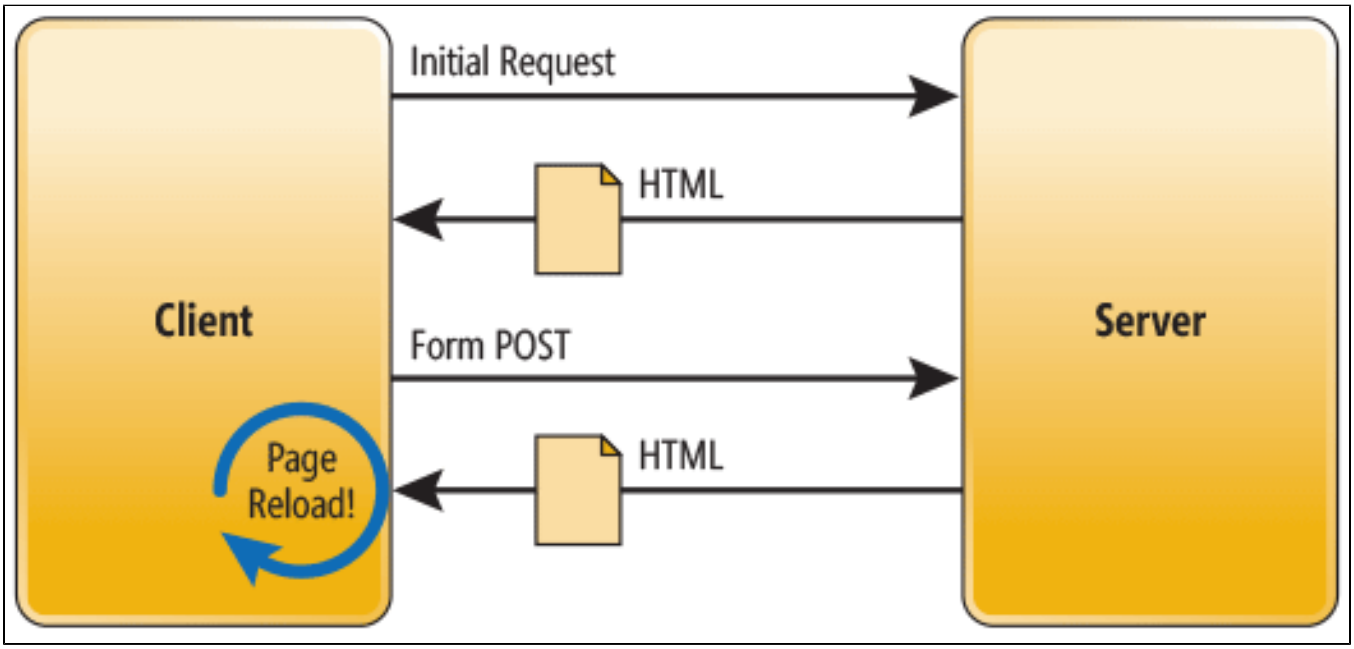
Day 06 (2019-07-24)
MPA (Multiple Page Application) vs SPA (Single Page Application)

Multiple Page Application

전통적인 웹 어플리케이션의 구현 방식

Client 요청에 따라 응답은 HTML 페이지

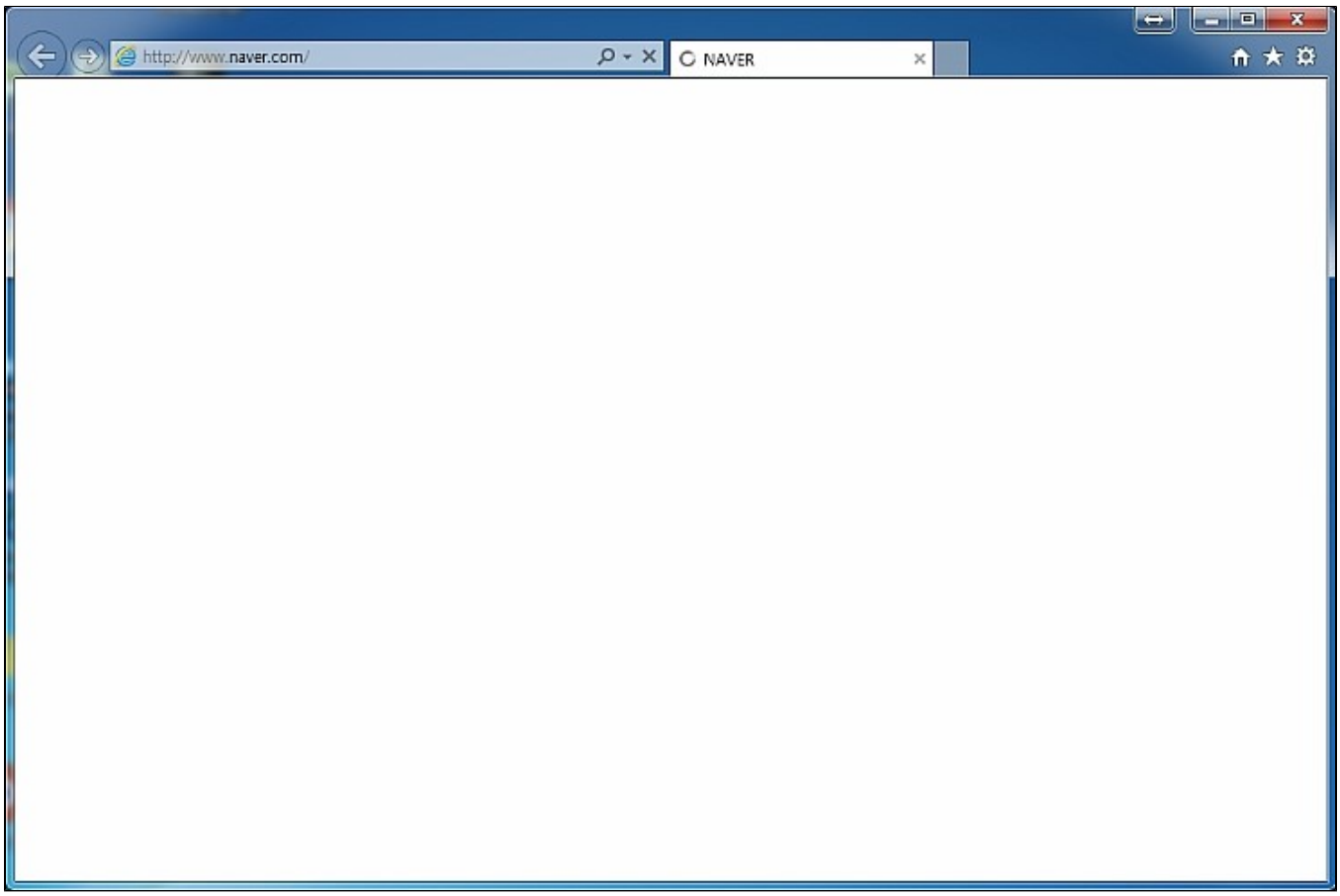


GET을 통해 조회를 하든, POST를 통해 등록을 하든 결과는 항상 HTML 페이지로 반환 됨

별도의 자바스크립트 기술이 필요 없음
→ 자바스크립트 표준의 제정이 늦은 것이 한 몫 함.

사용자의 액션이 발생할 때 마다 페이지가 깜빡하는 현상이 발생 함
→ 페이지네이션 (깜빡), 필터 입력 (깜빡) ...
→ ~~iFrame으로 대처가 가능하긴 함~~

무거운 HTML 문서가 통으로 네트워크를 이동하므로 비용의 낭비
→ 만약 해당 페이지에 무거운 쿼리가 존재한다면 사용자가 오랜시간 하얀 화면에서 대기해야 함.



발전하는 Javascript 비동기 기술

Ajax (Asynchronous Javascript And XML)

웹 서버와 비동기적으로 데이터를 교환할 수 있음

페이지 전체가 아닌 필요한 리소스만 송신, 수신 할 수 있음
→ 네트워크 대역폭 낭비를 줄임

동적인 화면 구현이 가능함
→ 페이지 일부분에 필요한 데이터 혹은 그 자체 HTML을 응답으로 받아 표시가 가능함

대표적인 기술, 구글의 검색어 추천

더 많은 곳에 접목

페이지의 깜빡임 없이 주요 Content 부분을 업데이트 할 수 있음

jQuery의 눈의 띄는 발전 (jQuery ajax, load ...)

Description: *Load data from the server and place the returned HTML into the matched elements.*

.load(url [, data] [, complete]) — **version added: 1.0**

url
Type: String
A string containing the URL to which the request is sent.

data
Type: PlainObject or String
A plain object or string that is sent to the server with the request.

complete
Type: Function(String responseText, String textStatus, jqXHR jqXHR)
A callback function that is executed when the request completes.

Note: Prior to jQuery 3.0, the event handling suite also had a method named `.load()`. Older versions of jQuery determined which method to fire based on the set of arguments passed to it.

Single Page Application의 등장

사용자에게 조금 더 좋은 UX를 줄 수 있음

비교적 적은 리소스를 통해 Content를 표시하므로 네트워크 비용이 줄어 듦

동적인 라우팅을 통해서 화면 이동 시 기존보다 사용자가 체감하는 속도가 빠름
→ 뭔가 보이기는 하네...

API의 재사용성 증가
→ 모바일 시대에서 App 개발을 위해 API 재사용 가능

실제 구현시 마주하는 문제점

사용자의 액션을 통해 동적으로 Content는 바뀌나 Browser의 History는 조작하지 않음

기존 Multiple Page Application에 익숙한 사용자들의 흔한 실수

- 현재 사용자가 보고 있는 페이지를 공유하기 힘듦
- 새로고침을 하면 페이지가 초기화 됨
- 뒤로가기 시 엉뚱한 페이지가?

→ 매우 나쁜 사용자 경험으로 다가감

앵커를 이용한 해결 법

사용자가 페이지 이동 액션을 할 때마다 현재 주소 뒤에 #을 통해 구분 처리

→ 대부분의 문제를 해결할 수 있음

pushState를 이용한 해결 법

URL에 #이 붙는 것이 싫은 유벌한 Client 들은 언제나 존재

Ajax를 이용한 뒤로가기 등의 문제를 해결하기 위해 HTML5에 새로 추가 된 History API

※ Front end Framework의 등장

하나의 어플리케이션을 구현 하려면 위와같은 기본적인 라우팅, 동적인 페이지 렌더링 (Content 영역), 리소스 관리, 데이터 관리 등은 지루하고 반복적인 작업

→ Copy & Paste

복붙을 통해 진항하다 한 쪽에서 버그가 발견되고 fix 되면?

→ Re-Copy & Paste

누구는 Hash 전략, 누구는 pushState

→ A 솔루션은 IE8에서도 잘 동작하던데 이 솔루션은 왜 안돼요? (# 붙는게 싫다고 하셨...)

누구는 jQuery load를 통해 동적 페이지 교체, 누구는 또 다른 라이브러리 사용

→ 정형화된 개발 방법이 없음

→ 이 소스, 저 소스 볼 때 마다 헷갈림

MVC, MVVW 등 디자인 패턴을 통해 정형화 된 방식을 제공 하고 지속적인 관리를 통해 올바른 트러블 슈팅 방법을 제안

SPA의 문제점

점점 커지는 Javascript로 인해 동적 라우팅을 통해 메뉴 이동은 빨라보이지만 최초 로딩은?
→ Require JS와 같은 AMD 모듈, Lazy Loading으로 어느정도 해결은 가능

SEO (Search Engine Optimization, 검색엔진 최적화)가 불가
→ 생각보다 큰 Google의 검색 봇을 통한 마케팅 효과
→ 하지만 검색봇이 조회한 페이지는...

모듈을 잘게 나누면서 소스 관리가 잘 안됨

사용자는 다양한 환경에서 어플리케이션을 이용 함 (IE, Edge, Chrome, Firefox 혹은 낮은 CPU 환경)

프리젠테이션 로직에 점점 비즈니스 로직이 녹아들어 감
→ 추가 된, 제거 된 항목 filter

Hybrid (MPA + SPA) ?

라우팅 시 매번 껍데기 HTML을 즉시 반환, Load 된 Javascript를 통해 필요 데이터는 Ajax를 통해 조회 및 화면 렌더링

→ 경계선이 애매 함
→ 템플릿 엔진을 통해 어느 부분까지 미리 렌더링하여 HTML로 반환하고 어느 부분 부터 Ajax 렌더링을 하죠?
→ 프로젝트의 일관성이 깨지기 쉬움

SSR (Server Side Rendering)과 Isomorphic javascript

기존 SPA의 가장 큰 문제점은?

→ 장점도 성능, 문제점도 성능 (동적 렌더링과 초기 구동 속도의 Trade-off)
→ 렌더링 시 CPU나 GPU 사용이 과하다면?

동적 렌더링을 통해 주는 좋은 사용자 UX는 부가적인 영역

초기 구동속도가 느려서 사용자가 **"서비스를 이용 못한다면?"**

렌더링 시 버벅이는 것을 떠나 브라우저가 반응이 없어 **"서비스를 이용 못한다면?"**

(실제로 좋은 UX를 주려면 큰 노력을 들여야 함, 데이터 로딩 시 화면이 밀리거나 데이터가 깜박하여 나타나는 것을 최소화 해야 함)

모든 문제는 Client Side에서 Rendering이 발생하기 때문

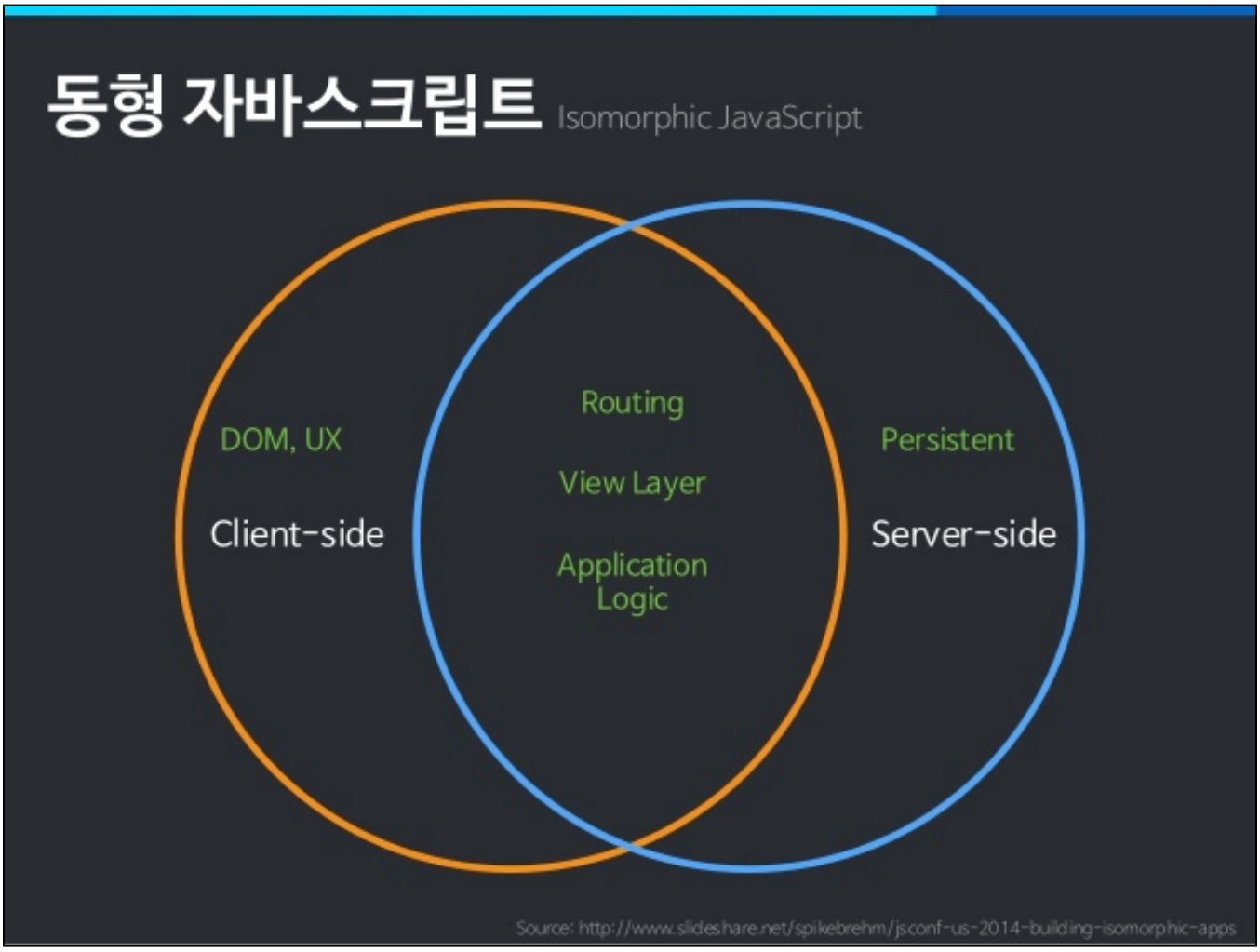
SSR은 클라이언트에서 **문제가** 발생하는 렌더링 영역을 서버사이드에서 처리 하여 반환 함

→ 꼭 문제가 발생하지는 않아도 됨

Isomorphic (동형)

클라이언트와 (렌더링)서버가 같은 코드를 공유 함. 즉, 클라이언트에서 렌더링하는 코드가 서버에 그대로 들어와도 정상 동작 해야 함을 의미(반대 방향도 성립 해야 함)

→ 큰 부담 없이 렌더링 코드가 실행 될 영역을 이동할 수 있음



React는 태생부터 SSR을 목표로 등장 함

Angular → Angular Universal

VueJs → Next.js, Nuxt.js

SSR의 단점

렌더링 서버를 하나 더 운영해야 하므로 비용 증가 및 배포 절차가 늘어남

→ 설정도 귀찮음, 서버 측에서 V8 엔진과 같이 Javascript를 실행할 수 있도록 추가 설정이 필요함 (mod_v8, Java Nashorn)

→ 렌더링 서버에 부하도 생각해야 함

→ 느린 네트워크 환경일 경우를 대비한 처리도 생각해야 함

은탄환은 없다 (No Silver Bullet)

SW 개발에 있어 복잡성을 한번에 해소할 방법은 없음

회사마다 조직마다, 솔루션 특성마다 적당한 방법을 선택하는 것이 제일 좋다.

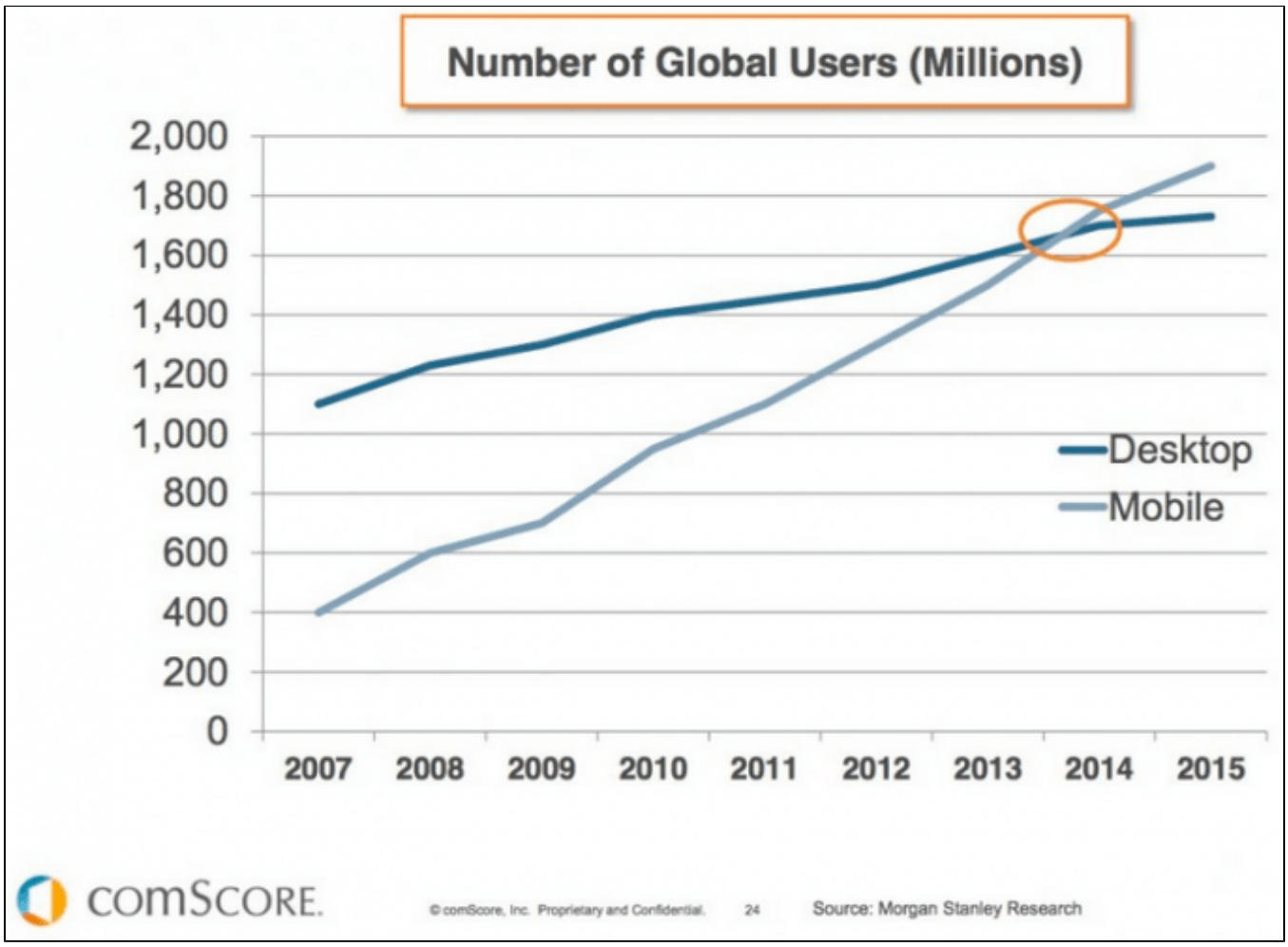
PWA (Progressive Web Application)

앱과 같은 웹 어플리케이션을 제작하기 위해 고안된 개념, 궁극적으로 앱 수준의 사용자 경험을 웹에서 제공하려는 것이 목표

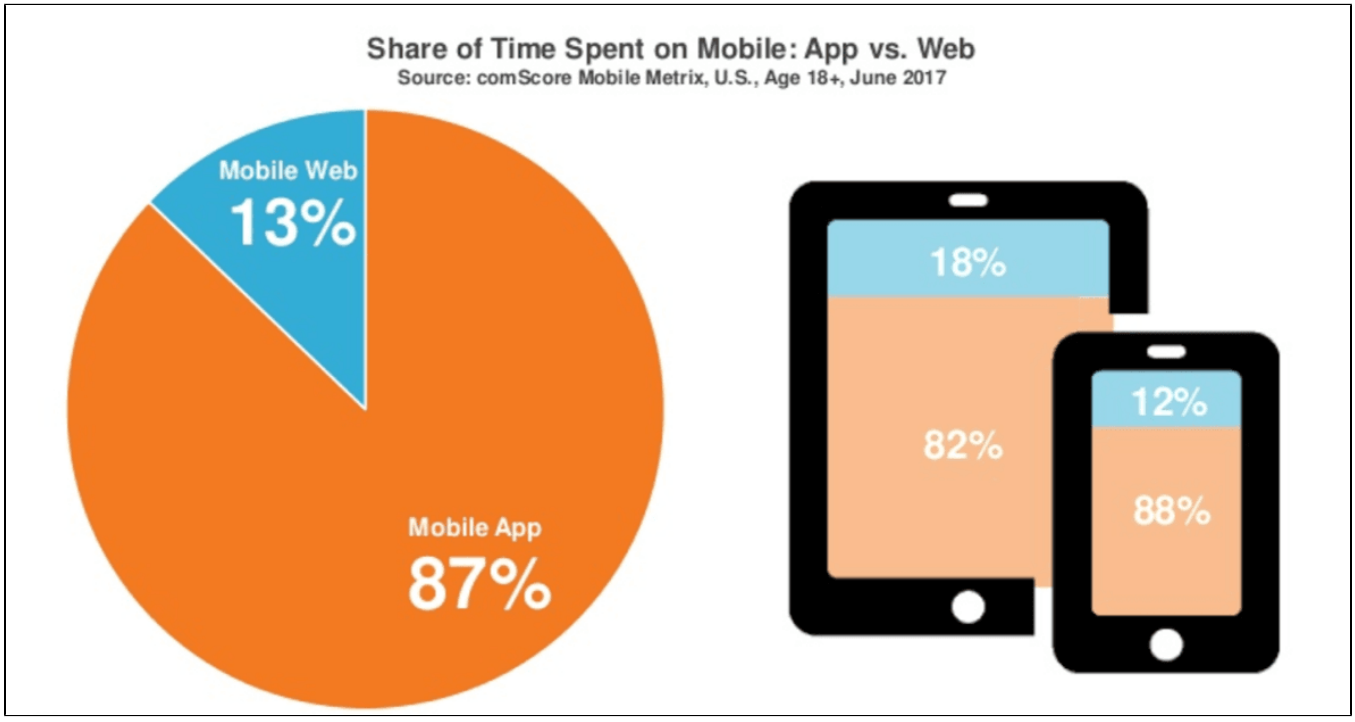
→ Push notification
→ Camera 및 Album 접근
→ Failback 처리를 위한 오프라인 동작
→ Installation

등장배경

모바일 사용자의 증가



모바일 사용자의 대부분은 웹보다 네이티브 앱에서 더 많은 시간을 소비 함

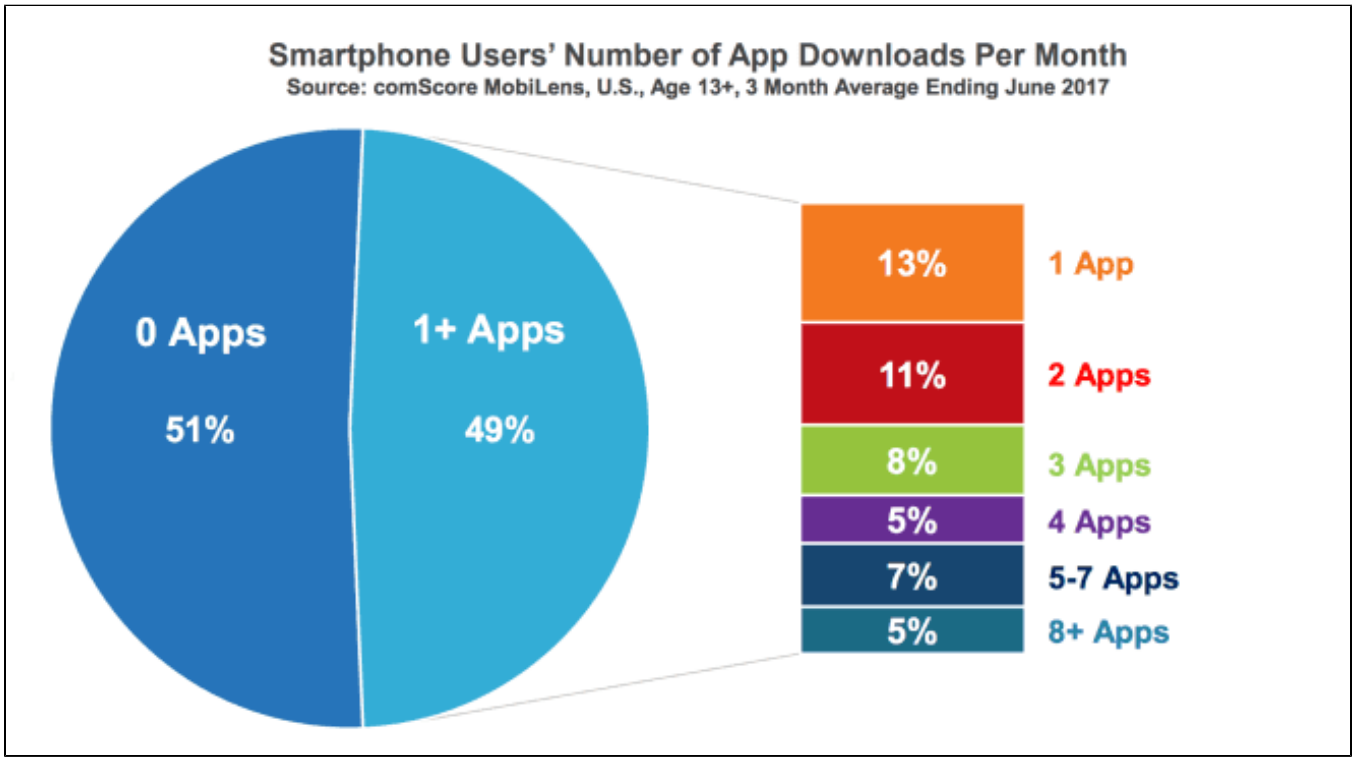


대부분의 사용자는 아래의 특징을 큰 장점으로 뽑음

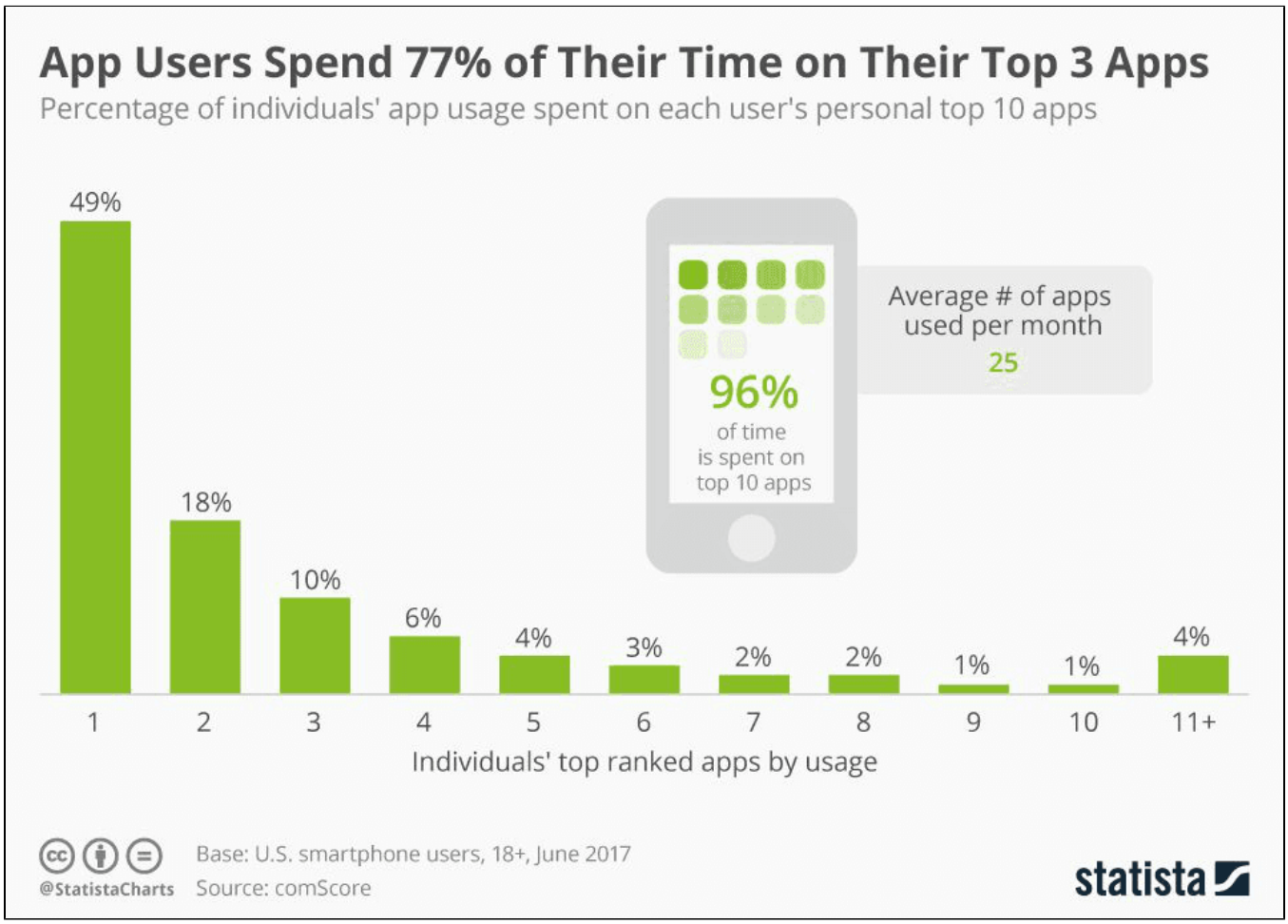
→ 홈에 아이콘이 있어서 접근성이 용이하였음
→ Push notification을 통해 유용한 알람을 받을 수 있었음

모바일 앱만 만들면 되나?

스마트폰 사용자들의 월별 앱 다운로드 횟수



앱 사용자들 상위 77%가 가장 많이 사용하는 상위 3개의 앱

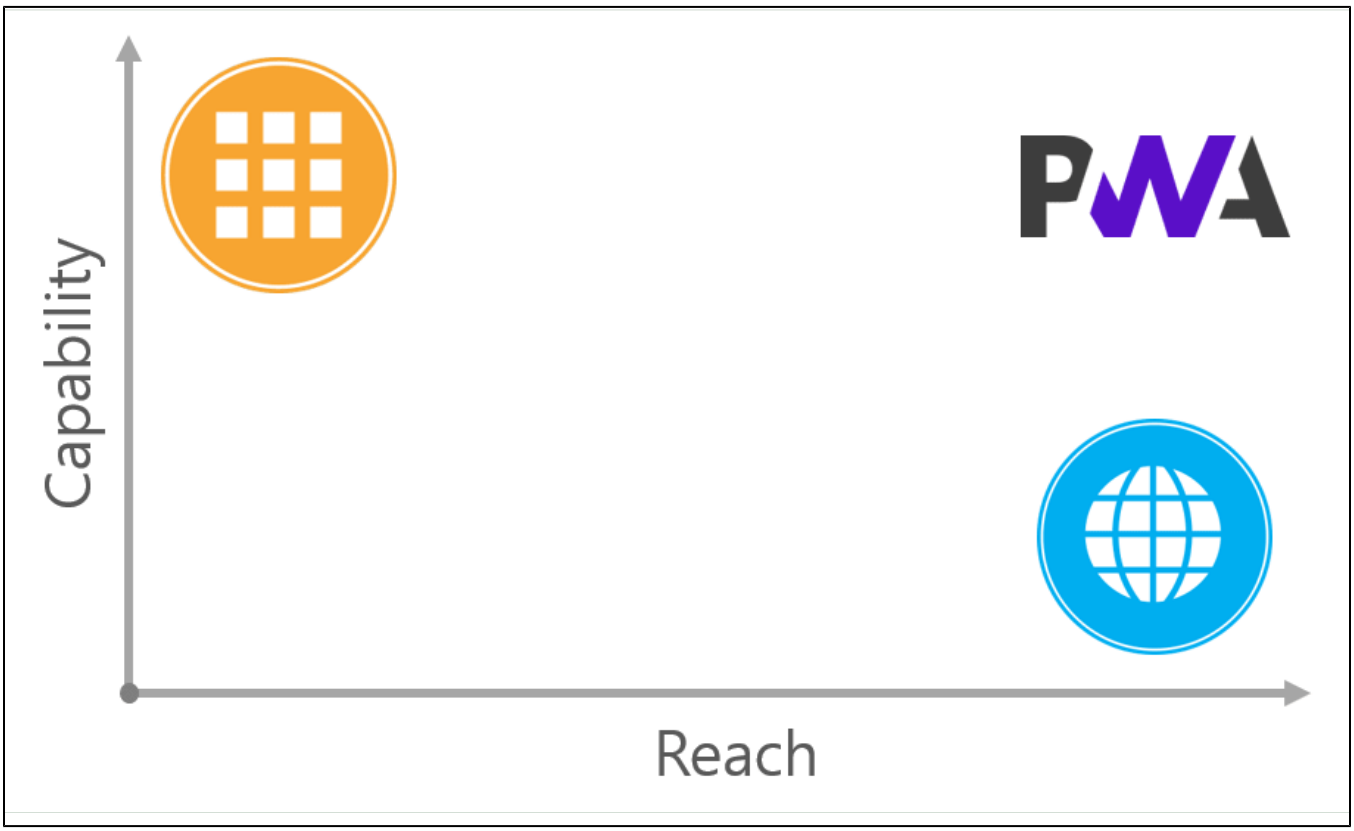


사용자는 새로운 앱을 다운로드 하는 것에 냉소적이며, 사용하는 앱만 더 자주 사용하는 경향이 있다.

→ 모바일 사용자가 한 달 앱 다운로드 수가 0개에 가까운데에 비해 웹 사이트는 100개 이상을 방문 함

→ URL이 가지는 힘, 네이티브 앱은 앱스토어에서 검색 후 다운로드 필요

Native App과 Web 그리고 PWA



Native App은 느리거나 불안전한 네트워크에서도 작동 할 수 있고(Failback) 백그라운드에서 Push notification을 보낼 수 있으며 카메라, 마이크 등 OS의 지원을 쉽게 받는다
→ Capability가 뛰어나다

Web은 URL을 통한 접근이 간단하고, 설치 과정이 필요 없다
→ Reach(Reachability)가 뛰어나다

채택 기술

- ResponsiveL CSS3, Flex 등
- Connectivity: Serverce Worker, HTML5 offline, puchDB, indexedDB
- App-like interactions: W3C Manifest, Web cache, App Shell Model
- Safety: Https
- Push Notification: Push API on Web

참고자료

MPA vs SPA

Isomorphic Javascript

Ajax를 사용할 때 '뒤로가기'의 구현

프로그레시브 웹 앱이란?

프로그레시브 웹 앱을 이해하기 위한 신기술들