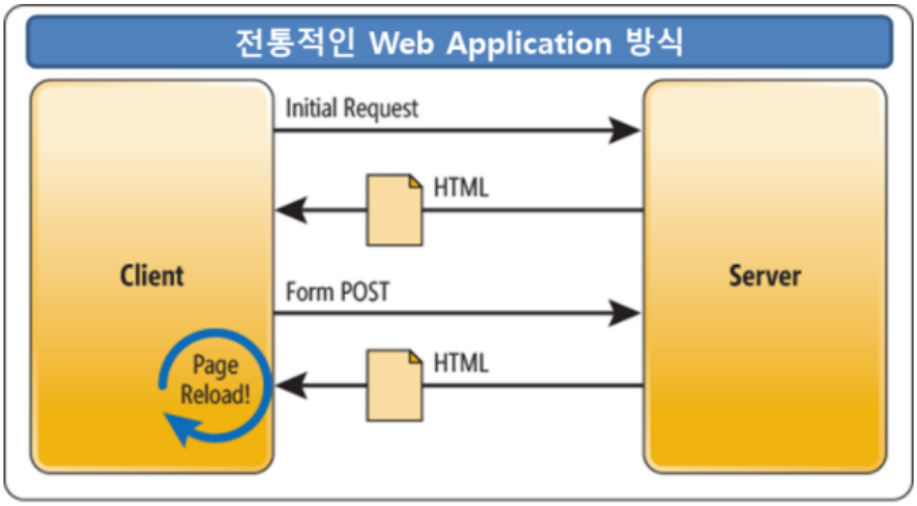
[전통적인 Web Application VS SPA(Single Page Application) Web Application 비교]

◈ 전통적인 Web Application 구현 방식

전통적인 웹 방식은 새로운 페이지를 요청할 때마다 정적 리소스가 다운로드 되고 전체 페이지를 다시 렌더링하는 방식을 사용하므로 새로고침 발생 시 사용성이 좋지 않으며, 변경이 필요없는 부분을 포함하여 전체 페이지를 갱신하므로 비효율적인 측면이 있습니다.

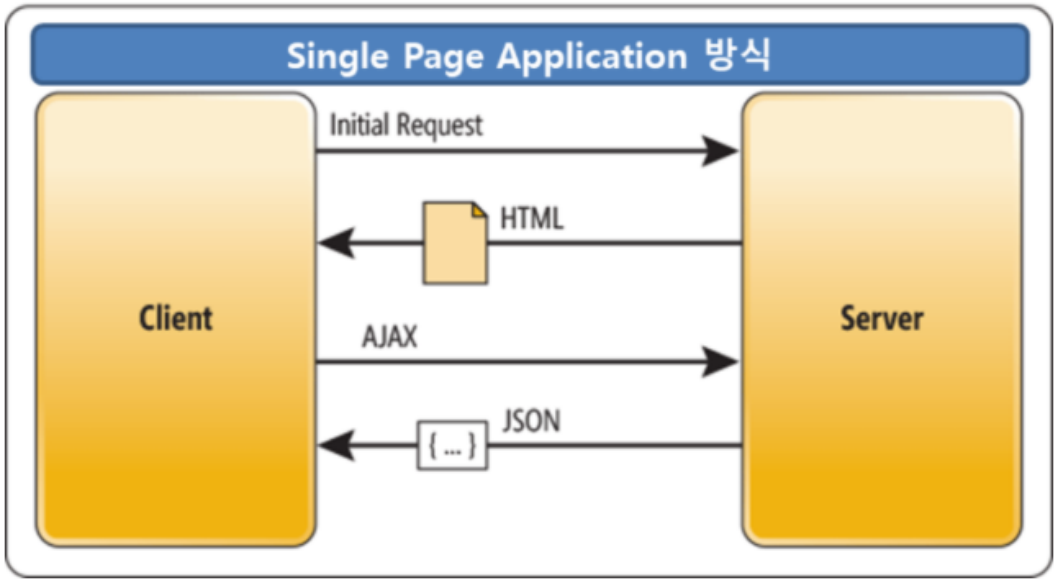


◈ SPA(Single Page Application) 구현 방식

SPA(Single Page Application) 구현 방식은 서버로부터 새로운 **페이지**를 불러오지 않고 현재의 **페이지**를 동적으로 다시 작성함으로써 사용자와 소통하는 웹 **애플리케이션**이나 웹사이트를 말합니다.

SPA에서 HTML, 자바스크립트, CSS 등 필요한 모든 코드는 하나의 페이지로 불러오거나, 적절한 자원들을 동적으로 불러들여서 필요하면 문서에 추가하는 등 사용자의 요청에 응답하는 형식으로 처리되는 방식입니다.

즉, 첫 페이지 요청시 단 한 번만 리소스(HTML, CSS, JavaScript)를 로딩하고, 그 이후로는 페이지 리로딩 없이 필요한 부분만 서버로부터 받아서 화면을 갱신합니다. 필요한 부분만 갱신하기 때문에 네이티브 앱에 가까운 자연스러운 페이지 이동과 사용자 경험(UX)을 제공할 수 있습니다.



[본 자료의 무단 전재 및 배포 금지]

◈ **SPA(Single Page Application)** 장점

1) 클라이언트가 모든 페이지를 가지고 있으므로 앱과 같은 자연스러운 사용자 경험을 제공합니다.
2) 페이지를 이동 하더라도 필요한 부분 (컴포넌트)만 부분적으로 교체하면 되므로 효율성이 증가합니다.
3) 서버가 해야 할 화면 구성을 클라이언트가 수행하므로 서버 부담이 경감됩니다.
4) 모듈화 또는 컴포넌트별 개발이 용이합니다.
5) 백엔드와 프론트엔드가 비교적 명확하게 구분됩니다.

SPA는 일단 로딩이 된 후에는 네이티브 앱과 비슷한 수준의 향상된 사용자 경험(UX)을 제공할 수 있습니다. 중복되는 리소스 없이 필요한 요청만 하게 되므로 효율성이 증가하는 측면도 있습니다.

또한 최근의 프론트엔드 라이브러리는 HTML 히스토리 모드를 지원하므로, SPA에서도 자연스러운 URL 이동을 구현할 수 있습니다.

서버는 초기 파일 전송 후에는 API 처리만 해줘도 됩니다. 서버 측면에서 웹과 모바일 앱에 공통으로 적용할 여지가 많아집니다. 앞단에서 NginX로 Static 파일을 처리하고 필요한 데이터는 API를 통해서 뒤에서 Node.JS 등으로 처리하는 방식도 많이 사용됩니다.

◈ SPA 단점

1) 처음 접속시 사이트 구성과 관련된 모든 리소스를 한 번에 받기 때문에 초기 구동 속도가 느릴 수 있습니다.
2) 데이터를 별도로 요청하고 받아와서 화면을 구성하므로 설계 방식에 따라서는 화면이 변하는 모습이 사용자에게 노출될 수 있습니다. 별로 보기 좋지는 않습니다.
3) SPA 구조 상 데이터 처리는 클라이언트에서 하는 경우가 많은데, 중요한 비즈니스 로직이 존재한다면 사용자가 굳이 파헤쳐 보겠다 하면 막기 어렵습니다.

앞서, 1번과 2번 단점은 lazy loading, pre-rendering을 통해서 어느 정도 극복이 가능합니다.
3번과 같은 경우, 노출되어서는 안 되는 중요한 로직이 있다면 해당 기능만 API를 통해서 서버에서 처리하도록 하면 됩니다. 다소 비효율적이긴 하더라도 큰 문제는 아닙니다.

[참고 웹 사이트]
https://m.blog.naver.com/dktmrorl/222085340333
https://paperblock.tistory.com/87

[본 자료의 무단 전재 및 배포 금지]