



Vue 사용 방법

우리는 웹에 "모두에게 맞는 하나의 정답"이 있다고 생각하지 않습니다. 그래서 Vue는 유연하고 점진적으로 도입할 수 있도록 설계되었습니다. 사용 사례에 따라 Vue는 스택의 복잡성, 개발자 경험, 최종 성능 사이에서 최적의 균형을 이룰 수 있도록 다양한 방식으로 사용할 수 있습니다.

독립형 스크립트

Vue는 독립형 스크립트 파일로 사용할 수 있습니다. 빌드 단계가 필요하지 않습니다! 이미 백엔드 프레임워크가 대부분의 HTML을 렌더링하고 있거나, 프론트엔드 로직이 빌드 단계를 정당화할 만큼 복잡하지 않다면, 이 방법이 Vue를 스택에 통합하는 가장 쉬운 방법입니다. 이런 경우 Vue를 jQuery의 더 선언적인 대체재로 생각할 수 있습니다.

이전에는 기존 HTML을 점진적으로 향상시키는 데 특화된 **petite-vue**라는 대체 배포판을 제공했지만, **petite-vue**는 더 이상 적극적으로 유지 관리되지 않으며, 마지막 버전은 Vue 3.2.27에서 배포되었습니다.

내장 웹 컴포넌트

Vue를 사용하여 **표준 웹 컴포넌트**를 만들 수 있으며, 이 컴포넌트는 렌더링 방식에 상관없이 어떤 HTML 페이지에도 삽입할 수 있습니다. 이 옵션을 사용하면 Vue를 완전히 소비자에 독립적인 방식으로 활용할 수 있습니다. 결과 웹 컴포넌트는 레거시 애플리케이션, 정적 HTML, 또는 다른 프레임워크로 구축된 애플리케이션에도 삽입할 수 있습니다.

싱글 페이지 애플리케이션(SPA)



반 로직이 필요합니다. 이러한 애플리케이션을 구축하는 가장 좋은 방법은 Vue가 전체 페이지를 제어할 뿐만 아니라, 데이터 업데이트와 내비게이션도 페이지를 새로 고침하지 않고 처리하는 아키텍처를 사용하는 것입니다. 이러한 유형의 애플리케이션을 일반적으로 싱글 페이지 애플리케이션(SPA)이라고 합니다.

Vue는 현대적인 SPA를 구축하기 위한 핵심 라이브러리와 포괄적인 툴링 지원을 제공하며, 뛰어난 개발자 경험을 제공합니다. 여기에는 다음이 포함됩니다:

- 클라이언트 사이드 라우터
- 매우 빠른 빌드 도구 체인
- IDE 지원
- 브라우저 개발자 도구
- TypeScript 통합
- 테스트 유틸리티

SPA는 일반적으로 백엔드가 API 엔드포인트를 노출해야 하지만, [Inertia.js](#)와 같은 솔루션과 Vue를 결합하여 서버 중심 개발 모델을 유지하면서도 SPA의 이점을 누릴 수 있습니다.

풀스택 / SSR

순수 클라이언트 사이드 SPA는 앱이 SEO와 콘텐츠 표시 시간에 민감할 때 문제가 될 수 있습니다. 이는 브라우저가 대부분 비어 있는 HTML 페이지를 받게 되고, JavaScript가 로드될 때까지 아무것도 렌더링하지 못하기 때문입니다.

Vue는 Vue 앱을 서버에서 HTML 문자열로 "렌더링"할 수 있는 일급 API를 제공합니다. 이를 통해 서버는 이미 렌더링된 HTML을 반환할 수 있어, 최종 사용자는 JavaScript가 다운로드되는 동안에도 즉시 콘텐츠를 볼 수 있습니다. 이후 Vue는 클라이언트 측에서 애플리케이션을 "하이드레이트"하여 상호작용이 가능하게 만듭니다. 이를 **서버 사이드 렌더링(SSR)**이라고 하며, **Largest Contentful Paint (LCP)**와 같은 Core Web Vital 지표를 크게 개선합니다.

이 패러다임 위에 구축된 더 높은 수준의 Vue 기반 프레임워크도 있습니다. 예를 들어 **Nuxt**는 Vue와 JavaScript를 사용하여 풀스택 애플리케이션을 개발할 수 있게 해줍니다.

JAMStack / SSG

필요한 데이터가 정적이라면 서버 사이드 렌더링을 미리 수행할 수 있습니다. 즉, 전체 애플리케이션을 HTML로 미리 렌더링하여 정적 파일로 제공할 수 있습니다. 이렇게 하면 사이트 성능이 향상되고, 각 요청마다 동적으로 페이지를 렌더링할 필요가 없으므로 배포도 훨씬 간단해집니다.



있습니다. 이 기술은 일반적으로 정적 사이트 생성(SSG)이라고 하며, **JAMStack**이라고도 불립니다.

SSG에는 싱글 페이지와 멀티 페이지 두 가지 방식이 있습니다. 두 방식 모두 사이트를 정적 HTML로 미리 렌더링하지만, 차이점은 다음과 같습니다:

초기 페이지 로드 후, 싱글 페이지 SSG는 페이지를 SPA로 "하이드레이트"합니다. 이 방식은 더 많은 초기 JS 페이로드와 하이드레이션 비용이 들지만, 이후 내비게이션은 더 빨라집니다. 전체 페이지를 다시 로드하는 대신 페이지 콘텐츠만 부분적으로 업데이트하면 되기 때문입니다.

멀티 페이지 SSG는 내비게이션할 때마다 새 페이지를 로드합니다. 장점은 최소한의 JS만 제공하거나, 상호작용이 필요 없는 경우 JS를 아예 제공하지 않아도 된다는 점입니다! **Astro**와 같은 일부 멀티 페이지 SSG 프레임워크는 "부분 하이드레이션"도 지원합니다. 이를 통해 정적 HTML 내부에 Vue 컴포넌트를 사용하여 상호작용이 가능한 "아일랜드"를 만들 수 있습니다.

비트리비얼한 상호작용, 긴 세션, 내비게이션 간에 유지되는 요소/상태가 필요하다면 싱글 페이지 SSG가 더 적합합니다. 그렇지 않다면 멀티 페이지 SSG가 더 나은 선택이 될 수 있습니다.

Vue 팀은 **VitePress**라는 정적 사이트 생성기도 유지 관리하고 있습니다. 이 사이트도 VitePress로 구동되고 있습니다! VitePress는 두 가지 SSG 방식을 모두 지원합니다. **Nuxt**도 SSG를 지원합니다. 같은 Nuxt 앱 내에서 라우트별로 SSR과 SSG를 혼합해서 사용할 수도 있습니다.

웹을 넘어서

Vue는 주로 웹 애플리케이션을 구축하기 위해 설계되었지만, 브라우저에만 국한되지 않습니다. 다음과 같은 작업이 가능합니다:

Electron 또는 **Wails**로 데스크톱 앱 만들기

Ionic Vue로 모바일 앱 만들기

Quasar 또는 **Tauri**로 동일한 코드베이스에서 데스크톱 및 모바일 앱 만들기

TresJS로 3D WebGL 경험 만들기

Vue의 **Custom Renderer API**를 사용하여 터미널용과 같은 커스텀 렌더러 만들기!

[GitHub에서 이 페이지 편집](#)

