단일 페이지 웹 어플리케이션

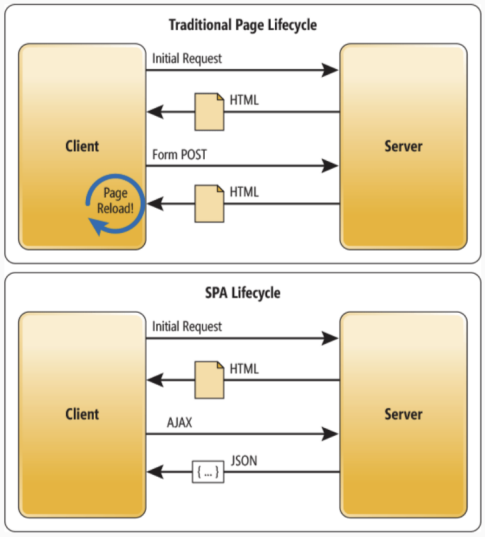
단일 페이지 웹 어플리케이션

Single-Page Application

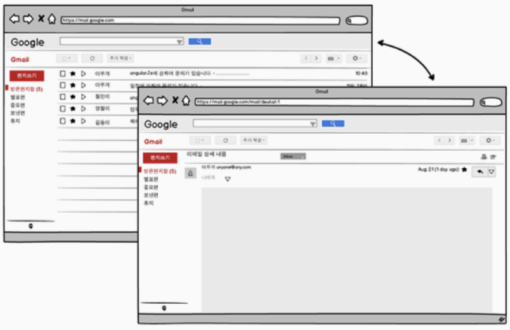
단일 페이지 웹 애플리케이션

사용자 인터랙션에 의해 URL이 변경 시 화면 전체의 로드가 없이 화면의 일부분만 동적으로 렌더링하여 데스크탑 어플리에이션과 비슷한 유저경험을 제공한다.

- AJAX를 이용해서 대부분의 리소스(HTML, CSS, Script)들은 어플리케이션 로드시 한번 읽는다.
- JSON과 같은 데이터만 어플리케이션 실행중에 읽어오고 관련된 화면을 변경시킨다.

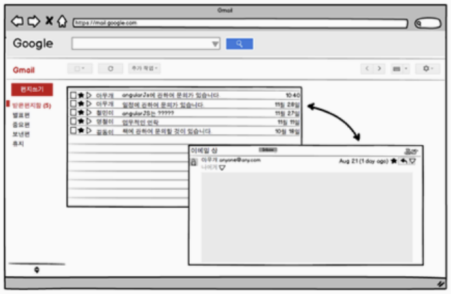


출처: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx



일반 웹 애플리케이션

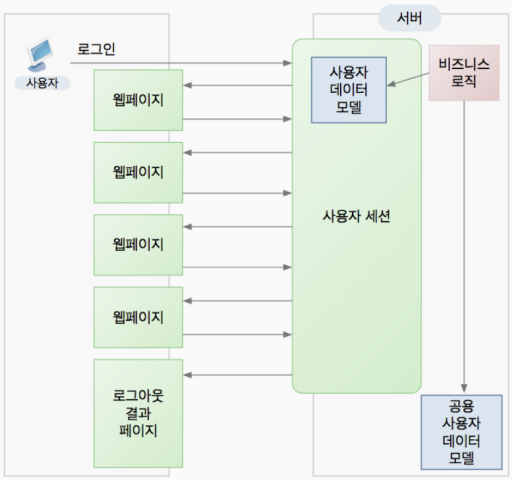
URL 변경시 모든 페이지가 다시 렌더링 된다.
자바스크트와 CSS들을 다시 서버로 요청하게 된다.



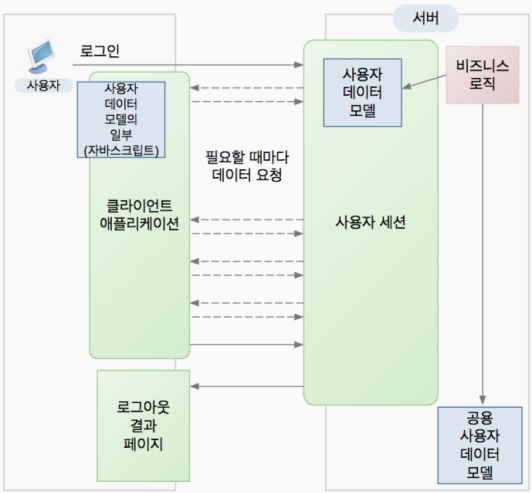
단일 페이지 웹 애플리케이션

URL 변경시 특정 영역만 렌더링 된다. 주로 이때 DOM 조작을 통하여 렌더링이 이루어지게 된다.

시간흐름에 따른 페이지 로딩절차 비교



일반적인 웹 페이지 로딩 절차



SPA 기반 페이지 로딩 절차

서버사이드 라우팅 처리

- 요청받는 URL에 따른 리소스를 반환
- 일반적인 웹사이트에서 사용자의 URL에 해당하는 웹 페이지를 반환하는 행위로 볼 수 있다.

```
@GetMapping("/customers/{customerId}")
public String findCustomerPage(
  @PathVariable("customerId") UUID customerId,
  Model model) {
 var customer = customerService.getCustomer(customerId);
 if (customer.isPresent()) {
   model.addAttribute("customer", customer.get());
   return "views/customer-details";
 } else {
   return "views/404";
 }
}

@GetMapping("/customers/new")
public String viewNewCustomerPage() {
 return "views/new-customer";
}
```

클라이언트 라우팅 처리

- SPA 특성상 클라이언트에서 동적으로 렌더링하기 때문에 URL의 변화에 따라 화면 상태를 변경
- HTML5 히스토리 API를 이용할 수 있다.

```
history.back();
history.pushState({}, "베이스볼게임", "game-started");
history.replaceState(state, title, url);
window.onpopstate = function(event) {
   console.log("history changed to: " +
     document.location.href);
}
```

- #(해쉬) 또는 $!(해쉬뱅)처리를 이용할 수 있다.

```
location.hash = '#game-started';
window.onhashchange = function () {
  hashChanged(window.location.hash);
}
```

CORS

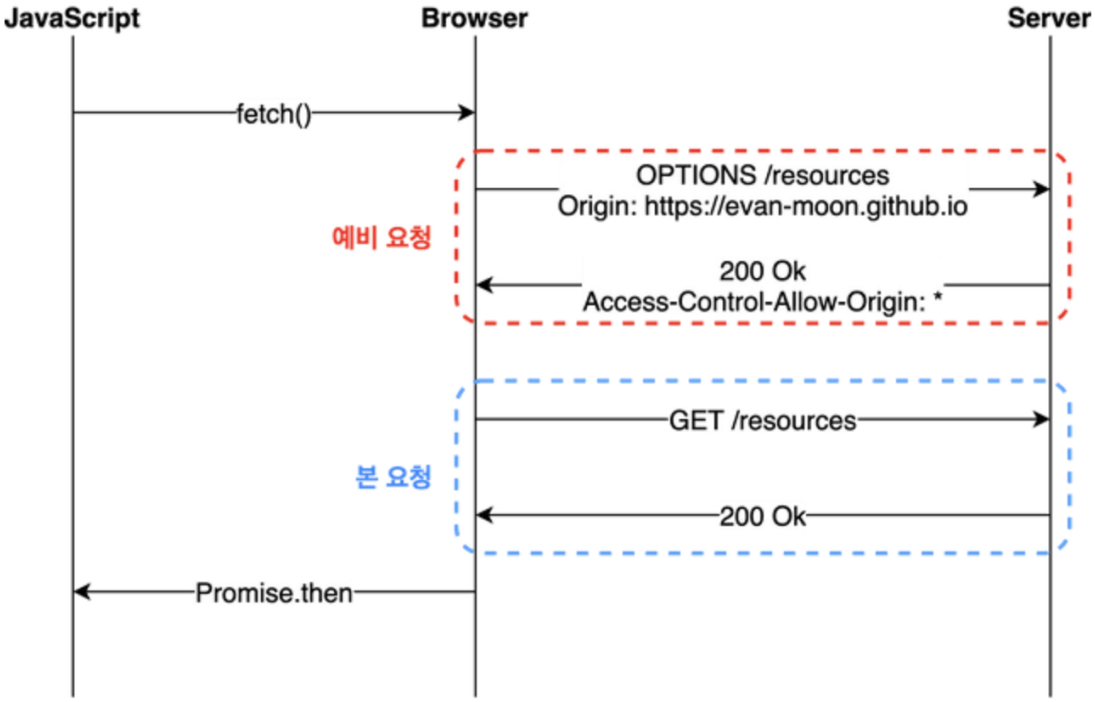
동일한 출처(same-origin)

예) http://store.company.com/dir/page.html 페이지의 스크립트가 아래 페이지에서 리소스 접근시

URL	결과	이유
http://store.company.com/dir2/other.html	성공	경로만 다름
http://store.company.com/dir/inner/another.html	성공	경로만 다름
https://store.company.com/secure.html	실패	프로토콜 다름
http://store.company.com:81/dir/etc.html	실패	포트 다름 (http:// 는 80이 기본값)
http://news.company.com/dir/other.html	실패	호스트 다름

출처: https://velog.io/@sangmin7648/SOP-CORS란

CORS 흐름



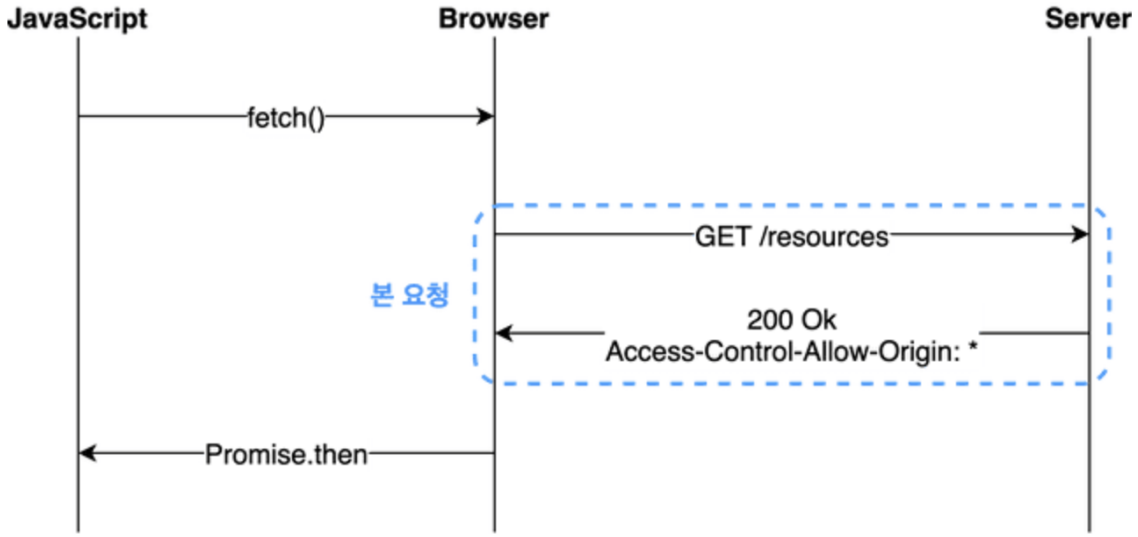
https://velog.io/@sangmin7648/SOP-CORS란

단순 요청(Simple requests)

일부요청은 CORS preflight 를 트리거하지 않습니다. Fetch 명세(CORS를 정의한)는 이 용어를 사용하지 않지만, 이 기사에서는 "simple requests"라고 하겠습니다. "simple requests"는 **다음 조건을 모두 충족하는 요청입니다:**

- 다음 중 하나의 메서드
  - `GET`
  - `HEAD`
  - `POST`
- 유저 에이전트가 자동으로 설정 한 헤더 (예를들어, `Connection`, User-Agent (en-US), Fetch 명세에서 “forbidden header name”으로 정의한 헤더)외에, 수동으로 설정할 수 있는 헤더는 오직 Fetch 명세에서 “CORS-safelisted request-header”로 정의한 헤더 뿐입니다.
  - `Accept`
  - `Accept-Language`
  - `Content-Language`
  - `Content-Type` (아래의 추가 요구 사항에 유의하세요.)
- `Content-Type` 헤더는 다음의 값들만 허용됩니다.
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`
- 요청에 사용된 `XMLHttpRequestUpload` 객체에는 이벤트 리스너가 등록되어 있지 않습니다. 이들은 `XMLHttpRequest.upload` 프로퍼티를 사용하여 접근합니다..
- 요청에 `ReadableStream` 객체가 사용되지 않습니다.

https://developer.mozilla.org/ko/docs/Web/HTTP/CORS



https://velog.io/@sangmin7648/SOP-CORS란

Let's write some code!