

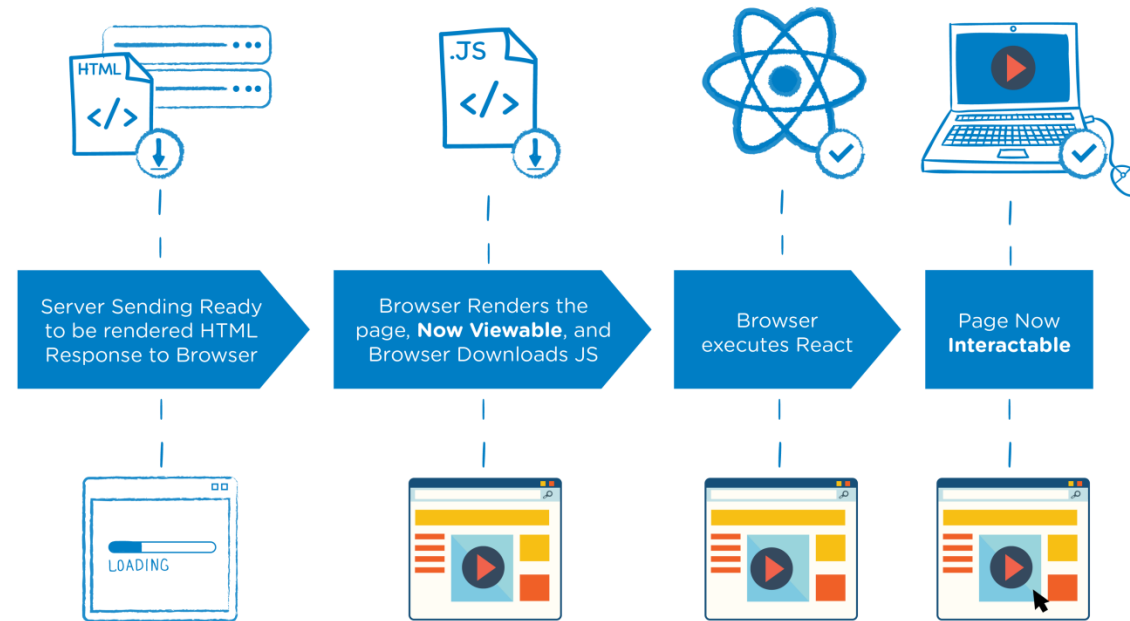


SSR과 CSR

Server Side & Client Side Rendering

조새나라

SSR



SSR이란?

Server Side Rendering의 약자로 사용자의 브라우저에서 구성하지 않고 서버에서 사용자에게 보여줄 페이지를 모두 구성하여 보여주는 방식. JSP/Servlet의 아키텍처에서 이 방식을 사용함.



SSR

SSR의 장점

- 페이지를 더 빨리 로드 할 수 있음.
- 사용자의 인터넷 연결 속도나 장치가 느릴 때 페이지 로드를 도와줌.
- *SEO (검색 엔진 최적화)에서 중요한 역할을 하며 웹 페이지를 색인화함.
구글과 같은 검색 엔진 사이트의 웹 크롤러가 쉽게 해석할 수 있는 구조.

SSR의 단점

- 복잡도와 크기가 큰 페이지를 렌더링할 때 시간이 오래 걸리고 서버 측에서 단일 병목 현상으로 인해 오히려 페이지 로드가 느려질 수 있음.
- 페이지 이동 시 화면 깜빡임이 필요함.
- 모든 요청 시 마다 모든 요소를 로딩하고 렌더링 하므로 불필요한 자원 소모.





SEO

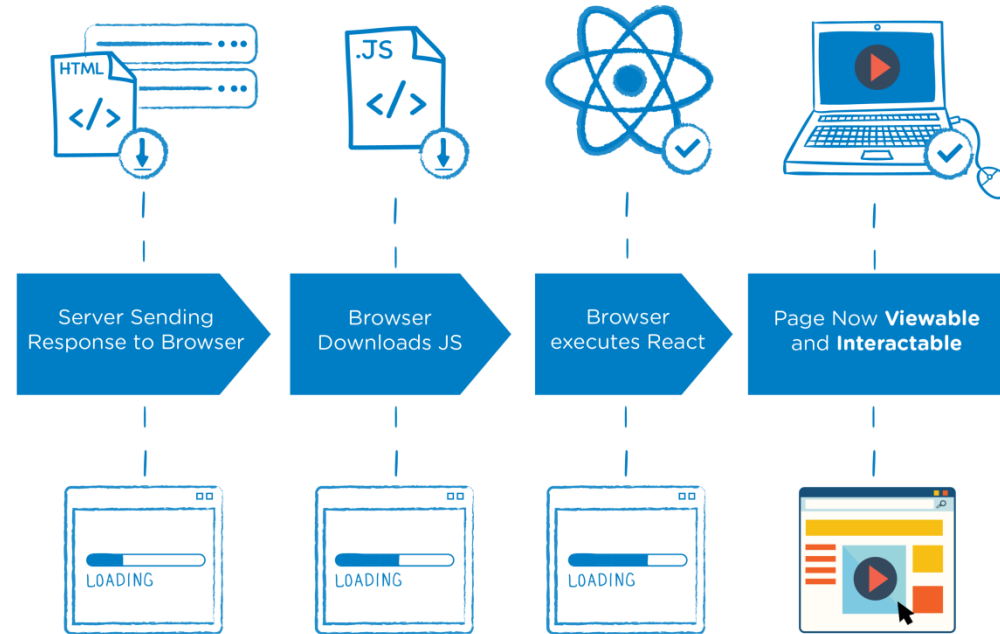
Search Engine Optimization, 검색 엔진 최적화의 약자로
구글과 같은 검색엔진의 웹 크롤러 봇이 페이지를 크롤링 하는 데 도움을 주는 것.

웹 크롤러 봇은 자바 스크립트를 읽지 않기 때문에 완성된 형태의 템플릿에 가까울
수록 웹 크롤러가 읽어오기 수월하며 이렇게 구성된 페이지일 수록 검색엔진이
해당 콘텐츠를 더 강하게 찾아오게 됨.

즉 구글과 같은 검색엔진에서 키워드를 검색했을 때 SEO를 고려하여 만들었다면
별도의 마케팅이 없어도 최상단에 검색될 수도 있음.



CSR



CSR이란?

Client Side Rendering의 약자로 JavaScript를 사용하여 브라우저에서 페이지를 직접 렌더링 하는 것. 모든 로직과 데이터, 템플릿 및 라우팅이 서버가 아닌 클라이언트에서 처리됨. DOM과 밀접하게 관계가 있으며 주로 SPA가 CSR로 되어있음.



CSR

CSR의 장점

- 전체를 요청하지 않고 필요한 부분만 요청하기 때문에 렌더링 속도가 빠름.
- 사용자의 행동에 따라 필요한 부분만 다시 읽으므로 빠른 인터랙션 가능.
- 코드 스플리팅을 활용하여 Lazy Loading 기능 활용 가능.
- SSR보다 일관성 있는 코드 작성이 가능.

CSR의 단점

- 일반적으로 SEO가 전혀 고려되지 않았으므로 검색엔진에 노출이 되지 않음.
- 페이지를 읽고 Js를 통해 화면을 그리는 시간까지 모두 마쳐야 콘텐츠가 사용자에게 보이므로 초기구동 속도가 느림.
- 클라이언트에서 자료 이동이 빈번하므로 보안 이슈가 커짐.





SSR vs CSR

이론적으로 SSR이 CSR보다 빠를 수 밖에 없기 때문에 성능이 우수하지만 모든 면에서 SSR이 뛰어나진 않다. 일반적으로 개발 시 CSR이 고려할 것이 적다.

현재는 React와 같은 프론트엔드 프레임워크와 Node.js 기반의 기술들 그리고 웹팩과 같은 모듈 번들링 기술들이 등장하며 서로가 서로의 장점을 보유하게 되고 서로를 이용하여 서로의 단점이 보완하여 개발이 가능하게끔 변화하고 있다.



SSR

Home Page



Category Page



Search Page



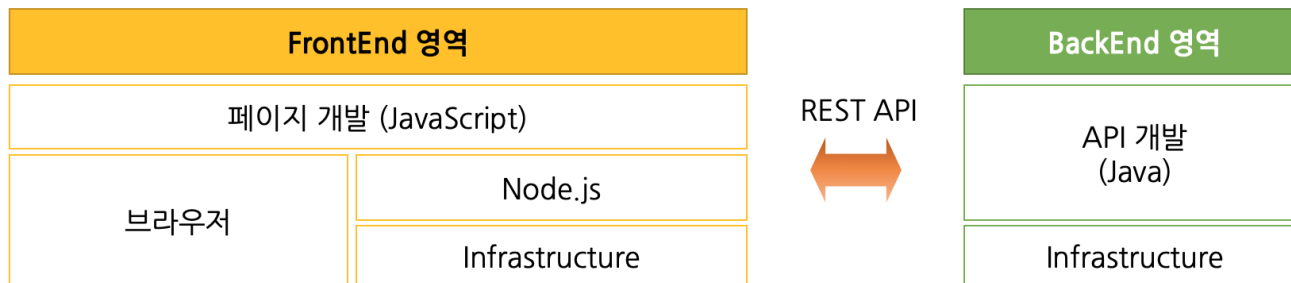
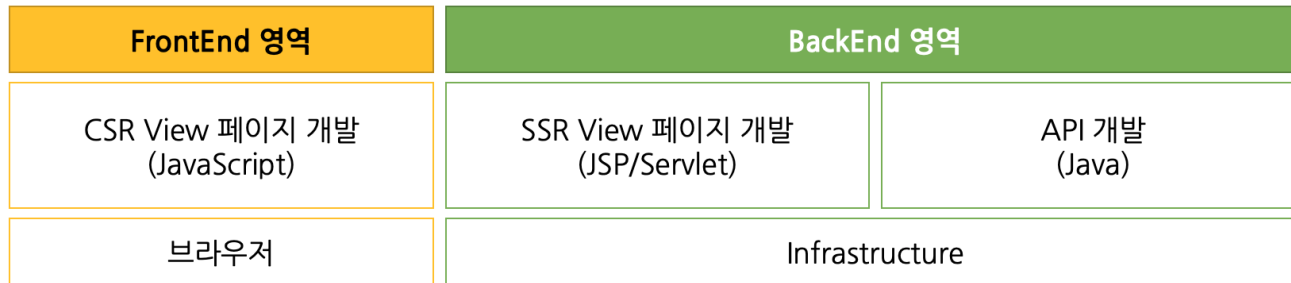
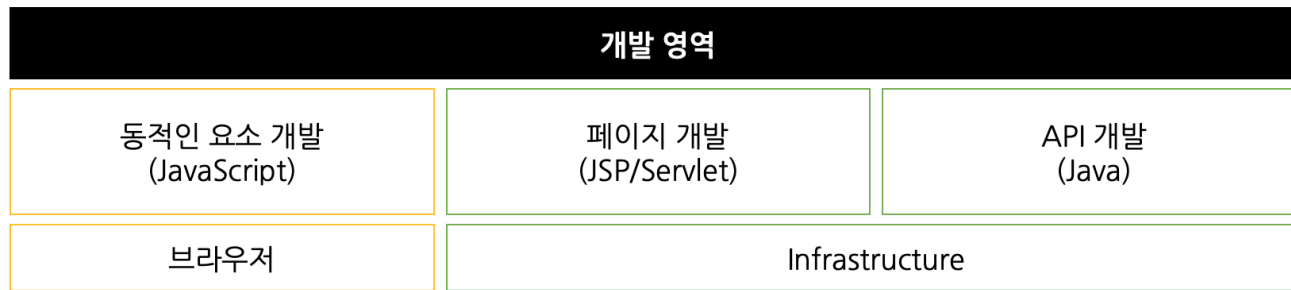
CSR



위는 SSR vs CSR로 렌더링 된 walmart.com의 실제 이용 사례.

각 페이지 별로 시간대마다 렌더링 된 화면이 확인 가능한 데
기능이 많을 수록 최초 응답 시간에서 CSR이 장점을 가지고 있지만
SSR의 렌더링이 빠르고 CSR은 빈 페이지를 띄우고 시작하는 단점이 눈에 띈다.





네이버 블로그 개발팀이 변화해온 개발 형태.
모바일 시대의 전환과 함께 Anuglar.js가 태동하기
시작하며 SPA로의 전환을 시작하였고.



이때 본격적으로 Front-End와 Back-End가
분리되었으며 CSR과 SSR을 병행하여 개발하게 됨.

2016년 이후 Front-End 생태계의 급속한 발전과
함께 CSR의 문제점이 대두되며 SSR로의 완전한
전환이 이뤄짐.

Rest Api를 통한 프론트와 백간의 느슨한 연결과
Node.js와 React를 통한 SSR 개발 환경을 구축,
개발 영역을 분리하여 생산성도 증대하고
전체적인 페이지 응답 속도의 향상도 이뤄냄.

현재는 SSR 기반에서 필요에 따라 CSR을 이용하는
형태로 변모함.



	Server 	←-----→			Browser 
	Server Rendering	“Static SSR”	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is removed .	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side <small>(request-response, HTML)</small>	Built as if client-side <small>(components, DOM*, fetch)</small>	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML and JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. <small>(thin client)</small>	Delivers static HTML	Renders pages <small>(navigation requests)</small>	Delivers static HTML	Delivers static HTML
Pros:	👍 TTI = FCP 👍 Fully streaming	👍 Fast TTFB 👍 TTI = FCP 👍 Fully streaming	👍 Flexible	👍 Flexible 👍 Fast TTFB	👍 Flexible 👍 Fast TTFB
Cons:	👎 Slow TTFB 👎 Inflexible	👎 Inflexible 👎 Leads to hydration	👎 Slow TTFB 👎 TTI >>> FCP 👎 Usually buffered	👎 TTI > FCP 👎 Limited streaming	👎 TTI >>> FCP 👎 No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	Next.js , Razzle , etc	Gatsby, Vuepress, etc	Most apps



결론

과거 SPA = CSR, 현재 SPA \supseteq (CSR, SSR).

현재는 SSR과 CSR의 완벽한 분리보단 사용자 경험(UX)과 SEO, 용도 등에 따라 SSR과 CSR이 혼용되며 사용되고 있다.

그러나 한 가지 확실한 것은 Node.js가 자리잡고 프론트엔드 프레임워크, 특히 React의 생태계가 엄청나게 확장되고 프론트와 백의 분리가 이뤄지며 SSR 기반에서의 개발이 주류가 되어 이뤄지고 있다.

그러나 SSR 기반의 개발은 프론트엔드 방면의 높은 이해도와 "새로운 개발 환경" 요구하므로 좋은걸 알면서도 도입하기가 쉽지가 않다.

더군다나 Prerendering 기법을 통해 CSR에서도 SEO를 고려하여 개발하고 SSR을 이용할 수 있기에 둘 사이의 확실한 우위에 대해선 논쟁의 여지가 크다.

물론 SSR과 CSR에 대해서 논의하기 전에 리소스에 대한 모듈 단위 처리와 번들링을 수행하지도 않고 프레임워크의 필요성을 모르고 있다면 소 귀의 경 입기이며 논의 자체가 아무런 의미가 없다고 볼 수도 있다.



감사합니다.

- 출처

어서 와, SSR은 처음이지? - 도입 편

<https://d2.naver.com/helloworld/7804182>

The Benefits of Server Side Rendering Over Client Side Rendering

<https://medium.com/walmartglobaltech/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8>

Rendering on the Web

<https://developers.google.com/web/updates/2019/02/rendering-on-the-web?hl=ko>

Client-side vs. Server-side vs. Pre-rendering for Web Apps

<https://www.toptal.com/front-end/client-side-vs-server-side-pre-rendering>

