>
4 </my-card>

```

🧠 왜 Web Components를 쓰는가?

| 장점 | 설명 |
|----------------------|---|
| ✅ 캡슐화 | 스타일, 구조, 이벤트 독립적 |
| ✅ 재사용성 | 다양한 프로젝트에서 재사용 가능 |
| ✅ 프레임워크 독립 | React, Vue 없이도 가능 |
| ✅ 브라우저 네이티브 성능 | 별도 런타임 없음 |
| ✅ HTML 태그처럼 직관적인 사용법 | <code><my-button></code> 같은 자체 태그 |

! 주의 사항

- 모든 브라우저가 100% 동일하게 지원하지는 않음
 - 대부분 모던 브라우저는 지원 (IE는 미지원)
 - Polyfill 필요 시: <https://github.com/webcomponents/polyfills>
- React/Vue와는 스타일 캡슐화 방식이 다르므로 함께 사용 시 주의
- 속성 변경 감지 시 `observedAttributes()` 구현 필요

📌 실전 활용 분야

- 디자인 시스템 구성 (`<app-button>`, `<app-card>` 등)
- 마이크로 프론트엔드(Micro Frontend)
- 광고/위젯 삽입
- 독립 배포형 UI 컴포넌트

18.2 Server-Side Rendering(SSR)과 HTML 렌더링

✅ 개요

Server-Side Rendering (SSR)은 HTML 콘텐츠를 브라우저가 아닌 서버에서 미리 렌더링해서 사용자에게 전달하는 방식이다.

브라우저는 서버로부터 **완성된 HTML**을 받아 **즉시 화면에 표시**할 수 있으며, 이는 퍼포먼스와 SEO 측면에서 큰 장점을 가진다.

💡 SSR은 전통적인 정적 HTML 페이지의 렌더링 방식과 SPA(클라이언트 렌더링) 사이의 중간 형태라고 볼 수 있다.

🔗 SSR의 렌더링 과정

1. 브라우저가 요청을 보냄
2. 서버는 필요한 데이터를 조회 및 HTML 구성
3. 서버에서 완성된 HTML 문서를 응답
4. 브라우저가 HTML을 바로 렌더링
5. 이후 자바스크립트가 로딩되며 동적 기능이 활성화됨 (Hydration)

📄 예시: 전통적 CSR (Client-Side Rendering)

```
1 <!-- index.html -->
2 <body>
3   <div id="app">Loading...</div>
4   <script src="bundle.js"></script>
5 </body>
```





- 초기 화면이 빈 HTML 또는 “로딩 중” 메시지뿐
- 자바스크립트가 실행되어야 콘텐츠가 보임
- SEO나 느린 네트워크에서 불리함

📄 예시: SSR 응답 HTML





```
1 <!-- 서버에서 생성된 HTML -->
2 <body>
3   <div id="app">
4     <h1>welcome, user!</h1>
5   </div>
6   <script src="bundle.js"></script>
7 </body>
```

- 서버에서 렌더링된 HTML이 초기부터 존재
- JS가 활성화되면 앱이 "Hydrate"됨 (동작 부여됨)

SSR의 장점

| 항목 | 설명 |
|---|---------------------------|
|  빠른 초기 렌더링 | JS 로딩 전에도 사용자에게 콘텐츠가 보임 |
|  SEO 최적화 | 검색 엔진이 콘텐츠를 크롤링하기 쉬움 |
|  저사양 환경 대응 | JS 실행이 느린 장치에서도 UI 노출 가능 |
|  SNS 미리보기 | meta 태그 등 HTML 정보가 미리 존재함 |

SSR의 단점

| 항목 | 설명 |
|---|--------------------------|
|  서버 부하 증가 | 렌더링을 클라이언트가 아닌 서버에서 수행 |
|  응답 지연 가능성 | 모든 요청마다 서버에서 HTML을 새로 생성 |
|  코드 복잡도 증가 | 클라이언트/서버 둘 다 렌더링 고려해야 함 |
|  상태 동기화 어려움 | Hydration 중 이슈 발생 가능 |

CSR vs SSR 비교 요약

| 항목 | CSR (Client) | SSR (Server) |
|------------|---------------------|-----------------------|
| 초기 로딩 속도 | 느릴 수 있음 | 빠름 |
| JS 필요성 | 필수 | 없어도 HTML 출력 가능 |
| SEO | 불리함 | 유리함 |
| 서버 부하 | 적음 | 많음 |
| 인터랙션 응답 속도 | 빠름 (이미 로드된 JS로 처리) | JS Hydration 후에 가능 |
| 사용 사례 | SPA, 대시보드, 내부 시스템 등 | 블로그, 뉴스, 커머스 등 공개 콘텐츠 |

SSR 기술 적용 예시

| 프레임워크 | SSR 지원 방식 |
|-----------------|---|
| Next.js (React) | <code>getServerSideProps()</code> 등으로 SSR |
| Nuxt.js (Vue) | 자동 SSR 또는 SPA 모드 선택 가능 |
| SvelteKit | 기본 SSR + 클라이언트 전환 가능 |

| 프레임워크 | SSR 지원 방식 |
|---------------------------------|---------------------|
| ASP.NET MVC, Django, Spring MVC | 전통적인 SSR 방식 |
| Express + EJS | 템플릿 렌더링으로 SSR 구현 가능 |

Hydration이란?

- SSR로 전달된 HTML을 자바스크립트로 "살리는" 과정
- 이벤트 리스너 연결, 상태 관리 활성화 등을 수행
- React, Vue, Svelte 등은 `hydrate()` 메서드를 통해 동작

```
1 | ReactDOM.hydrate(<App />, document.getElementById('app'));
```

 Hydration 과정에서 서버/클라이언트 HTML 불일치가 발생하면 경고나 오류 발생 가능

HTML 렌더링 전략 요약

| 전략명 | 설명 |
|------------------|---------------------------------------|
| Static Rendering | 미리 HTML 생성 (정적 사이트 생성기: Gatsby, Hugo) |
| Server Rendering | 요청마다 서버에서 HTML 생성 |
| Client Rendering | JS 실행 후에만 콘텐츠 렌더링 |
| Hybrid Rendering | SSR + CSR + ISR 등 혼합 방식 |

SSR이 특히 중요한 경우

- 검색엔진 최적화가 중요한 페이지
- 초기 사용자 경험(첫 페이지 로딩 속도)이 중요한 서비스
- SNS 링크 미리보기/meta 태그 노출 필요
- 콘텐츠 위주 사이트 (뉴스, 블로그, 포트폴리오 등)

18.3 Progressive Web App (PWA) 마크업

개요

Progressive Web App (PWA)는 웹 기술만으로도 앱처럼 동작하는 웹 애플리케이션을 만들기 위한 접근 방식이다. HTML/CSS/JS 기반으로 개발하면서도 아래 기능들을 제공할 수 있다:

- 오프라인 지원
- 홈 화면 설치 (앱처럼)
- 푸시 알림

- 빠른 로딩 (Service Worker 기반 캐시)

💡 핵심 개념: 웹 앱이지만, 네이티브 앱처럼 작동하도록 만들자!

🧩 PWA를 위한 필수 구성 요소 3가지

| 요소 | 설명 |
|-------------------------------|---|
| 1. <code>manifest.json</code> | 앱 정보 정의 (이름, 아이콘, 시작 경로, 색상 등) |
| 2. Service Worker | 오프라인 캐시 및 백그라운드 기능 구현 |
| 3. HTTPS 환경 | 보안 프로토콜은 필수 (Service Worker는 HTTPS에서만 동작) |

📄 1. HTML 마크업에 포함할 기본 설정

```
1 <!DOCTYPE html>
2 <html lang="ko">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6
7   <!-- 📱 앱 설명 및 모바일 최적화 -->
8   <meta name="theme-color" content="#ffffff">
9   <meta name="description" content="이 앱은 PWA 기능을 지원합니다.">
10
11  <!-- 📄 PWA 매니페스트 연결 -->
12  <link rel="manifest" href="/manifest.json">
13
14  <!-- 🍏 iOS 지원 -->
15  <meta name="apple-mobile-web-app-capable" content="yes">
16  <link rel="apple-touch-icon" href="/icons/icon-192x192.png">
17
18  <!-- ✅ 필수 favicon -->
19  <link rel="icon" href="/favicon.ico">
20
21  <title>PWA 마크업 예제</title>
22 </head>
23 <body>
24   <h1>Progressive Web App 예제</h1>
25   <script src="/main.js"></script>
26 </body>
27 </html>
```

2. manifest.json 예시

```
1 {
2   "name": "My PWA App",
3   "short_name": "PWAApp",
4   "description": "앱처럼 동작하는 웹앱입니다.",
5   "start_url": "/index.html",
6   "display": "standalone",
7   "background_color": "#ffffff",
8   "theme_color": "#007bff",
9   "icons": [
10    {
11      "src": "/icons/icon-192x192.png",
12      "sizes": "192x192",
13      "type": "image/png"
14    },
15    {
16      "src": "/icons/icon-512x512.png",
17      "sizes": "512x512",
18      "type": "image/png"
19    }
20  ]
21 }
```

`display: standalone`을 설정하면 주소창 없이 앱처럼 보임
아이콘은 반드시 192x192 이상, 512x512 권장

3. Service Worker 등록 (HTML 또는 JS에서)

```
1 // main.js
2 if ('serviceWorker' in navigator) {
3   window.addEventListener('load', () => {
4     navigator.serviceWorker
5       .register('/sw.js')
6       .then(reg => console.log('✅ Service worker 등록됨:', reg))
7       .catch(err => console.log('❌ 등록 실패:', err));
8   });
9 }
```

4. Service Worker 기본 템플릿 (sw.js)

```
1 self.addEventListener('install', e => {
2   console.log('Service worker 설치됨');
3   e.waitUntil(
4     caches.open('pwa-cache').then(cache => {
5       return cache.addAll([
6         '/',
7         '/index.html',
```



```

8      '/main.js',
9      '/styles.css'
10    ]});
11  })
12  );
13  });
14
15  self.addEventListener('fetch', e => {
16    e.respondWith(
17      caches.match(e.request).then(response => {
18        return response || fetch(e.request);
19      })
20    );
21  });

```

설치 가능하게 만드는 조건

PWA가 "설치 가능" 상태가 되려면 다음 조건을 만족해야 한다:

- `manifest.json` 이 올바르게 연결됨
- HTTPS 환경
- 최소 1개의 Service Worker 등록됨
- `start_url` 과 실제 경로가 일치

Chrome에서는 설치 아이콘이 주소창 우측에 나타남.

테스트 도구

- **Chrome DevTools** → **Application** 탭
 - Manifest, Service Workers 상태 확인
- **Lighthouse** 성능 진단: PWA 최적화 점수 측정

추가 확장 기능

- 푸시 알림 (Push API)
- 백그라운드 동기화
- IndexedDB 활용한 오프라인 저장소

정리: PWA 마크업 체크리스트

| 항목 | 설명 |
|--|-------------|
| <code><link rel="manifest"></code> | manifest 연결 |
| <code><meta name="theme-color"></code> | 앱 UI 색상 지정 |

| 항목 | 설명 |
|--|------------|
| <code><meta name="apple-mobile...</code> | iOS 대응 |
| <code><script></code> 에서 Service Worker 등록 | 오프라인/캐시 기능 |
| HTTPS 환경에서 배포 | 필수 조건 |

18.4 AMP(Accelerated Mobile Pages)

✓ 개요

AMP (Accelerated Mobile Pages)는 구글이 주도한 오픈소스 프로젝트로, **모바일 환경에서 웹 페이지를 즉시 로딩할 수 있도록 최적화된 HTML 프레임워크**다.

목표: "속도와 성능을 극단적으로 최적화한 모바일 웹"

AMP는 캐싱, 미리 렌더링, 제한된 JS 사용 등을 통해 **순식간에 페이지를 보여주는 웹 표준**으로, 주로 뉴스, 블로그, 전자상거래 콘텐츠에서 사용됨.

📦 AMP 구성 요소

AMP 문서는 아래의 세 가지 요소로 구성된다:

1. **AMP HTML**: 제한된 문법을 따르는 HTML5의 서브셋
2. **AMP JS 라이브러리**: 비동기 렌더링을 위한 JS 프레임워크
3. **AMP Cache**: Google CDN을 통한 사전 렌더링 캐시

📄 기본 AMP 마크업 구조

```

1 <!doctype html>
2 <html ⚡ lang="ko">
3   <head>
4     <meta charset="utf-8">
5     <title>AMP 예제 페이지</title>
6     <link rel="canonical" href="https://example.com/page.html">
7     <meta name="viewport" content="width=device-width,minimum-scale=1">
8
9     <!-- AMP 핵심 라이브러리 -->
10    <script async src="https://cdn.ampproject.org/v0.js"></script>
11
12    <!-- AMP 컴포넌트 예시 (이미지 등) -->
13    <script async custom-element="amp-img"
14      src="https://cdn.ampproject.org/v0/amp-img-0.1.js"></script>
15
16    <!-- AMP 스타일 정의 (inline으로만 허용) -->
17    <style amp-boilerplate>
18      body { -webkit-animation: -amp-start 8s steps(1,end) 0s 1 normal both;
19            -moz-animation: -amp-start 8s steps(1,end) 0s 1 normal both;
20            -ms-animation: -amp-start 8s steps(1,end) 0s 1 normal both;
```

```

21         animation: -amp-start 8s steps(1,end) 0s 1 normal both; }
22     </style>
23     <noscript><style amp-boilerplate>body{animation:none}</style></noscript>
24
25     <style amp-custom>
26         body {
27             font-family: sans-serif;
28             padding: 1rem;
29         }
30         h1 {
31             color: #007acc;
32         }
33     </style>
34 </head>
35 <body>
36     <h1>Hello AMP!</h1>
37
38     <!-- AMP 이미지 컴포넌트 -->
39     <amp-img src="/logo.png" width="300" height="100" layout="responsive" alt="logo">
40 </amp-img>
41 </body>
42 </html>

```

🚫 주요 제한 사항

AMP는 성능을 위해 아래와 같은 **제한적 규칙**을 강제한다:

| 항목 | 제한 내용 |
|------------|--|
| JavaScript | 사용자 정의 JS 사용 불가 (AMP JS만 허용) |
| CSS | <code><style amp-custom></code> 으로 75KB 이하만 허용 |
| 외부 리소스 | 반드시 <code>async</code> 로 로드해야 함 |
| 광고/미디어 | AMP 전용 컴포넌트 (<code>amp-ad</code> , <code>amp-video</code> 등) 사용 |
| 레이아웃 | 모든 리소스는 <code>width</code> , <code>height</code> , <code>layout</code> 필수 지정 |

🌟 AMP 전용 컴포넌트 예시

| 컴포넌트 | 설명 |
|-----------------------------------|----------------|
| <code><amp-img></code> | AMP 전용 이미지 렌더링 |
| <code><amp-video></code> | 동영상 삽입 |
| <code><amp-carousel></code> | 슬라이더/캐러셀 |
| <code><amp-form></code> | 폼 처리 |

| 컴포넌트 | 설명 |
|------------------------------------|----------|
| <code><amp-analytics></code> | 추적 코드 삽입 |

🔍 SEO 및 검색 노출 측면

- AMP 페이지는 구글 검색에서 **즉시 로딩(preloaded)** 처리됨
- 모바일 검색에 특화된 캐시 페이지로 우선적으로 제공됨
- `canonical` 태그로 원본 페이지와 연계해야 함

📱 AMP와 일반 웹의 차이

| 항목 | AMP | 일반 HTML |
|--------|-------------------|---------------|
| 성능 | 극단적인 최적화 | 개발자 역량에 따라 다름 |
| JS 사용 | 제한적 (AMP JS만) | 자유 |
| SEO 측면 | 구글 AMP 캐시로 빠르게 노출 | 표준 검색 캐시 방식 |
| 개발 유연성 | 낮음 | 높음 |
| 유지보수성 | 복잡할 수 있음 | 일반적인 웹과 동일 |

📋 AMP 적용 절차

1. 기존 페이지를 AMP 규칙에 맞게 재작성
2. `<link rel="canonical">` 로 원본 페이지 명시
3. `<link rel="amphtml">` 를 원본 페이지에 삽입
4. AMP Validator로 검사
5. Google Search Console에 제출

⚠️ AMP 폐지 논의 및 현재 위치

- 구글은 2021년 이후 **AMP**를 공식 **SEO** 랭킹 요인에서 제외
- Core Web Vitals 중심으로 전환
- 그러나 여전히 일부 뉴스·광고·콘텐츠 플랫폼에서는 사용됨

✓ AMP 도입이 적합한 경우

- 뉴스 기사, 블로그, 콘텐츠 중심 페이지
- 빠른 초기 로딩이 매우 중요한 환경
- Google Discover, News Feed 대상 페이지

18.5 HTML과 웹앱 manifest (manifest.json)

✓ 개요

웹앱 매니페스트(`manifest.json`)는 웹 애플리케이션이 "네이티브 앱처럼" 설치되거나 실행될 때의 메타데이터를 정의하는 **JSON 파일**이다.

이를 통해 웹사이트는 홈 화면에 설치되며, 전체화면 실행, 아이콘, 테마 색상 등 앱스러운 경험을 제공할 수 있다.

💡 `manifest.json`은 Progressive Web App(PWA)의 핵심 구성 요소이며, `<link rel="manifest">`로 HTML에 포함된다.

📁 기본 구성 예시

```
1 {
2   "name": "My Awesome App",
3   "short_name": "AwesomeApp",
4   "start_url": "/index.html",
5   "display": "standalone",
6   "background_color": "#ffffff",
7   "theme_color": "#2196f3",
8   "orientation": "portrait",
9   "lang": "ko-KR",
10  "icons": [
11    {
12      "src": "/icons/icon-192x192.png",
13      "sizes": "192x192",
14      "type": "image/png"
15    },
16    {
17      "src": "/icons/icon-512x512.png",
18      "sizes": "512x512",
19      "type": "image/png"
20    }
21  ]
22 }
```

✂ 주요 속성 설명

| 속성 | 설명 |
|-------------------|---------------------|
| <code>name</code> | 앱의 전체 이름 (설치 시 표시됨) |

| 속성 | 설명 |
|-------------------------------|--|
| <code>short_name</code> | 아이콘 아래에 표시될 줄인 이름 |
| <code>start_url</code> | 앱 실행 시 로드될 시작 URL |
| <code>display</code> | 실행 방식 (<code>fullscreen</code> , <code>standalone</code> , <code>minimal-ui</code> , <code>browser</code>) |
| <code>background_color</code> | 로딩 스크린의 배경색 |
| <code>theme_color</code> | 브라우저 UI의 색상 지정 (주소창 등) |
| <code>orientation</code> | 앱 화면 방향 (<code>portrait</code> , <code>landscape</code>) |
| <code>lang</code> | 앱의 언어 코드 (<code>en</code> , <code>ko-KR</code> 등) |
| <code>icons</code> | 설치용 아이콘 목록 |

아이콘 설정

- PNG 이미지로 지정하며, 다양한 크기를 제공해야 함
- 보통 192x192, 512x512는 필수
- iOS에서는 `<link rel="apple-touch-icon">` 별도 필요

```

1  "icons": [
2    {
3      "src": "/icons/icon-192.png",
4      "sizes": "192x192",
5      "type": "image/png"
6    },
7    {
8      "src": "/icons/icon-512.png",
9      "sizes": "512x512",
10     "type": "image/png"
11   }
12 ]

```

HTML에 포함하는 방법

```
1 | <link rel="manifest" href="/manifest.json">
```

- ✦ 보통 `<head>` 안에 위치시킴
- ✦ 반드시 **HTTPS 환경**에서 동작

실행 방식 `display` 설명

| 값 | 설명 |
|-------------------------|----------------------|
| <code>fullscreen</code> | 완전한 전체 화면 (상태바 없음) |
| <code>standalone</code> | 브라우저 UI 없이, 네이티브 앱처럼 |
| <code>minimal-ui</code> | 주소창만 있는 최소한의 UI |
| <code>browser</code> | 일반 브라우저처럼 동작 (기본값) |

테스트 방법

- 크롬 개발자 도구 → Application 탭 → Manifest 섹션 확인
- 모바일 Chrome에서 “홈 화면에 추가” 기능 테스트
- Lighthouse PWA 점검 → "Web app manifest exists" 체크 항목

고급 속성들

| 속성 | 설명 |
|--|--|
| <code>scope</code> | 앱의 URL 접근 범위 제한 |
| <code>dir</code> | 텍스트 방향 (<code>ltr</code> , <code>rtl</code>) |
| <code>prefer_related_applications</code> | 네이티브 앱 우선 설치 유도 가능 |
| <code>related_applications</code> | 스토어 정보 등록 (Play Store 등) |
| <code>description</code> | 앱 설명 |
| <code>screenshots</code> | 스토어처럼 앱 미리보기 등록 |

CORS 설정 주의

만약 `manifest.json` 이 다른 도메인에 있다면:

```
1 <link rel="manifest" href="https://cdn.example.com/app.webmanifest" crossorigin="use-credentials">
```

✓ 실전 체크리스트

| 항목 | 확인 여부 |
|---|-------|
| <code>manifest.json</code> 작성 완료 | ✓ |
| HTML에 <code><link rel="manifest"></code> 포함 | ✓ |
| 아이콘 이미지 192px, 512px 이상 제공 | ✓ |
| <code>theme_color</code> , <code>background_color</code> 설정 | ✓ |
| <code>start_url</code> 와 앱 시작 경로 일치 | ✓ |
| HTTPS 환경에서 테스트 | ✓ |

■ 관련 툴

- [PWA Asset Generator](#)
- [Web App Manifest Generator](#)