Front-end Dev

2018 & beyond

2018.11.20 | 박재성

[Image source] https://www.flickr.com/photos/daecks/139589387/

오늘날의 JS 개발은?

프레임워크 + 개발도구

대다수 '기능'들은

npm 패키지를 사용, '조립'

Vanilla is not common

- Vanilla 개발은 더이상 일반적이지 않음
- Framework + toolchains(transpiler, bundling)

실작업 코드는 브라우저에서 직접적인 실행 어려움
최종적 결과물은 transpiler를 통해 JS로 변환

개발 방식의 진화

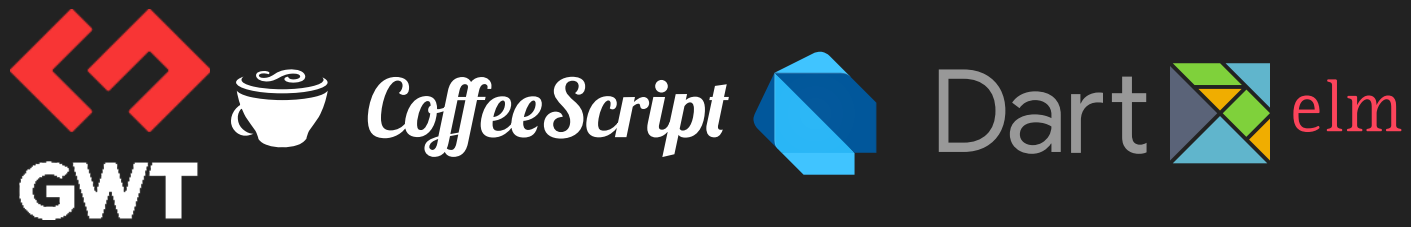
Library → Framework

- 직접적인 DOM 핸들링 필요성 감소
- DOM API native 지원향상 (ex. CSS selector 엔진 불필요)
- 컴포넌트 단위의 기능에 집중, DOM은 프레임워크에 맡김
  → 프레임워크에 따라 보다 효율적인 DOM(Virtual DOM 등) 처리제공

[참고] Removing jQuery from GitHub.com frontend

Transpiling Era

From non/pure-JavaScript code
Compiles/Generates to JavaScript



오래전 부터 JS로 컴파일 되는 도구/언어들의 시도는 지속

6to5 (2014) BABEL

새로운 syntax를 즉시 사용할 수 있게 됨
ES6+ 코드를 이전 브라우저 호환 코드로 컴파일

```
// ES6 arrow function code
var sum = (num1, num2) => num1 + num2;

// transpiled to ES5
var sum = function(num1, num2) {
   return num1 + num2;
}
```

브라우저가 새 문법을 지원하지 않더라도 사용가능.
브라우저 native 지원 시, 필요하지 않은 영역

[참고] 6to5(ES6 → ES5) (2014/09), become 'Babel' 2015/02 (Not Born to Die)
Sebastian McKenzie: JavaScript Transformation | JSConf US 2015

2018년 동향

2018년 동향

언어적 관점

[참고] 2018년과 이후 JavaScript의 동향 - JavaScript(ECMAScript)

버전 구분 중요성 감소

ES20xx/ESx

- Annual release로 인해 해마다 릴리스
  - 새로운 명세의 릴리스까지 너무 오래 소요
  - 단일 릴리스 모델의 한계
- Transpiler(Babel)는 최신 스펙 사용 이슈 해결
- 과거와 같이 버저닝에 큰 변화가 아닌 feature 단위
- Evergreen 브라우저는 스펙을 빠르게 구현

[참고] JavaScript — WTF is ES6, ES8, ES 2017, ECMAScript… ?

What's Coming?

ES2017/ES8

- Async/Await

```
async function getAPI() {
    let data = await some-async-call();  // 비동기 호출
    console.log(data);
}
```

- Shared Memory and Atomics

```
var sab = new SharedArrayBuffer(1024); // 버퍼 생성
worker.postMessage(sab); // worker를 통해 데이터 공유

var int32 = new Int32Array(sab);  // shared typedArray 객체 생성
Atomics.wait(int32, 0, 0);  // typedArray 값을 테스트, true면 sleep

console.log(int32[0]); // 쓰기 스레드가 새로운 값을 저장시 깨어나 값을 반환
```

ES2018/ES9

- RegExp improvement
- Promise.prototype.finally
- Async Iterators

```
async function main() {
   for await (const x of ['c', 'd']) {
       console.log(x);
   }
}
```

- Rest/Spread Properties

```
// Rest Properties
let {a, ...z} = { a: 1, b: 2, c: 3};  // z: {b, c}
// Spread Properties
let val = {a, ...z};  // val: {a, b, c}
```

ES2019/ES10

- Optional catch binding catch 파라미터 생략 가능

```
 // 사용되지 않는 catch paramter
try { ... } catch (unused) { ... }

// allowing the omission of a catch binding
try { ... } catch { ... }
```

- JSON Superset extend ECMA-262 syntax into a superset of JSON
  - ECMAScript는 JSON.parse에서 JSON이 subset 임을 명시하나
  - JSON에는 unescaped U+2028(LINE SEPARATOR)와 U+2029 (PARAGRAPH SEPARATOR) 포함될 수 있지만 ECMAScript String에는 포함되지 않는 문법적 오류 해결을 위한 스펙

[참고] TC39: Finished Proposals

WA

WEBASSEMBLY

프로그래밍 언어(지금은 C와 C++)를 컴파일해,
어느 브라우저에서나 빠르게 실행되는
바이너리(low-level language)로 바꾸는 기술

주요 브라우저 및 Node.js 모두 지원!

[참고] Support on Chrome, FF, Safari and Edge
Node.js v8 support!
Webpack v4: WASM example

정적 타입 시스템

JavaScript의 부족한 영역(loose typing)을 채워줌

- 버그감소 (15% 감소 연구결과)
- 효율적 코드작성 (Code Assist) - TS definition(.d.ts)
- as Languages: TypeScript, ReasonML, PureScript
- as Tools: Flow

TypeScript의 지속적인 영향력 확대
Angular, Vue.js, Polymer, ...

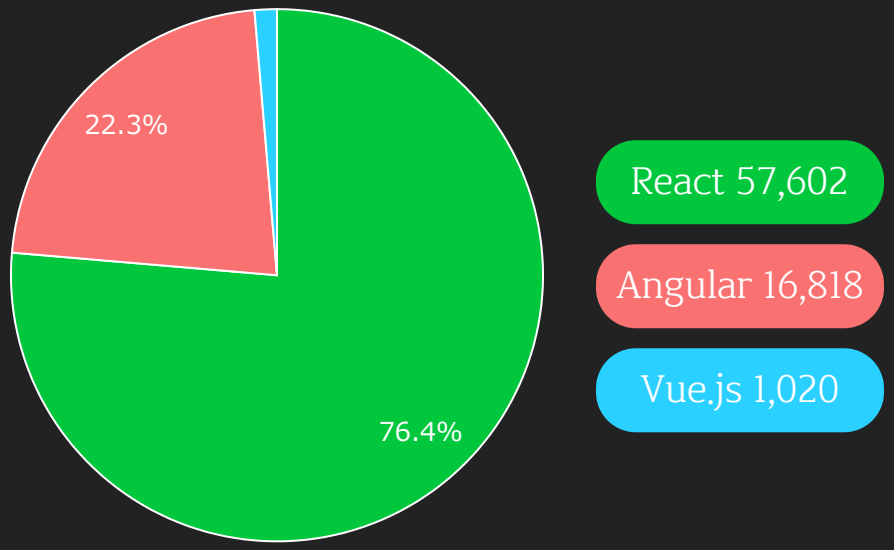
2018년 동향

라이브러리/프레임워크

[참고] 2018년과 이후 JavaScript의 동향 - 라이브러리와 프레임워크

Job Trends

총 FE Developer 포지션 - 24,220



[참고] indeed.com (미국 / 2018.11.14 기준)
React / Angular / Vue.js

React

- 2017 라이선스 이슈 (BSD + Patents → Re-licensed to MIT)
- Features
  - Improved Fragments: Child node 추가 개선
  - New ContextAPI: prop drilling 이슈 해결 (이전 API는 16.x까지만 동작)
  - Upcoming Rendering Improvements

    → Time Slicing(컴퓨터 성능 UX), Suspense(네트워크 성능 UX)
  - create-react-app 인큐베이션 졸업, v2 release (2018.10.1)
  - New Component Lifecycle (v16.3): componentWillMount, componentWillReceiveProps & componentWillUpdate v17에서 제거예정

[참고] React v16.0

React Hook (Proposal)

- 클래스를 작성하지 않고 state와 다른 React 기능을 손쉽게 사용
- React v16.7.0-alpha를 통해 사용 가능
- 'use' prefix를 갖으며, 커스텀 hook을 만드는 것도 가능

```
import { useState } from 'react';

function Example() {
  // 상태 변수와 setter 선언
  const [count, setCount] = useState(0);

  return (
      <button onClick={() => setCount(count + 1)}>
        You clicked {count} times
      </button>
  );
}
```

[참고] React Conf 2018: React Today and Tomorrow and 90% Cleaner React With Hooks
Introducing Hooks

Vue.js

- Impressive 2017 - 40K GitHub Stars
- Ecosystem is growing - Nuxt.js, Vuetify
- 네이티브 앱 개발 - NativeScript-Vue
- Features
  - drop IE support
  - vue-cli v3: Build Targets - Zero configuration (2018/8/13)

    → app, lib or webcomponent(wc/wc-async)

Vue.js 3.0 (likely 2019)

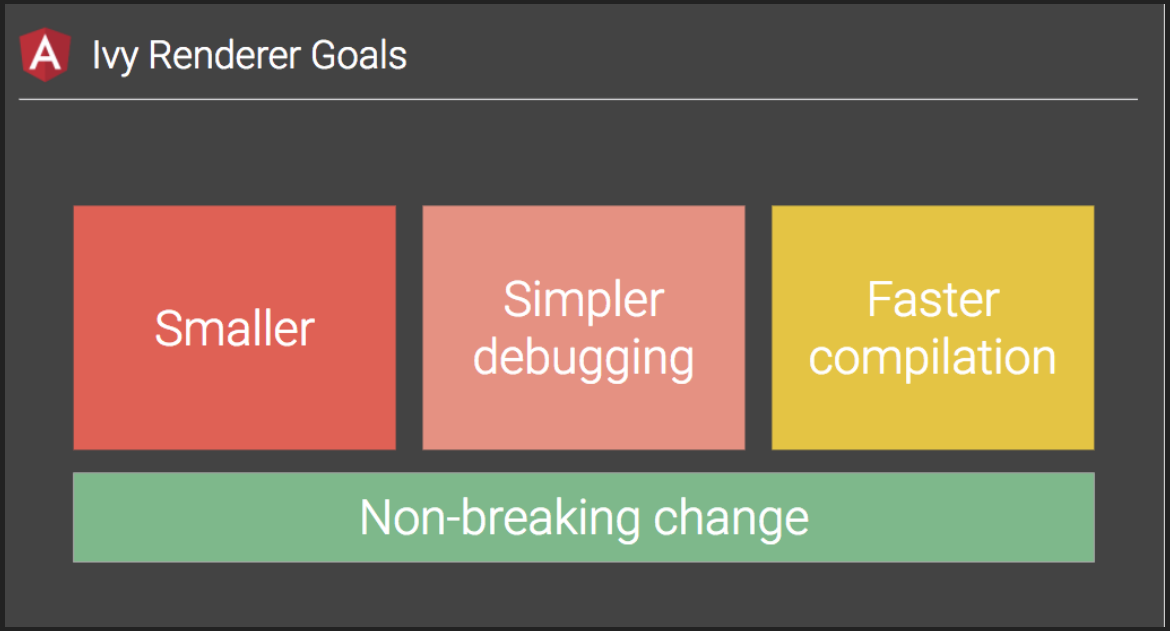
- Transpile 필요없는 native 클래스기반 컴포넌트
- 3.x 코드 베이스는 TypeScript로 작성 예정
- Proxy-based observer: v2.0의 Object.defineProperty 기반 제한 제거
- Lazy observation by default: startup data oberserve 오버헤드 제거
- Tree-shaking 친화적: 빌트인 컴포넌트&디렉티브 런타임 헬퍼 온디멘드 임포트
- new Virtual DOM implementation
- 2.0보다 100% 이상 빠른 성능향상
- 별도 빌드 버전을 통한 IE11 지원

[참고] Evan You - Plans for the Next Iteration of Vue.js (2018/9/30)

Angular

Ivy Renderer: Next generation rendering pipeline

새로운 렌더러, 작고 빠른 컴파일 (v7에 포함되지 않음)



[참고] Ivy Renderer: ng-conf: Day 1 Keynote - Brad Green, Miško Hevery, Kara Erickson

Angular Labs

- Angular Labs (안정성 유지, 신규 연구결과 균형 조절 및 전달)
  - Schematics: 코드를 변환, 생성하는 도구
  - Component Dev Kit: 다양한 확장 도구
  - ABC(Angular + Bazel + Closure): Angular 애플리케이션 빌드도구
- Angular Elements: Angular 컴포넌트 → Custom Element

[참고] Release Schedule

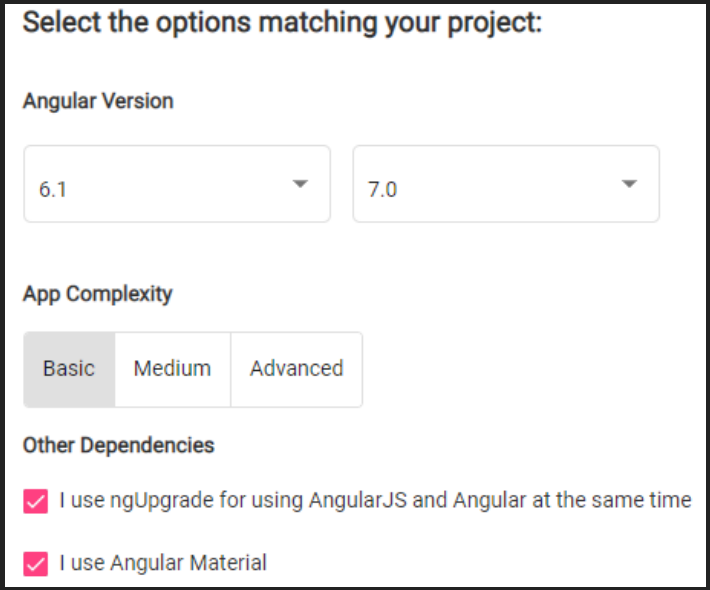
v7.0 (2018/10/18)

Material Design: CDK (Component Dev Kit)

- Virtual Scrolling
  <cdk-virtual-scroll-viewport> 태그를 통해 뷰포트에 보여지는 내용만 출력 사용자가 스크롤시, load/unload
- Drag and Drop
  다양한 드래그&드롭 기능 지원

Angular CLI

```
# update to v7
$ ng update @angular/cli @angular/core
```



- Built-In prompt: routing, CSS processor (ex.SASS)
- Bundle budget (2MB warn, 5MB error)

[참고] Version 7 of Angular—CLI Prompts, Virtual Scroll, Drag and Drop and more
Angular Update Guide

Web Components

- All major browsers support

  → Edge w/Polyfill, ShadowDOM & Custom Elements in development
- HTML imports → ESM 전환

Angular(Angular Element), Vue.js → 웹컴포넌트로 컴파일 가능

Polymer

- Factory 메서드 통한 요소정의

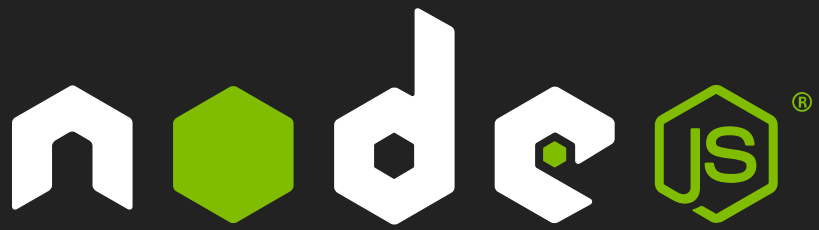
  → 표준 ES6 클래스/커스텀 엘리먼트 활용
- lit-html: HTML 템플릿팅 라이브러리
  - DOM 렌더링 집중 like JSX → 표준기술(Tagged Template)이므로 빌드 과정 필요없음
  - VDOM과 유사하게 업데이트된 영역만 렌더링 하지만, diffing 작업없어 효율적

```
import {html, render} from 'lit-html';

// A lit-html template uses the `html` template tag:
let sayHello = name => html`<h1>Hello ${name}</h1>`;

// It's rendered with the `render()` function:
render(sayHello('World'), document.body);
```

2018년 동향

브라우저 밖의 영역

[참고] 2018년과 이후 JavaScript의 동향 - 브라우저 밖의 JavaScript



- 2017년, 2억5천만 다운로드
- 다양한 영역의 기업들 사용증가
- 주로 Back-end 영역의 사용
- N-API (Node.js API) 런타임 엔진과 Node.js 간의 추상화 레이어

  → Replacing NAN(Native Abstractions for Node.js)
- 실험적 Node-ChakraCore
- WebAssembly and Node.JS

[참고] 10 Things I Regret About Node.js - Ryan Dahl - JSConf EU 2018

패키지 관리자: npm

Do not use Bower

- 전세계 가장 큰 레지스트리로 성장 (2018/10 기준, 80만개)
- npx, package runner
- repo 변경: https://github.com/npm/npm → https://github.com/npm/cli
- GitHub issue → npm.community 포럼 전환 (2018/7)
- npm의 사고들 (Linux 파일시스템 권한 변경, eslint-scope)

패키지 관리자: yarn

50만개 프로젝트에서 사용

- workspaces: 다중 패키지의 monorepo 구조에서의 효율적 패키지 관리
- resolution: 의존성 서브 패키지의 버전 관리

얼만큼 사용되나? (yarn을 통한 패키지 다운로드)

- 2018.10/13 ~ 11/10: 6백9십만 다운로드 발생

→ 전체 npm 다운로드의 0.03%

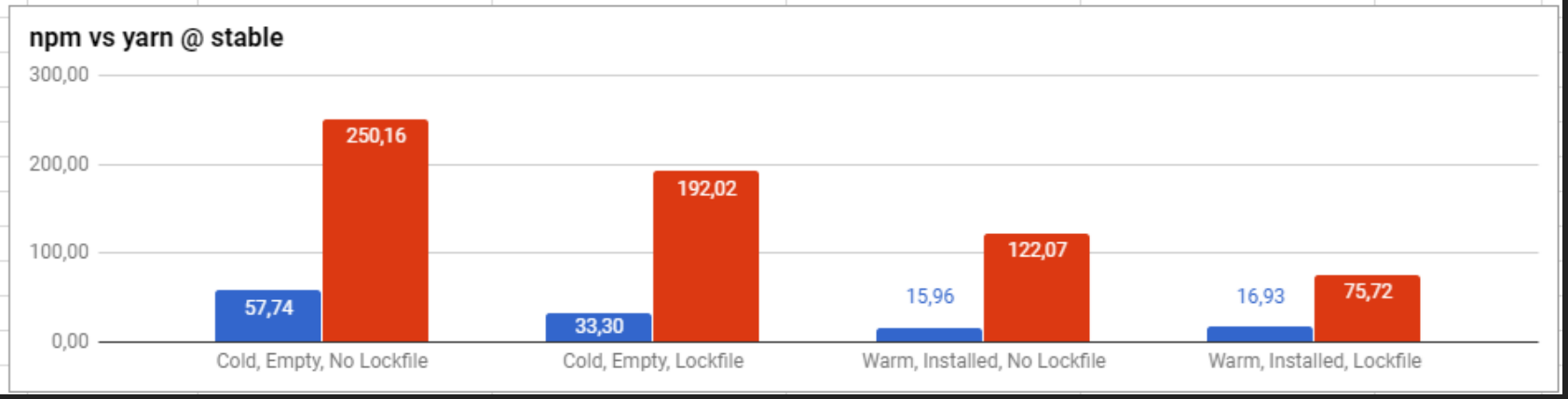
[참고] https://stats.yarnpkg.com/

npm vs yarn

초기 yarn의 장점(속도) 점점 퇴색

Comparing npm and yarn. Updated daily. See https://github.com/thomaschaaf/npm-vs-yarn

Installer	Latest Version	Cold, Empty, No Lockfile	Cold, Empty, Lockfile	Warm, Installed, No Lockfile	Warm, Installed, Lockfile
npm@stable	5.8.0	57,74	33,30	15,96	16,93
yarn@stable	1.6.0	250,16	192,02	122,07	75,72
Difference		433,25%	576,71%	764,70%	447,35%



[참고] https://github.com/thomaschaaf/npm-vs-yarn

번들러/빌드 도구

webpack v4 (2018/02)

- 향상된 빌드 시간: 60~98% 감소
- mode property (0CJS)
  → development | production | none
- split cli: webpack-cli

생태계는 메이저 업데이트에
준비되지 않았을 수 있다.

0CJS

Zero Configuration JS

Configuration Headache

PARCEL

Parcel과 같은 도구는 무설정 번들링 지향

모바일 애플리케이션

- webview를 래핑하는 형태의 hybrid-app은 죽어감
- JS to Native 형태의 네이티브 개발이 자리잡은 상태

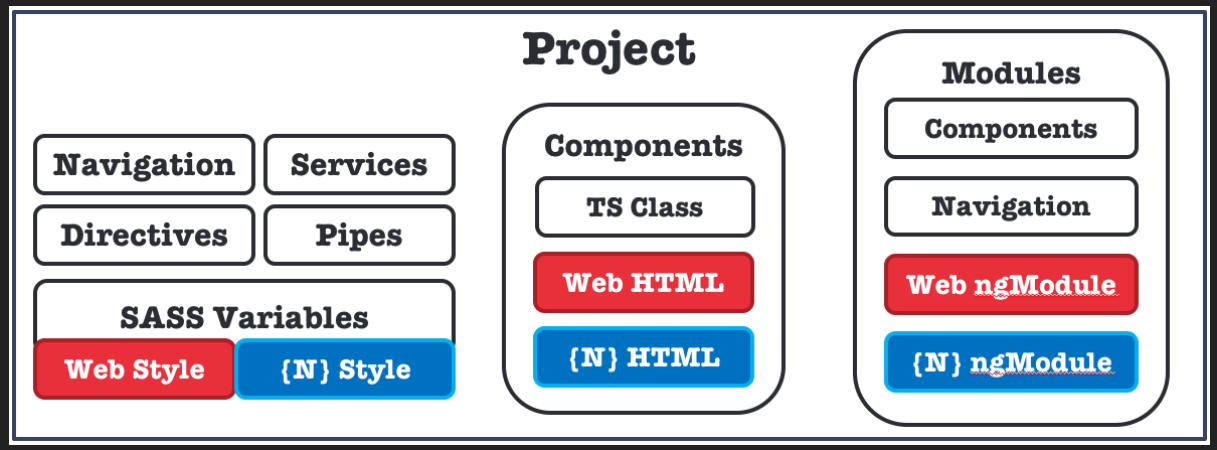
NativeScript

- Angular friendly
- NativeScript UI: UI 컴포넌트(차트, 캘린더, 등) 무료제공
- NativeScript-Vue를 통한 Vue.js 지원
- AR및 TV 플랫폼 지원 확산

[참고] Roadmap & Release Notes

NativeScript: Code Sharing

nativescript-schematics을 통해
웹과 모바일 앱을 단일 프로젝트에서 개발할 수 있다.




```
$ npm i -g @nativescript/schematics
$ ng new --collection=@nativescript/schematics my-mobile-app
```

[참고] Code Sharing
@angular/cli: 6.1+ 이상에서 사용가능

React Native

- React 인기에 더불어 인기상승
- create-react-native-app 발표 (July 24, 2017)
  → has been merged with Expo CLI (September 21, 2018)
- TV 플랫폼(Android/Apple) 지원
- 다양한 생태계/도구체인 - ex. Expo

데스크톱 애플리케이션

모두 Chromium 기반

ELECTRON

세밀한 제어 기능 제공

→ Windows95 같은 프로젝트 등장

NW.js

웹앱을 빠르게 데스크톱앱으로 전환시 장점

PWA

Safari와 Edge 지원 추가로 드디어 기본 사용환경 갖춤

- is ServiceWorker ready?

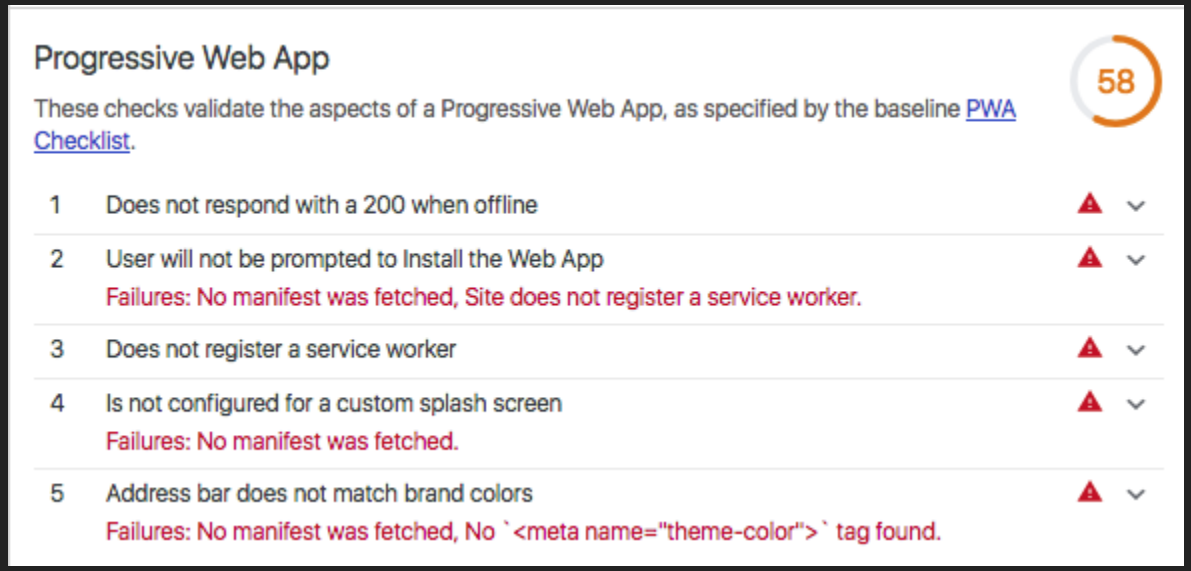
본격적인 분위기는 아직 무르익지 않았지만,
계속해서 사용 증가할 것으로 예측

다양한 도구의 등장

- Workbox, HNPWA, PWA Starter Kit, pwa-helpers

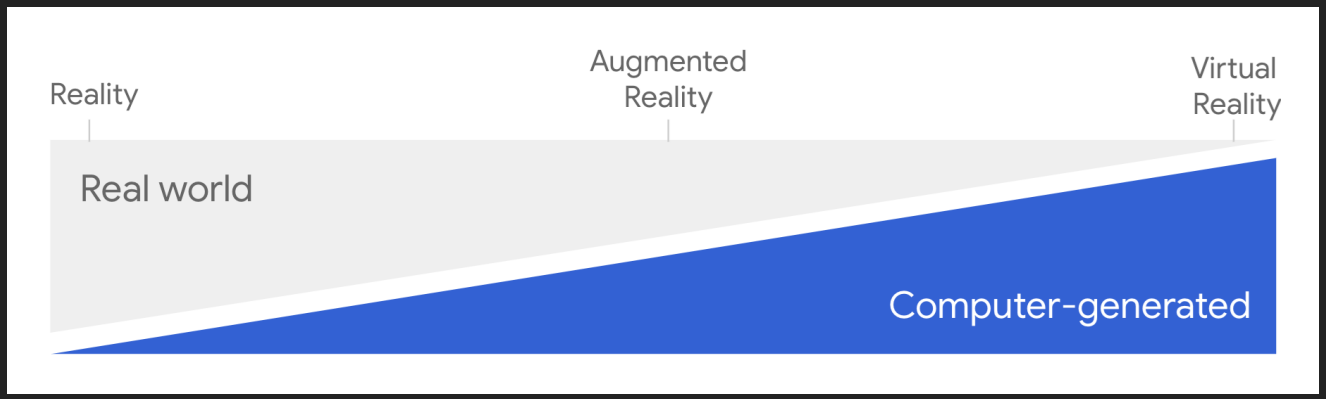
Chrome Audit

크롬 개발자 도구 성능 측정 도구에서 PWA 항목 평가 제공



WebXR Device API

- WebVR refactoring 필요성 제기
  - WebVR API → WebXR Device API
- Immersive Web: 웹에서 몰입 환경 구현 위한 신기술 모음



아직 성숙 단계는 아니기 때문에,
가능성을 확인하는 형태로의 접근

[참고] Welcome to the immersive web / WebXR Samples

Chrome에서 활성화

chrome://flags/#webxr

```
navigator.xr.requestDevice().then(device => {
   device.supportSession({ exclusive: true })
       .then(() => {
           // enabled VR
       }).catch(err => {
           console.error(`VR not found!: ${err}`);
       });
});
```

[참고] The future of the web is immersive (Google I/O '18)

JavaScript는

어디로 가고 있나?

흐릿해지는 경계

(Backend/Frontend)

FE와 BE 구분없이, 하나의 코드셋으로 모든 곳에서 실행

Universal JavaScript

[참고] Node.js는 Back-end에서 주로 사용

Node.js:2017 User Survey Executive Summary

Intent to merge

Node.js + JS Foundation

JS의 역할과 다양한 환경으로의 확장에 따른 환경변화

지속적이고 견실한 성장을 위한
생태계 협업의 필요성이 어느 때보다도 중요

node JS Foundation

JS Foundation

[참고] Node.js Foundation and JS Foundation Announce Intent to Create Joint Organization (2018.10.4)

JavaScript로부터
생존하기

JavaScript Fatigue

많은 가이드들은 몇가지만 설치/설정하면
쉽다(out of the box)...고 말한다.

수많은 플러그인과 설정 등을 접하다보면

쉬운거 맞어?

우리는

- 무엇을 알아야 할까?
- 얼만큼 알아야 할까?

How to stay: Up-to-date?

1) Trends

- GitHub Trending: 새로운/떠오르는 프로젝트 확인
- FE Tech Mailings: 최신 소식 업데이트

2) Browsers Update

STABLE	NEXT UP	
Chrome 69	Chrome 70	Chrome 71
Beta was Aug 2 - Aug 9	Beta between Sep 13 - Sep 20	Beta coming Oct 25 - Nov 1
Stable 34 days ago (Sep 4)	Stable in 8 days (Oct 16)	Stable in 57 days (Dec 4)

Chrome / WebKit / MS Edge / Firefox

Keep eyes on

다른 사람들은 어떻게 생각하나?

- State Of JS
- Stack Overflow Annual Developer Survey
- Medium's javascript posts

[참고] Frontend Roadmap
Frontend Path & Courses for Learning Front End

컨퍼런스

많은 컨퍼런스들이 행사 종료 후, 세션 비디오를 공개

스케줄을 살펴보는 것만으로도
현재의 흐름을 파악할수 있다.

- JSConf
- Fluent
- dotJS
- JS Kongress
- Netflix JavaScript Talks

[참고] https://github.com/frontendfront/front-end-conferences

무엇이 옳은 선택인가?

매몰비용 오류의 함정

특정 도구에 투자한 노력이 많아지면, 나의 선택(도구)이 합리적이었다는 것을 자신이나 다른 사람들에게 설득하기 위해 노력

AngularJS 등장 → 모든 것을 AngularJS로!

현재는?

React, React and React

Magpie Developer syndrome

대체로 개발자들은
기존에 문제없이 사용하던 기술들을 뒤로하고,

'새롭고 반짝'

이는 것들에 너무 쉽게 관심을 빼앗긴다.

[참고] The Magpie Developer
Magpie logo from, Magpie Brewing Co.

무엇을 사용하던간에

결국 JavaScript 아닌가?

Transpiling: 결국 실행되는 JS 코드 생성 필요

> *TS adds sweetness, but at a price.*
> *– Douglas Crockford*

[참고] JavaScript vs. TypeScript vs. ReasonML
Here is why you might NOT want to use TypeScript

지속적 복잡도 증가

이미 필수적인 도구들도 넘쳐나는 상황에서
무언가를 하나 추가할 때마다 비례하는 도구들의 증가

xxx-loader-xxx, xxx-plugin-xxx, ...

2018/11 기준, 플러그인 수:

- webpack
  - plugin: 3,534개
  - loader 3,650개
- babel 3,315개

최신 업데이트
항상 정답인가?

도구들의 업데이트에 따라 코드 수정이 필요한 모순적 상황
이러한 작업은 때에 따라 많은 리소스 수반

메이저 업데이트 후, 수많은 플러그인들은
대비되지 않거나 대체제가 없기도

[참고] Babel 7: Major Breaking Changes
Webpack 4: Breaking changes for plugins/loaders

Webpack's case

webpack-dev-server: 'maintenace mode'로 전환발표
이제, 다시 개발 진행한다고 함

webpack-dev-server → webpack-serve
→ 다시 webpack-dev-server

webpack-cli도 동일 → webpack-command (now deprecated)

Why deprecated webpack-command?
webpack-cli is very stable and have more features

[참고] Remove obsolete maintenance note
Clarify relationship with webpack-serve

각자의 사정은 모두 다르다.

'무엇'이 정답이라 할수 없다.

과거 jQuery, AngularJS가 답처럼 여겨졌지만,
지금 React가 대세적인 흐름의 상태에서
Angular가 정답이라고 할수 있을까?

그때는 맞고, 지금은 틀리다.

모든 것을 다 알아야 할까?

누군가는 '특정'영역에 보다 관심을 쏟을 수 있고,
더 '많이' 알고 있을 수도 있다.

그것이 모범적인 '답변'일까?

모든 것을 알수 있을까?

Linus Torvalds가 모든 Linux의 모든 생태계의 변화들을,
Brendan Eich가 모든 JS 생태계의 변화들을

알고 있을거라고 생각하시나요?

우리의 자세는?

Everything is amazing
and nobody is happy

- "Expert"로써 모든 플랫폼의 모든 것을 알아야 한다는 생각
- 모든 것을 배우는 것이 해결책인가?
- 단순하게 현재 작업에 필요한 것, 또는 끌리는 것에 집중

[참고]
Why I'm Thankful for JS Fatigue
A Study Plan To Cure JavaScript Fatigue

결국은 적당한 밸런스

너무 많은 것을 알려고 하지 않아도 된다.
그러나, 적당한 '호기심'과 '지속적'인 꾸준함은 필요

JavaScript
is a Journey!

고맙습니다.
Thank You.
Gracias.
[Image source] https://www.flickr.com/photos/deucer/7869331470/