The state of JavaScript & FE

2021 Edition

2021년 FE 동향, 한방에 끝내기

DEVIEW 2021

NAVER 박재성 / Platform FE

talk is about

빠르게 변하는 JS/FE의 2021년 생태계의
변화들을 이야기합니다.

💡 발표 자료는 2021/10월 기준으로 작성되었습니다.

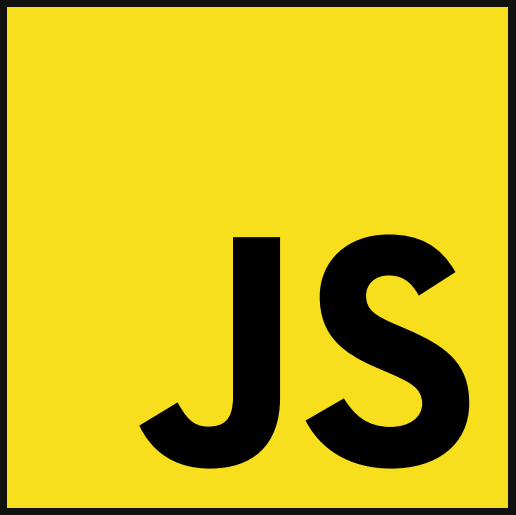
생태계의 방대함과 시간적 제약으로 인해, 세부 내용까지 다루지는 않습니다.
보다 자세한 내용은 각 장표에 포함된 참고 링크를 통해 확인 바랍니다.

네이버 기술 블로그: D2 HelloWorld

JavaScript 동향 시리즈

연도	문서
2016	2016년과 이후 JavaScript의 동향
2017	JavaScript(ECMAScript) 라이브러리와 프레임워크 브라우저 밖의 JavaScript
2018	JavaScript(ECMAScript) 라이브러리와 프레임워크 브라우저 밖의 JavaScript
2019	JavaScript(ECMAScript) WebAssembly 라이브러리와 프레임워크 (#1, #2) 브라우저 밖의 JavaScript (#1, #2)
2020	JavaScript(ECMAScript) WebAssembly 라이브러리와 프레임워크 (#1, #2)

JavaScript
(ECMAScript)



ECMAScript 2021/ES12

명세	설명

String.prototype.replaceAll	정규식 `g` 플래그와 특수문자 escape 처리 불편함을 제거
	`'xxx'.replace(/(?:)/g, '-'); // '-x-x-x-'`
	`'xxx'.replaceAll('', '-'); // '-x-x-x-'`

Promise.any	여러 개의 Promise 모음 중, 첫 번째로 이행(fulfilled)되고, 처리(resolved)된 Promise를 반환
WeakRefs	약한 참조(weak reference)를 갖는 객체를 저장해, 참조되지 않지만 GC에 포함되지 않는 상황을 예방
Logical Assignment Operators	`&&= ||= ??=` 논리 연산자와 할당 표현식의 혼합 사용
Numeric Separators	긴 숫자 값에 대한 가독성을 위해 _(underscore)를 구분자로 사용 ex) `1000000000 --> 1_000_000_000`

ECMAScript 2022/ES13

public, private, static 사용에 대한 명세

- Private methods and getter/setters for JavaScript classes
- Class field declarations for JavaScript
- Static class features

```
// Field Declaration
class Counter extends HTMLElement {
  x = 0; // public field
  static red = "#ff0000";  // static public field

  constructor() {
    super();
    // this.x = 0; <-- 이렇게 생성자에서 하지 않고, 상위 선언 가능
  }

// Private Fields
class Counter extends HTMLElement {
  #x = 0; // private field
  static #red = "#ff0000"; // static private field

  clicked() {
    this.#x++;

    static #some() { // static private method
```

명세	설명
RegExp Match Indices	새로운 `d` 플래그를 통해 매칭 문자열의 시작과 종료 index를 반환
Top-level await	async 함수 없이, 상위 레벨에서 `await` 사용
Ergonomic brand checks for Private Fields	private 필드값의 존재 유무 판단하고 없다면, fallback 제공
Accessible Object.prototype.hasOwnProperty	`Object.prototype.hasOwnProperty` ➡ Object.hasOwn()로 접근성 개선
Class Static Block	클래스 정의시에 추가적인 static 초기화 방법을 제공

What's coming?

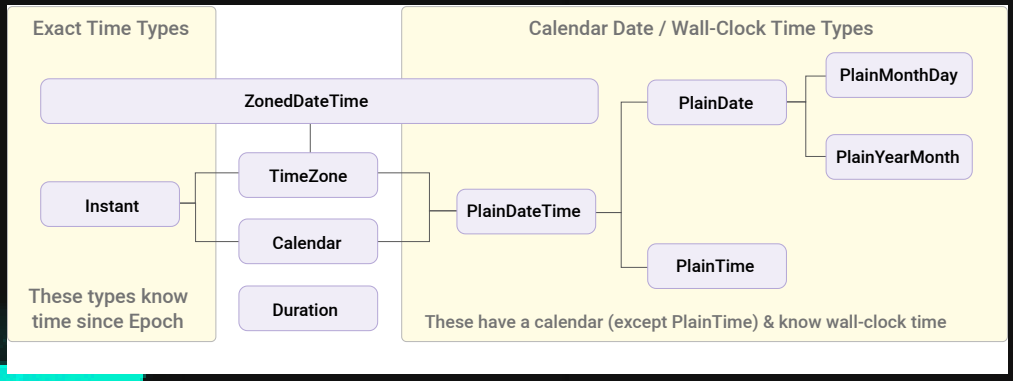
- Import Assertions (Stage 3)
  서버가 잘못된 MIME 타입을 잘못 처리해
  발생될 수 있는 보안이슈를 제거하기 위함
- JSON Modules (Stage 3)
  JSON을 보편적 방식의 모듈로 import 한다.
  Import Assertions 명세의 일부였으나, 20/7월 명세를 분리

```
1 import json from "./foo.json" assert { type: "json" };
2 import("foo.json", { assert: { type: "json" } });
```

- Error Cause (Stage 3)
  오류 객체 초기화시 추가 옵션 객체 필드 `cause` 추가

```
async function doJob() {
  const rawResource = await fetch('//domain/resource-a')
    .catch(err => {
      throw new Error('Download raw resource failed', { cause: e
    });
}

try {
  await doJob();
} catch (e) {
  console.log(e);
  console.log('Caused by', e.cause);
}
// Error: Upload job result failed
// Caused by TypeError: Failed to fetch
```

- Temporal (Stage 3)
  모던한 날짜/시간 API를 제공해 기존 Date 객체를 개선

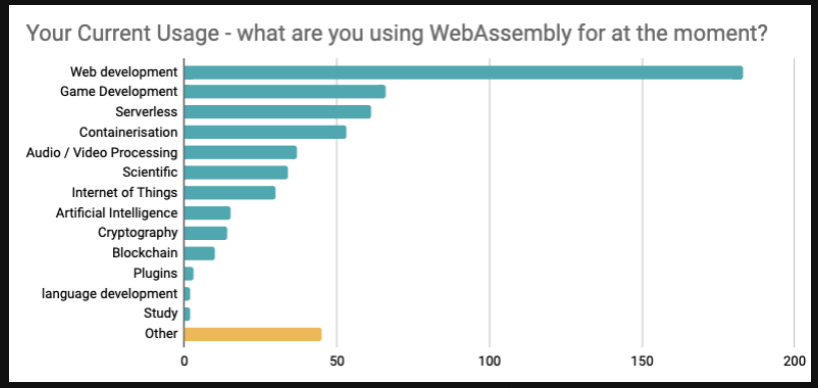


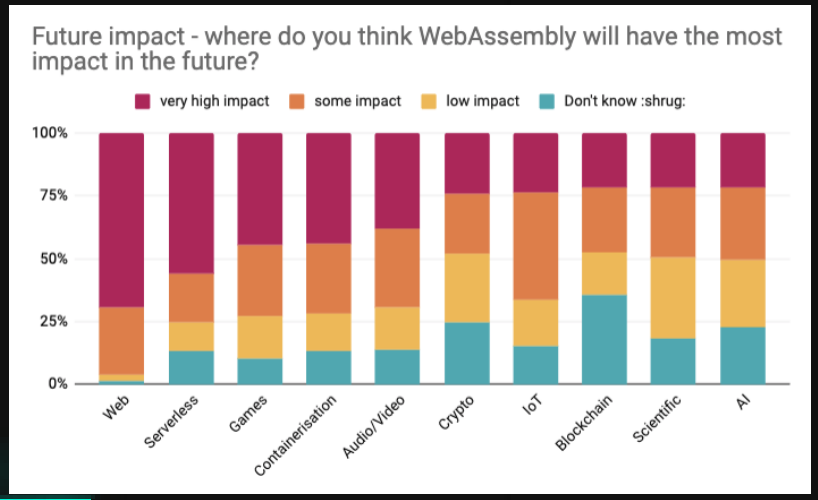
💁 ECMAScript proposals: Stage 3

WEBASSEMBLY

웹에서의 wasm

많은 이들은 현재 웹 개발을 위해 사용하고,
미래에도 웹에 커다란 영향을 줄것이라 생각하고 있다.

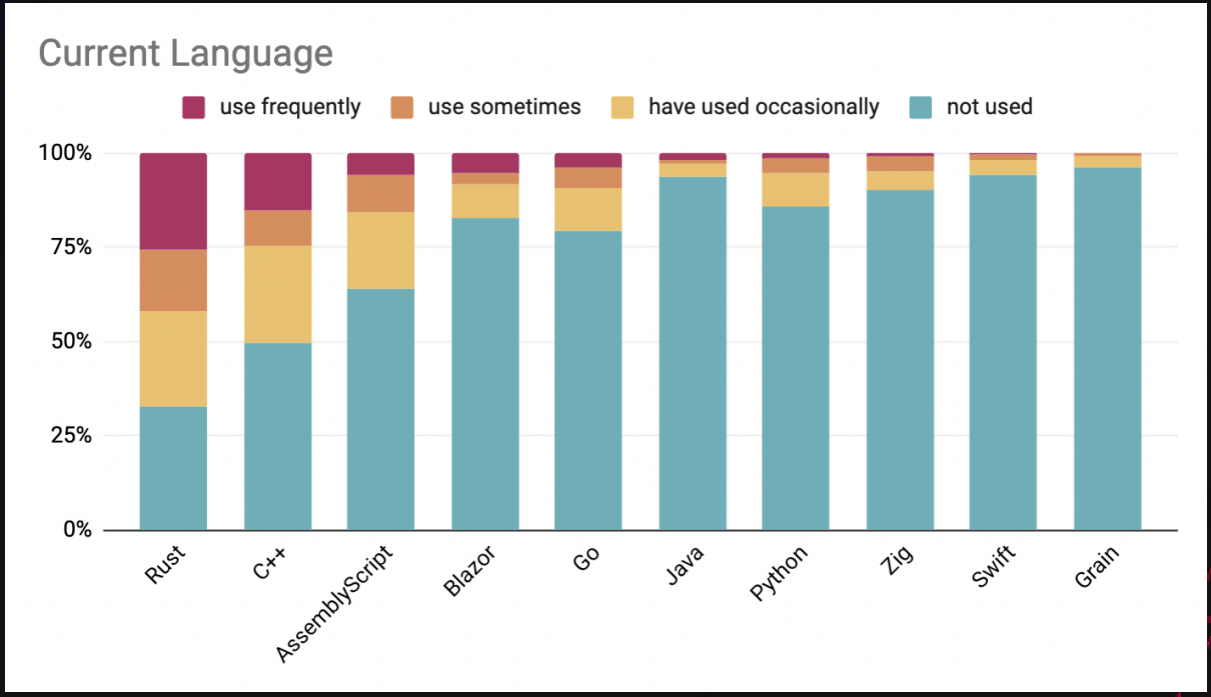




🤷 The State of WebAssembly 2021

wasm 개발에 사용하는 언어?

Rust(26%) > C++ > AssemblyScript > Blazor 순



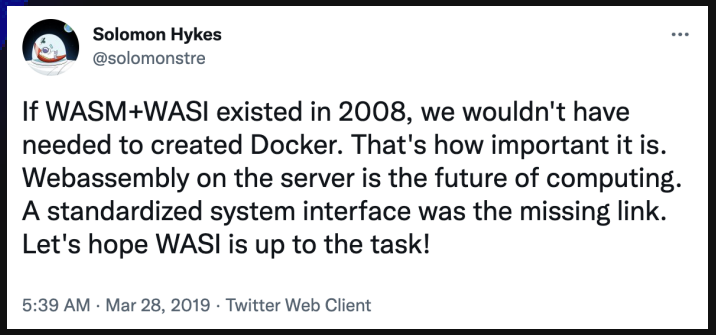
🤦 Languages for wasm

Universal Runtime(VM)?

wasm의 사용은 웹으로만 국한될까?

WASM + WASI가 2008년에 존재했었더라면, Docker를 만들 필요성이 없었을 것이다…서버에서 wasm은 컴퓨팅의 미래다.
Solomon Hykes (Docker 공동 창업자)

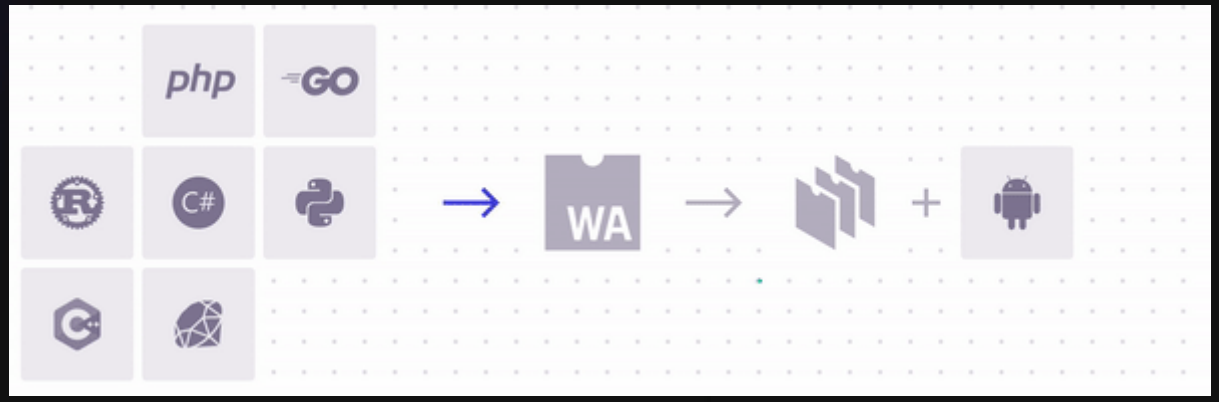
https://twitter.com/solomonstre/status/1111004913222324225



범용 VM 아이디어 실현은 https://webassembly.org 이다.
Brendan Eich

https://twitter.com/BrendanEich/status/1001307081725562882

친숙한 도구와 언어로 개발하고 wasm으로 컴파일
모든 OS에서 실행되거나 다른 언어에 임베드해서 사용 가능

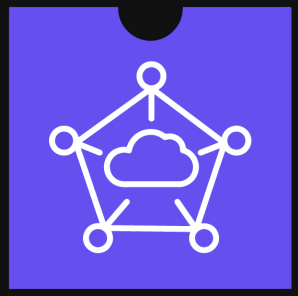


Wasmer

- 12여개 언어 임베딩 지원

`wasmtime`

- 6개 언어 임베딩 지원



WasmEdgeRuntime

클라우드 네이티브, edge, 분산화 환경
애플리케이션을 위한 wasm 런타임이며,
가장 빠른 성능을 보유하고 있다.

Second State에서 SSVM(Second State Wasm VM) 이름으로
19/6월 개발을 시작했고,
0.8 릴리스에 현재의 프로젝트명으로 변경했다.

CNCF 샌드박스 프로젝트로 21/6월 승인되었다.

C/C++, Rust, Swift, AssemblyScript, Kotlin 코드로 작성된
wasm 바이트코드 프로그램 실행과 임베드된 QuickJS 엔진을 통해
wasm 내에서 JavaScript 코드도 실행할 수 있다.


https://github.com/WasmEdge/WasmEdge

wasmtime + Lucet = wasmtime

Lucet (Fastly)과 Wasmtime(Mozilla)은 각각 개발사들이
Bytecode Alliance에 참여한 후, 프로젝트가

모두  저장소로 이동했었다.

2020년 중반, Lucet 팀은 Wasmtime 프로젝트에 노력을 집중하기로 결정하고,
Lucet는 메인터넌스 모드로 전환되었다.

Bytecode Alliance 산하의 또 다른 프로젝트인
WAMR(WebAssembly Micro Runtime)도 개발되고 있으며,
wasm 인터프리터, AoT, JIT 컴필레이션을 지원한다.

QuickJS

QuickJS는 C로 작성된 임베딩 JavaScript 엔진(JIT 지원없는 인터프리터) 이다.

WasmEdge은 별도의 포팅 버전 wasmedge-quickjs이 제공되며,
또다른 런타임인 Wasmer에서도 qjs.wasm 모듈을 통해 사용할 수 있다.

```
1 $ wasmer qjs.wasm
2 QuickJS - Type "\h" for help
3 qjs > const i = 1 + 2;
4 qjs > console.log("hello " + i);
5 hello 3
```

wasm 모듈로 제공되므로,
사실상 모든 wasm 런타임에서 사용할 수 있다.

WebAssembly Roadmap

	Chrome 91	Firefox 90	Safari 15.0	Wasmtime 0.22	Wasmer 2.0	Node.js 16.4
Standardized features						
JS BigInt to Wasm i64 integration	✓	✓	✓	n/a	n/a	✓
Bulk memory operations	✓	✓	✓	✓	✓	✓
Multi-value	✓	✓	✓	✓	✓	✓
Import & export of mutable globals	✓	✓	✓	✓	✓	✓
Reference types	⏳	✓	✓	✓	✓	⏳
Non-trapping float-to-int conversions	✓	✓	✓	✓	✓	✓
Sign-extension operations	✓	✓	✓	✓	✓	✓
Fixed-width SIMD	✓	✓	✗	⏳	✓	✓
In-progress proposals						
Exception handling	⏳	⏳	✗	✗	✗	⏳
Extended constant expressions	✗	⏳	✗	✗	✗	✗
Module Linking	✗	✗	✗	⏳	✗	✗
Tail calls	⏳	✗	✗	✗	✗	⏳
Threads and atomics	✓	✓	⏳	✗	⏳	✓

Roadmap
WebAssembly proposals

WebAssembly W3C 프로세스

Phase	단계	설명 & 필수요건
0	Pre-Proposal [Individual Contributor]	초기 제안단계로, design repo를 통해 제안
1	Feature Proposal [Community Group]	초기 제안 진행여부 투표 통과
2	Proposed Spec Text Available [Community + Working Group]	영문 텍스트 제안서 제공
3	Implementation Phase [Community + Working Group]	구현과 테스트 세트
4	Standardize the Feature [Working Group]	2개 이상의 Web VM, 1개 이상의 툴체인에서의 구현과 커뮤니티 그룹의 승인
5	The Feature is Standardized [Working Group]	완료. 기능적 완료에 대한 워킹 그룹 구성원들간의 공감대 형성

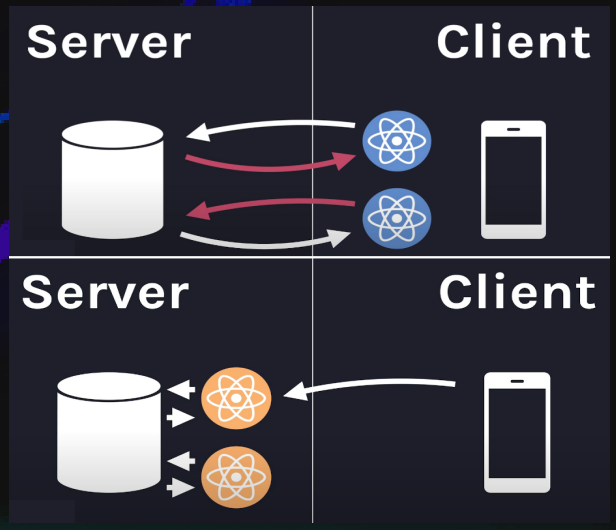
Frameworks

Another day, another new JavaScript Framework

React

Server Components

새로운 유형의 컴포넌트 제안



- 클라이언트와 서버 통신은 결과적으로 느리다.
- 컴포넌트를 서버로 이동, 데이터 처리는 서버에서 이뤄지도록 한다.
- 서버 컴포넌트는 번들(webpack으로 번들)에 포함하지 않고 필요한 코드만 로딩되게 한다.
- 개발 진행을 통해 `18 ~ 29%` 번들 크기 감소
- SSR을 대체하는 것이 아님

컴포넌트	확장자	의미
Server	`.server.js`	BE 리소스(DB, 파일시스템 등)에 직접 접근
Client	`.client.js`	현재의 일반적 클라이언트 컴포넌트를 의미 BE 리소스 접근되지 않는다.
Shared	`.js`	데이터 변환 처리 로직만을 포함, BE 리소스의 상태/영향을 사용하지 않아 어디서든 사용가능하다.

💁‍♂️ Introducing Zero-Bundle-Size React Server Components

v18 Plan

21/6월 차기 버전에 대한 내용을 공개

- New Root API
- SSR support for Suspense

Concurrent features

- startTransition
  특정 상태 전환 업데이트에 대해 "transition"으로 처리해, 응답성을 유지
- useDeferredValue
  화면에서 덜 중요한 영역의 업데이트를 지연해, 중요한 영역이 우선되게
- `<SuspenseList>` (데모)
  하위 트리에 있는 `<Suspense>`의 공개와 로딩 인디케이터 노출 순서를 조율
- Streaming SSR with selective hydrations
  앱 로드 및 인터랙티브 속도 향상

The Plan for React 18

- Automatic Batching

  💡 `Batch?` 성능을 위해, 다중 상태 업데이트를 단일 재렌더링으로 수행

  React 이벤트 핸들러내 작업만 배칭에서, Promise, setTimeout 등으로 확장
- react-devtools
- React Working Group

  GitHub의 discussion을 사용한 워킹그룹으로
  생태계의 v18 점진적 채택을 위한 준비

  💁‍♂️ React Labs: React 18 Working Group Q&A [Audio]

21/5월, 첫 번째 alpha 공개했고, `@alpha` tag를 통해 다운로드 가능

```
npm install react@alpha react-dom@alpha
```

v18 릴리스 일정:

- 공개 베타: 최소 수개월 뒤
- RC: 공개베타 이후 수주 뒤
- GA: RC 이후 수주 뒤 공개

💁‍♂️ Installing React 18 Alpha

Vue.js

IE11 지원중단

Vue.js 3에 대한 IE11 빌드 제공에 대한 계획을 밝혔었지만,
최종적으론 지원 중단을 발표

그 이유로는

- MS 스스로도 IE 사용하지 않도록 권고 및 자사 서비스들의 지원중단 선언
- IE11 지원은 많은 비용 필요
- 꼭 필요한 경우라면, Vue 2를 사용할 것을 권고
  Vue 3의 중요한 기능들이 Vue 2.7에 구현될 예정

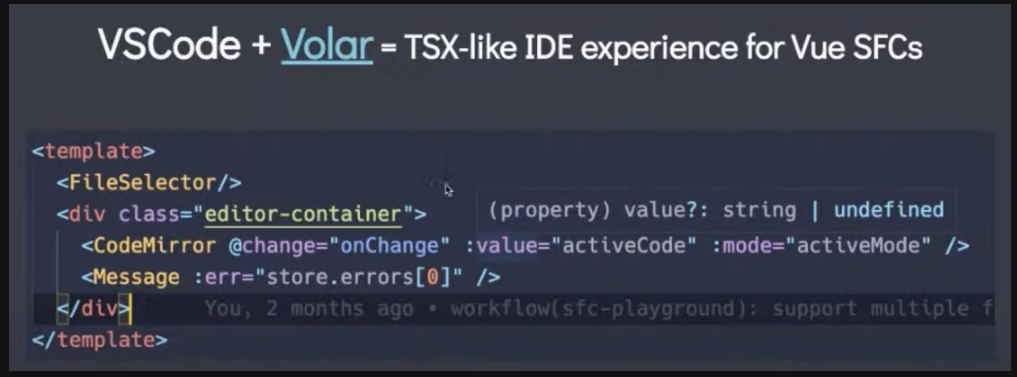
👨‍💻 vue3-ie11-support
Proposal for dropping ie11 support in Vue 3

DX 개선

Developer eXperience

- Authoring experience
  - `<script setup>`
    SFC의 새로운 스크립트 타입으로, 최상위 바인딩을 템플릿에 노출
  - `<style vars="{some}">` 변수 주입
    SFC 스타일에 컴포넌트 상태 CSS 변수 주입
  - Ref sugar
    반응형 변수 `$()` 사용을 위한 컴파일러 매크로 집합
- Vue Devtools 6.0 beta
  - Vue 2, 3 듀얼버전 지원
  - Vuex 통합 예정
  - 타임라인 뷰, 성능 프로파일링
- Better IDE/TS 지원
  - Vetur Vue tooling for VS Code
  - VueDX A set of tools for better DX for Vue

생태계: Volar



- Vue의 언어적 지원을 위한 VSCode IDE 플러그인
- VSCode 익스텐션과 CLI 타입검사 도구를 제공
- SFC(Single File Component)를 위한 TSX 같은 IDE 경험을 제공

Plan

- 새로운 VSCode 공식 익스텐션은 Volar를 기반 예정
- @vue/cli 타임검사 도구는 volar의 vue-tsc 기반

v3.2 (21/8월)

- 새로운 SFC 기능들: `<script setup>`, `<style> v-bind`: 컴포넌트 상태값 바인딩
- Reactivity 시스템과 템플릿 컴파일러 성능 개선
- SSR 개선: `@vue/server-renderer` 패키지
  Node.js 빌트인과 디커플링된 ESM 빌드 제공,
  비 Node.js 런타임에서 사용되도록 번들링 가능

  💁‍♂️ Vue.js Server-Side Rendering Guide
- 웹컴포넌트 요소인 Custom Elements 생성을 도와주는
  새로운 defineCustomElement 메서드

```
1  import { defineCustomElement } from 'vue'
2
3  const MyVueElement = defineCustomElement({
4  // normal Vue component options here
5  })
6
7  // Register the custom element.
8  // After registration, all `<my-vue-element>` tags
9  // on the page will be upgraded.
10 customElements.define('my-vue-element', MyVueElement)
```

2021/2Q 에 npm 배포 태그(latest)가 3으로 전환될 것이라고 밝혔지만,
현시점(21/10)까지도 v3은 `next` 태그에서 배포 중

💁‍♂️ Vue 3.2 Released!

@vue/compat

Vue 3 기반의 특별한 빌드로, Vue 2 앱이 3에서 Vue 2 모드로 실행될 수 있게 한다.
Vue 2 애플리케이션을 3로 업그레이드하거나
라이브러리들이 3를 지원하는데 사용될 수 있다.

```
// package.json
"dependencies": {
-   "vue": "^2.6.12",
+   "vue": "^3.1.0",
+   "@vue/compat": "^3.1.0"
    ...
},
"devDependencies": {
-   "vue-template-compiler": "^2.6.12"
+   "@vue/compiler-sfc": "^3.1.0"
}
```

기존 Vue 2 앱에, Vue 2를 제거하고 Vue 3와 compat를 설치하고,
앱의 실행시 오류와 경고 메세지에 따른 항목들을
수정한 후, compat 패키지를 제거하는 것으로 마이그레이션을 수행

👨‍⚖️ Migration Build

Angular

v12 (21/5월)

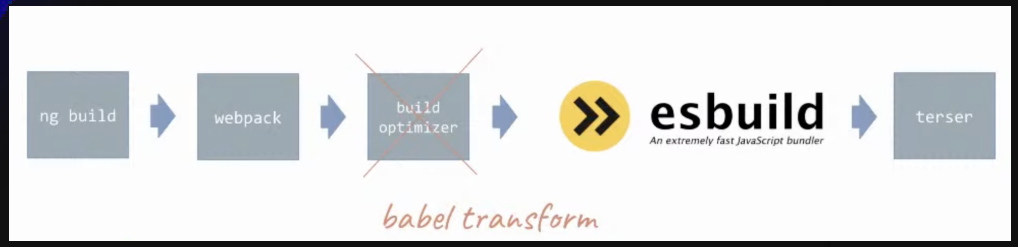
- `enableIvy:false` 설정은 더이상 View Engine을 사용한 애플리케이션 개발은 지원되지 않는다.
  v12로 작성된 애플리케이션은 모두 Ivy를 사용하게 된다.
- Tooling은 Webpack 5를 사용하며, Webpack 4 지원과 사용은 제거
- 새로운 빌드 옵션 `inlineStyleLanguage`: 스타일의 인라인 컴포넌트 스타일 정의
  - CSS(기본값), Sass, SCSS, LESS 지원
- Critical CSS 인라이닝은 기본 설정되며,
  해제하려면 `inlineCritical:false`를 설정해야 한다.
- `ng build`는 기본값으로 production 빌드를 생성
- 템플릿에서 Nullish Coaslescing(`??`) 지원

```
1 // 기존 템플릿 조건식
2 {{age !== null && age !== undefined ? age : calculateAge() }}
3
4 // Nullish Coalescing을 통해 단순화된 조건식
5 {{ age ?? calculateAge() }}
```

 Angular v12 is now available

v13 (21/11월 예정) & Roadmap

- IE11 지원 중단, View Engine은 제거 예정
- Modern Angular Package Format (APF) 제안
  - npm에 배포되는 `Ivy-native` 라이브러리 포맷으로, 기존 ViewEngine 라이브러리 배포 포맷을 대체
  - APF는 UMD 번들 제거, ES2020 output
- 빌드 경험 개선 `ng build`
  - CLI: JS optimizer로 `esbuild` + `terser` 도입



전체 빌드 파이프라인 개선과
`build optimizer ➡ Babel Transform` 전환을 통해 빌드 속도 `20%+` 향상

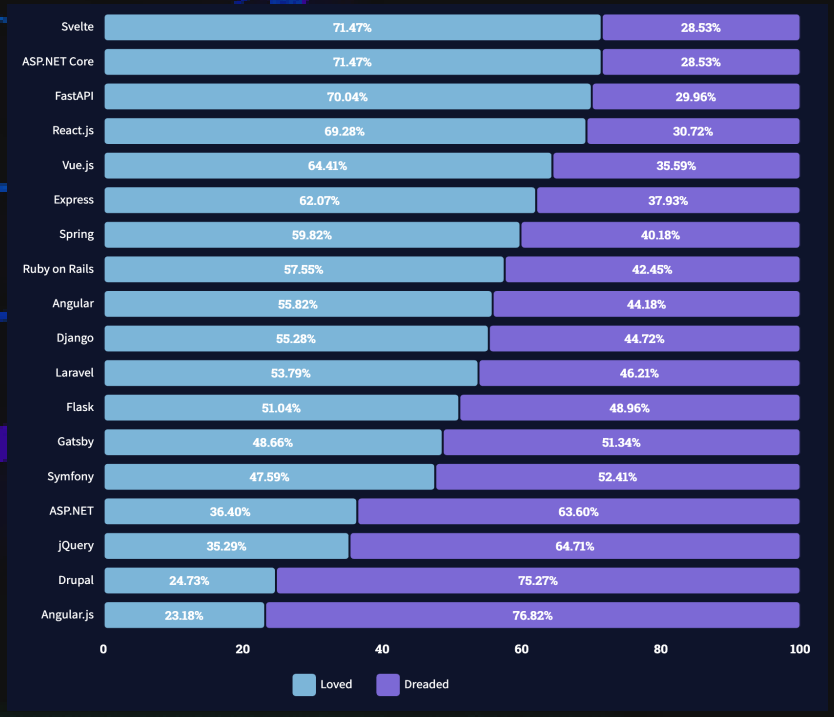
e2e 테스트 프레임워크 Protractor 미래 논의

2022년말 개발 중단 계획

- 2013년 처음 개발당시와 현재 환경 발전의 차이로 인해 테스트 재작성 없이 새로운 기술을 활용할 수 없는 문제
- 신규 프로젝트에 포함시키지 않고 선택사항으로 변경. CLI를 통해 다른 서드파티 포함 옵션 추가 계획
  Cypress, WebbdriverIO, TestCafe와 협업 진행중

💁‍♂️ Future of Angular E2E & Plans for Protractor

SVELTE



- Stackoverflow 설문조사에서 개발자들이 가장 관심(loved)갖는 프레임워크로 답변
- 전에도 그리고 앞으로 사용여부 질문엔 5위를 기록 (1위는 React)

👨‍💻 stackoverflow 2021 Developer Survey: Web frameworks

SVELTE KIT

Sapper를 계승하는 애플리케이션 프레임워크
Next.js의 Svelte 버전이라고 생각하면 된다.

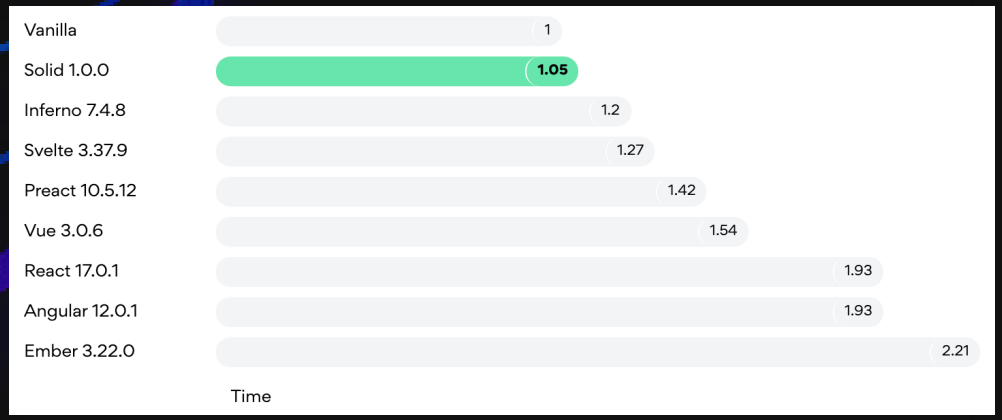
모던 기술 프랙티스를 활용해 Svelte 앱을
쉽게 개발할 수 있으며 파일 기반 라우팅을 제공한다.

- Snowpack ➡ Vite 전환
- SvelteKit SSR 향상을 위한 신규 패키지 `svelte/ssr`
- 21/3월 public bata 및 1.0까지 작업 중으로, 마일스톤 상으론 84% 완료

👨‍🏫 https://kit.svelte.dev/

SOLID

Solid는 사용자 인터페이스를 만들 수 있는 선언적이며,
유연한 라이브러리 VDOM 사용하지 않고,
템플릿을 통해 실제 DOM으로 컴파일



- 2016년 개발 시작,
  5년만에 1.0 릴리스
- JSX 문법 사용

React의 철학과 유사하게 단방향 데이터 흐름, 읽기/쓰기 분리,
불변성 인터페이스 원리를 따르지만
VDOM을 사용하지 않고 완전히 다르게 구현

💁 Official results for js web frameworks benchmark

Example

1초 인터벌마다 상탯값이 증가하는 카운터 예제

main.tsx

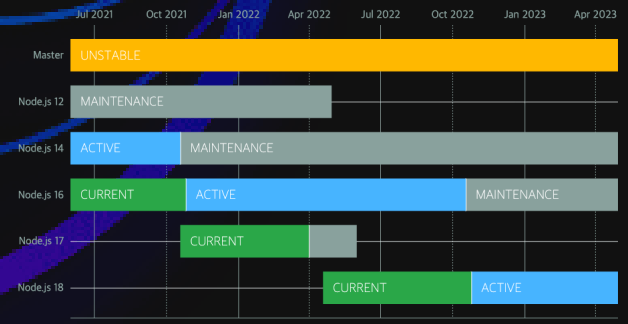
```tsx
import {render} from "solid-js/web";
import {onCleanup, createSignal} from "solid-js";

const CountingComponent = () => {
  const [count, setCount] = createSignal(0);
  const interval = setInterval(() =>
      setCount(count => count + 1)
    , 1000);

  onCleanup(() => clearInterval(interval));

  return <div>Count value is {count()}</div>;
};

render(() => <CountingComponent />,
    document.getElementById("app"));
```

Compiled output

```tsx
import {template, render, createComponent, insert} from 'solid-js/web';
import {createSignal, onCleanup} from 'solid-js';

const _tmpl$ = template(`<div>Count value is </div>`, 2);

const CountingComponent = () => {
  const [count, setCount] = createSignal(0);
  const interval = setInterval(() =>
    setCount(count => count + 1)
  , 1000);

  onCleanup(() => clearInterval(interval));

  return (() => {
    const _el$ = _tmpl$.cloneNode(true);
          _el$.firstChild;

    insert(_el$, count, null);

    return _el$;
  })();
};

render(() => createComponent(CountingComponent, {}),
    document.getElementById("app"));
```

💁 Example: Counter

Runtime



- N-API ➡ Node-API로 이름 변경
  Node-API는 네이티브 Node 애드온 개발에 사용되며, 종종 "NAPI"로 발음되었다.
  그러나, 경멸적 용어로 오인될 소지가 있어, 이름을 변경
- WASI(WebAssembly System Interface) 지원
  실험적 WASI API를 제공한다. WASI는 샌드박스된 wasm 애플리케이션에 OS의 POSIX 유사함수에 접근할 수 있게 한다.
- next-10
  성공적이었던 지난 10년을 기반으로, 향후 10년에 대한 전략적 방향성 논의를 위한 기록 저장소

💁 Release schedule

Corepack

Node.js 16.9에 기본 포함된 실험적 스크립트 도구
Node와 패키지 매니저(Yarn, Pnpm 같은)간 브릿지처럼 동작

npm 외의 서드파티 패키지 관리자를
전역에 설치하지 않고 사용할 수 있게 한다.

많은 경우 npm 사용이 문제 없지만, 필요에 따라
다른 패키지 관리자(현재 Yarn과 pnpm 지원)를 사용해야 하는 경우를 위해 제공

```
# corepack 사용을 활성화
$ corepack enable

# 프로젝트에 yarn을 설치하고 사용을 활성화 한다.
$ corepack prepare yarn@1.22.11 --activate

# 특정 패키지 매니저를 활성화 하는 경우
$ corepack enable yarn

# yarn을 사용해 axio를 설치
$ corepack yarn add axios

# 비활성화 하는 경우
$ corepack disable
```

```
// https://nodejs.org/api/packages.html#packages_p
// package.json
{
    "packageManager": "yarn@1.22.11",
    ...
}
```

https://github.com/nodejs/corepack
What is Corepack in Node.js?

Pure ESM

```
// package.json
{
  "type": "module
  ...
}
```

가장 가까운 부모의 pakcage.json에 `"type":"module"` 설정된 경우, `.js` 확장자는 ESM으로 로딩된다.

즉, 프로젝트 의존성 패키지가 `"type":"module"` 설정되었다면, 자신도 `"type":"module"` 설정필요

이 경우, `.js`는 ESM으로 처리되고, CJS는 `.cjs` 확장자를 가져야 한다.

그렇지 않을 경우

```
Error [ERR_REQUIRE_ESM]: Must use import to load ES Module: ...
require() of ES modules is not supported.
require() of ... from ... is an ES module file as it is a .js file whose near
contains "type": "module" which defines all .js files in that package scope a
Instead rename index.js to end in .cjs, change the requiring code to use impo
remove "type": "module" from .../package.json.
...
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1080:13)
    at Module.load (internal/modules/cjs/loader.js:928:32)
    ...
```

🤦 Pure ESM packages

사용자 측면

- CJS를 ESM으로 전환하고, package.json에 `"type":"module"` 설정
- 비동기 문맥에서는 `await import(...)`를 적용
- ESM 이동이 준비되지 않았다면, 현재 상태(의존성 패키지 버전 포함)를 유지

라이브러리 측면

Dual CJS/ESM

아직 생태계는 준비되지 않았으므로,
Dual CJS/ESM 패키지를 제공하는 것 필요

```
// package.json
{
  "type": "module",
  "exports": {
    "import": "./main-module.js",
    "require": "./main-require.cjs"
  },
}
```

Dual CommonJS/ES module packages

Deno

20/5월, 1.0 릴리스 이후,
사용 가능성과 확장성을 위한 작업들을 진행

- Deno Company 설립
  초기 투자에 Next.js의 Guillermo Rauch(Rauch Capital),
  Mozilla 주식회사 등이 참여
- Deno Deploy 발표
  - Deno CLI를 통해 배포할 수 있는 분산 시스템 환경
  - 21/10 현재, Beta 2 상태로, 무료로 제공되나 이후 유료 전환예정
  - 전세계 25개 리전을 통해 배포
- deno.land/x - 패키지 레지스트리
- deno 번들러: Packup 개발중
- MDN 호환성 테이블에 Deno 항목 노출되기 시작

Package Manager

GitHub의 npm 인수 (20/3월) 후,
주요 소식은 GitHub 블로그를 통해 공유

npm v7 (20/10월)

- Workspaces
  단일 최상위 레벨(root 패키지) 내에서 여러 패키지를 관리 지원 제공
- Peer dependency 자동설치
  v7 이전엔 개발자들이 직접 peer dependency 설치 필요했지만,
  새로운 알고리즘은 `node_modules` 트리에서 일치하는 패키지가 발견되도록 보장
- 새로운 package-lock 포맷 v2, 그리고 yarn.lock 지원
  - 새로운 포맷은 재현 가능한 빌드 구성과,
    패키지 트리 구축에 필요한 모든 정보를 포함
  - v7 전에는 무시되었던 `yarn.lock`은
    패키지 구성과 가이던스 소스로 활용 가능

💁 Presenting v7.0.0 of the npm CLI

npm v8 (21/10월)

v8이 1년여 만에 새로 릴리스 되었고,
Node.js의 LTS 생명주기에 맞추기 위한 목적

- 이에 따라 `node<12` 지원 중단
- 지원되는 Node.js 버전이 아닌경우, 설치되지 않도록 변경
  `npm@6`가 `node@8` 환경에서 `npm@7` 설치가 가능해 발생했던 이슈를 제거

💁‍♂️ npm CLI upgraded to version 8
[RRFC] Breaking changes for npm@8

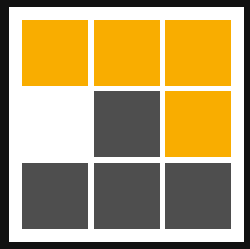
- 참여 확대: 메인터너들 중, Facebook 소속은 이제 없다.
- v3 작업 시작
  - Node 10 지원중단
  - Node corepack 지원
  - esbuild 지원
    - esbuild를 사용해 Yarn 번들을 생성
    - @yarnpkg/esbuild-plugin-pnp

      ➡ Yarn과 esbuild를 사용할 수 있도록 해준다.
  - PnP 모드에서 ESM 지원

현재 RC 단계로, 다음과 같이 설치할 수 있다.

```
$ npm install -g yarn     # update the global yarn to lates
$ yarn set version berry  # enable v2
$ yarn set version 3.x    # update to v3
```

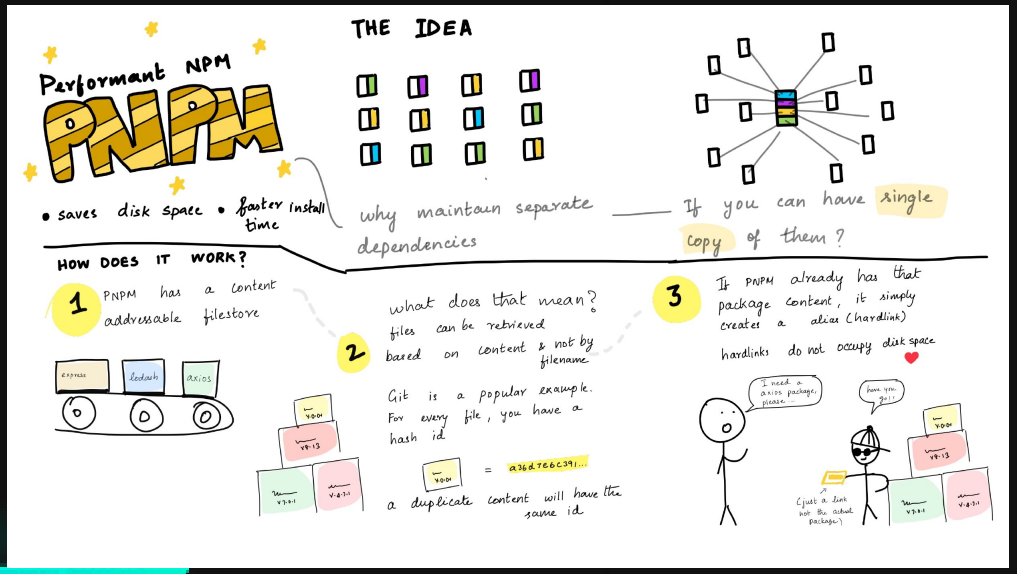
👨‍🎓 Yarn 3.0 🚀🤖 Performances, ESBuild, Better Patches, ...
Yarn 3.0 Changelog



pnpm

2016년 처음 발표되었던, pnpm(Performant npm)의
가장 큰 특징은 '효율적 디스크 사용'

어떻게 동작하나?
단일 공간에 패키지를 저장하고, 심볼릭 링크를 통해 공유



https://pnpm.io/

npm은 모든 패키지들이 모듈 디렉토리 루트(`node_modules`)에
플랫하게 위치(hoist)하며, 동일 패키지들이 각 프로젝트별 설치된다.

플랫하게 만드는 이유는 소스 코드들이 프로젝트 의존성으로
추가되지 않는 경우라도 접근이 가능하게 하기 위함

pnpm은 여러 프로젝트들에서 의존성을 갖는 동일 패키지를 한 번만 설치하고,
심볼릭 링크로 연결해 디스크를 효율적으로 사용한다.

```
$ npm install express

/node_modules/
    .bin/
    accepts/
    array-flatten/
    body-parser/
    bytes/
    content-disposition/
    cookie-signature/
    ...
    # 총 48개 디렉토리 생성
```

```
$ pnpm add express

/node_modules/
    .pnpm/    # virtual store directory
    .modules.yaml
    express # Symbolic Link

  # express 심볼릭 링크엔 다음과 같은 내용이 포함된다.
  # .pnpm/express@4.17.1/node_modules/express
```


https://github.com/zkochan/comparing-node-modules

6

도구의 전성시대

Tools

Bundler/Build

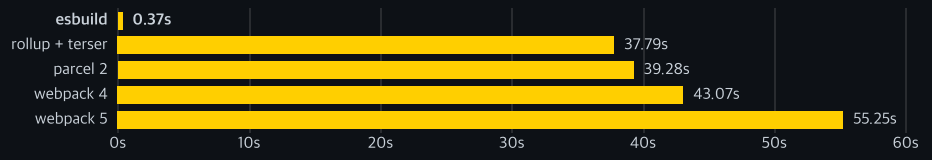
새로운 트렌드?

- 비 JavaScript 언어를 JavaScript 도구에 사용
  - Rust: Deno, swc, Rome, dprint, Volta, fnm
  - Go: esbuld
  - OCaml: Flow
- 개발 시점에선 unbundled 방식의 접근
- 빌드 도구들이 다른 빌드/번들 도구에 의존
- 스캐폴딩 + 빌드/번들러 형태로의 확장

esbuild

Go로 작성된 번들러로, 20/1월 Evan Wallace에 의해 개발되었다.

빌드 도구의 성능에 새로운 시대를 여는 것을 목표하며,
TS/JSX 등 모던 번들러가 제공하는 기능들이 기본 제공된다.



esbuild를 사용하는 도구들:

- Yarn
- Angular CLI
- Vite ➡ SvelteKit
- Packup

💁‍♂️ https://esbuild.github.io

Vite

quick을 뜻하는 프랑스어 “Vite”(`/vit/`, “veet”로 발음)는
Evan You가 개발(20/4월 개발시작)한 빌드 도구다.

- 개발시 번들링 수행않고, ESM으로 로딩
- prebundling(CJS/UMD ➡ ESM 변환)은 esbuild, 배포 버전은 Rollup 사용.
- Rollup 플러그인 호환

```
netil@vite $ npm init vite@latest my-app
```

기존 vue-cli와 vite는 일단 공존하나,
장기적으론, 2개 도구의 통합 필요
➡ Vite(스피드) + vue-cli(포괄적인 지원성)

🤦 https://vitejs.dev/

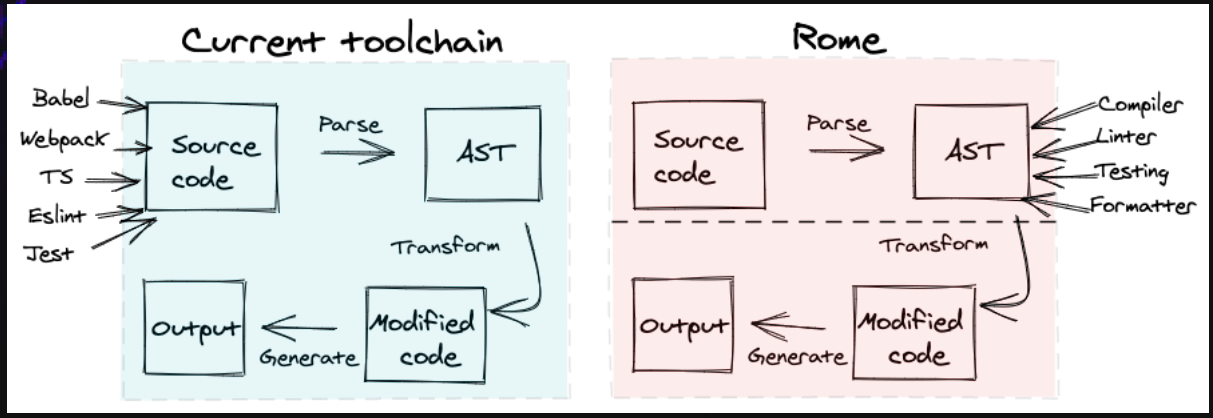
6to5(Babel)를 개발했던 Sebastian McKenzie의
프로젝트로, 20/8월 첫 베타 릴리스

Rome is a
JavaScript Compiler

Babel, ESLint, Webpack, Prettier, Jest 등
오늘날 모던 웹 애플리케이션 개발을 위해 필요한
툴 체인들을 단일 도구로 대체하기 위한 목적

🤦 https://rome.tools/

기본 아이디어는 단일 AST 생성을 재사용 하는것.



🤦 Do all roads lead to Rome?

- 20/5월 Rome Tools Inc 설립
  초기 투자에 GitHub 공동창업자인 Tom-Preston Werner도 참여
- 21/9월, Rust로 재작성 계획 발표
  Rome 내부에서 타이트하게 모든 코드들을 관리해 성능,
  메모리 사용량 그리고 정확성(correctness)과
  타입 안정성을 제공하기 위해서 필요

PARCEL

2019년에 시작되었던 v2 작업은 오랜 시간 끝에 21/10월 릴리스

- swc 를 컴파일러로 사용
  Babel AST 기반 컴파일러 ➡ swc 기반 Rust로 작성
- Tree shaking 기본 활성화, 자동 코드분할(code splitting)
- Automatic differential bundling
  모던 브라우저를 위한 네이티브 ESM과 레거시 브라우저를 위한 fallback 2가지 버전 제공
- 향상된 JSX 지원 React 17의 JSX 런타임 지원

💁 Announcing Parcel v2!

WMR

Preact 개발자인 Jason Miller가 개발
의존성 없는 단일 파일 통합 개발도구

The tiny all-in-one development tool for modern web apps

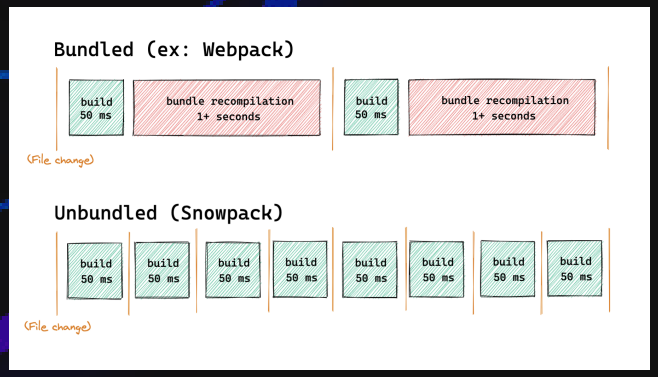
- 엔트리 포인트 없이 스크립트 로딩만을 통해 실행
  `<script type="module" src="/index.js">`
- 스캐폴딩을 위한 `create-wmr` 제공
- npm 패키지의 설치없이 import
- HMR: Preact 컴포넌트와 CSS
- 빌트인 TS 및 JSX 브라우저 디버깅 기능
- 정적 자원들에 대한 hot reloading
- Rollup 플러그인 지원 (Rollup 사용되지 않는 경우에도)

💁 https://wmr.dev/

Snowpack

@pika/web (2019) ➡ Snowpack (2020)

Unbundle 개발 철학



- 개발 모드에선 번들링 하지 않고, 각 파일은 빌드 후 캐싱 개별 파일들은 네이티브 ESM으로 로딩
- npm의 CJS 모듈은 어떻게 로딩할까? 브라우저 실행을 위해 snowpack은 이들을 단일 파일로 번들링하고 네이티브 ESM으로 사용될 수 있게 한다.

💡 SvelteKit에서 채택되었지만, Vite에 밀려났다.

🤦 https://www.snowpack.dev/
A Future Without Webpack

Progressive Web Apps

PWA

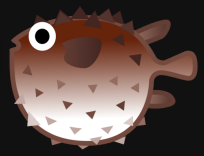
브라우저 벤더들의
서로 다른 방향성

Google, MS, Samsung 진영

"가능성/능력"(capability)에 대한 고려를 통해,
웹 플랫폼에서 더 많은 것을 수행할 수 있는 가능성/능력에 적극성을 띈다.

Apple, Mozilla 진영

"개인정보 보호"(Privacy) 이슈를 가장 최우선으로 고려하기 때문에
웹에서의 새로운 가능성을 제공하는 기능들에 부정적 입장을 취한다.



Fugu 프로젝트

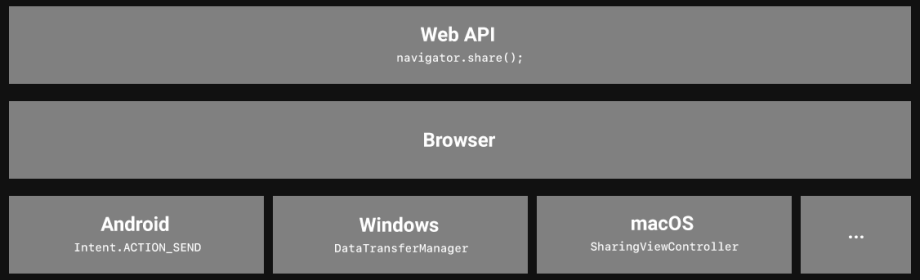
Web Capabilities

2019년 주요 벤더들(Google, MS, Intel, 삼성)이
네이티브(모바일/데스크톱) 앱에서 가능한 것들을
웹앱에서도 가능하게 만들기 위해 시작한 프로젝트

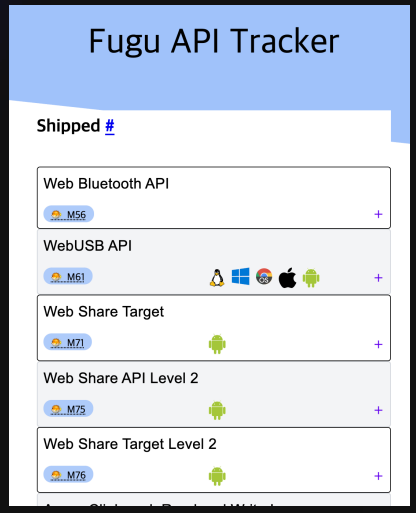
We believe web apps should be able
to do anything native apps can.

네이티브 기능들이 웹에 노출되더라도, 사용자 보안, 신뢰 및
개인정보 보호 같은 핵심적 원칙은 유지되어야 한다

"Fugu"는 복어(ふぐ)를 뜻하는 일본어로, 복어 요리는 특성상
잘 준비되면 맛있는 요리지만, 독이 제거되지 못하면 치명적



- 웹에 필요한 네이티브 API를 평가하고, Web API로 노출
- Web API는 OS 네이티브 API와 애플리케이션간 추가적 레이어로 동작

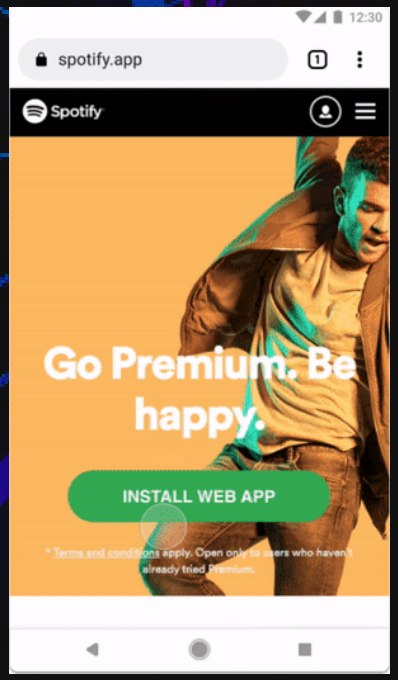


Fugu API Tracker

현재 진행되고 있는 Fugu API들의
구현 및 진행 상태 확인

🤦 New capabilities status
Web Capabilities (Project Fugu 🐡)

Richer Install UI



- M91, Dev/Canary 버전에서 `#mobile-pwa-install-use-bottom-sheet` 플래그 활성화시 사용가능
- manifest.json에 `screenshots`과 `description` 필드 추가하면 끝.

PWA Richer install UI on Mobile

Richer PWA installation UI
PWA Summit 2021: Make your PWAs look and launch beautifully

WebKit

WebKit 팀은 블로그를 통해 보안 및 개인정보 보호 우려에 따라
다음의 Web APIs 구현하지 않는다고 발표

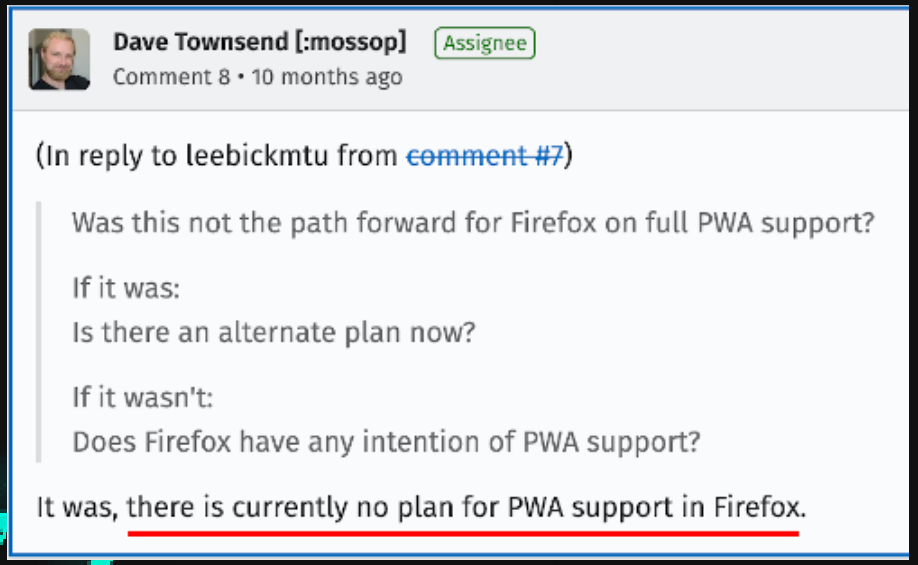
- *Web Bluetooth*
- *Web MIDI API*
- *Magnetometer API*
- *Web NFC API*
- *Device Memory API*
- *Network Information API*
- *Battery Status API (Firefox도 구현했다가 제거)*
- *Web Bluetooth Scanning*
- *Ambient Light Sensor*
- *HDCP Policy Check extension for EME*
- *Proximity Sensor*
- *WebHID*
- *Serial API*
- *Web USB*
- *Geolocation Sensor (background geolocation)*
- *User Idle Detection*

💁 Tracking Prevention in WebKit

moz://a

PWA 방향성 전환?

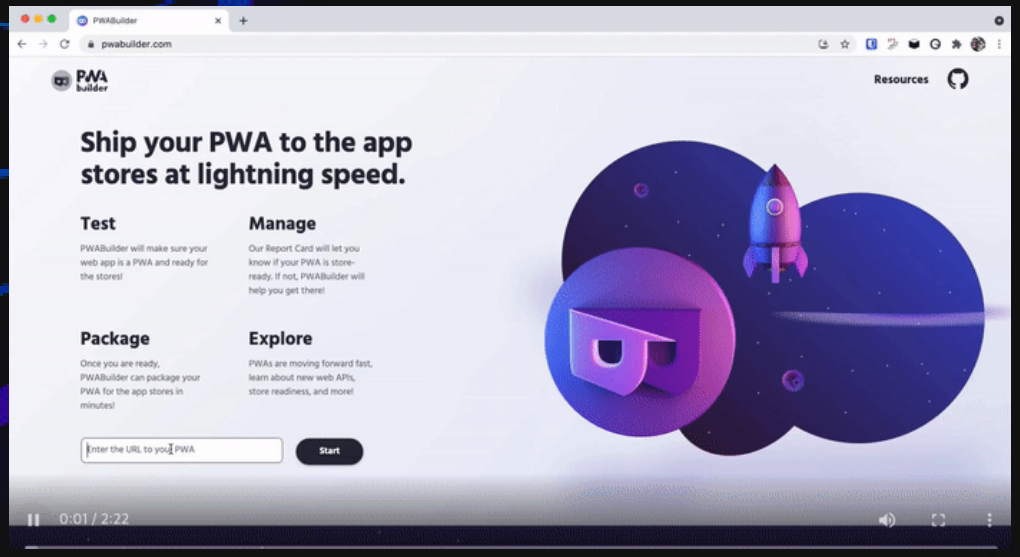
데스크톱 지원을 위한 실험적 기능인
SSB(Site Specific Browser)를 제거하고,
PWA 지원 계획 없다고 코멘트



Store 배포

가장 간단한 방법은 MS/Google이 개발에 참여하고 있는

PWA builder 를 사용하는 것이다.



PWABuilder는 기본 코어로
구글의 Bubblewrap을 활용

AppStore는 정책 변경에 따라 지원이 중단되었다.
Where is the iOS target?

🤦 PWA Builder

PWA Summit 2021: Bringing your PWA to the app store (발표 슬라이드)

Android/MS

Android

Trusted Web Activity(TWA) 사용
Android 앱에서 Custom Tabs 기반 프로토콜을 사용, 웹앱 콘텐츠를 여는 방법

💡 WebAPK?

PWA 앱을 '홈 화면에 추가'를 통해 설치시,
크롬에서 자동 생성해 설치하는 특별한 APK(Android Application Package)로
이를 통해 PWA앱이 네이티브 앱처럼, 앱런처나 Android 설정 등에 노출되게 만든다.

MS

- Easily distribute your PWAs and get more exposure
- Microsoft Store에 PWA 제출
- MS - Welcoming Progressive Web Apps to Microsoft Edge and Windows 10

Apple App Store

iOS/iPad 14, Big Sur를 통해 추가된 "App-Bound Domains"를 통해 가능하다.

💁‍♂️ WWDC 2020: App-bound domains

이 기능은 웹뷰의 네비게이션을 작은 규모의 origins 셋으로 제한하도록 만든다.
이는 네이티브 코드의 웹 콘텐츠 상호작용을 줄이고,
웹뷰에서 API 강요성을 높인다.

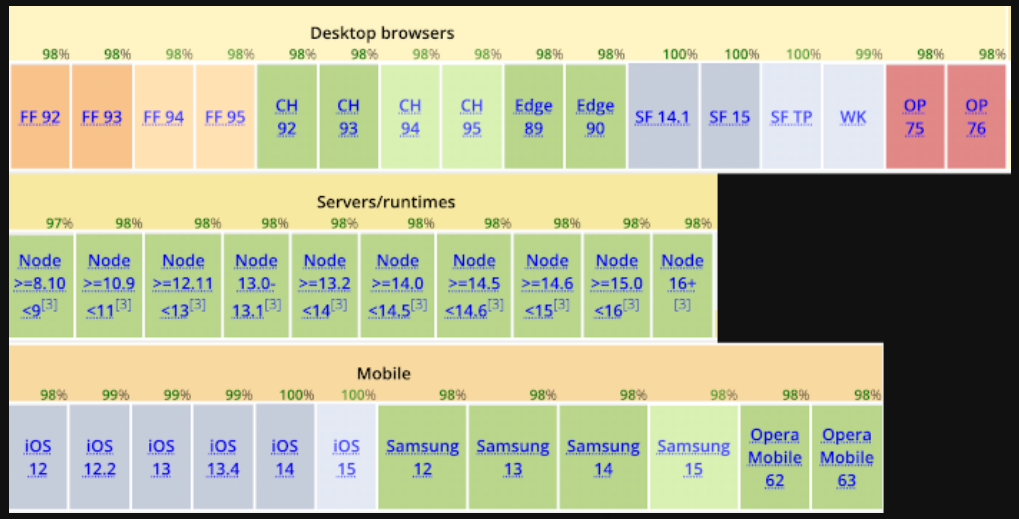
App-Bound Domain이 활성화 된 경우,
PWA의 핵심적 API인 ServiceWorkers를 사용할 수 있게된다.

💁‍♂️ Publishing Progressive Web Apps (Pluralsight: 유료강의)

네이티브로의 이동

ES6 지원율

- 주요 브라우저와 런타임에서 98~100% 도달
- 주요 프레임워크들(Vue 3, Angular 13)과 도구(Wordpress)들의 IE11 지원중단

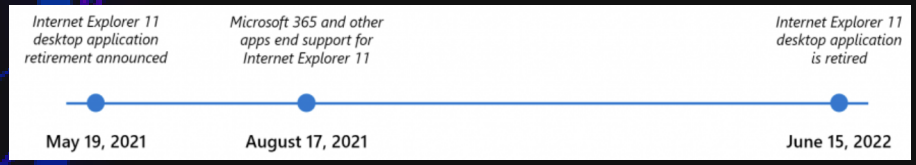


우리들은 계속 ES5(또는 레거시 브라우저)를 위해
코드를 Transpile할 필요성이 있을까?

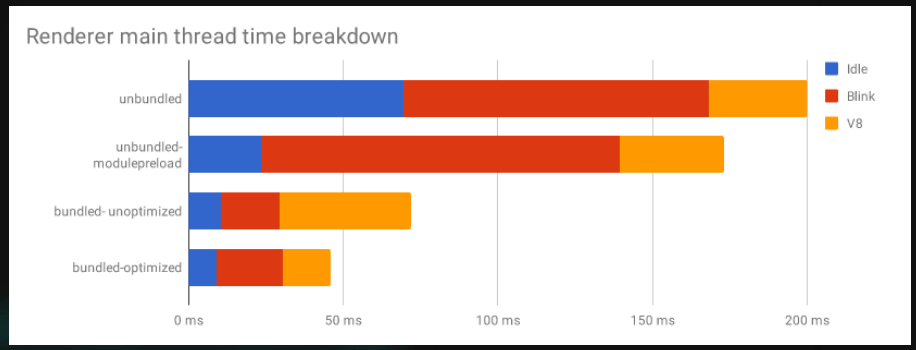
💁 ECMAScript 6 compatibility table

트랜스파일링,
번들링은 계속 필요한가?

- MS의 IE11 지원, 2022년 6월15일 종료 발표



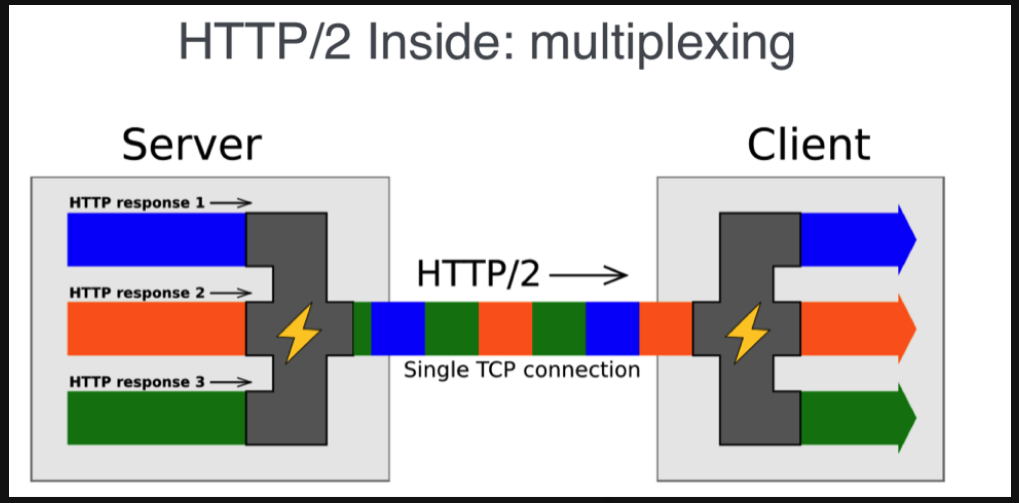
- Node.js Test Runner인 AVA는 Babel 트랜스파일링을 3.0 부터 제거
- 그러나 번들러는 import/export 구문을 정적 분석,
  사용되지 않는 export를 제거 최적화 제공 측면에선 아직 유용



🤦 ES Moduling loading
Modern web apps without JavaScript bundling or transpiling

HTTP/2 Multiplexing

HTTP/2는 Multiplexing을 통해,
단일 TCP 연결을 통해 다중 요청/응답을 처리한다.



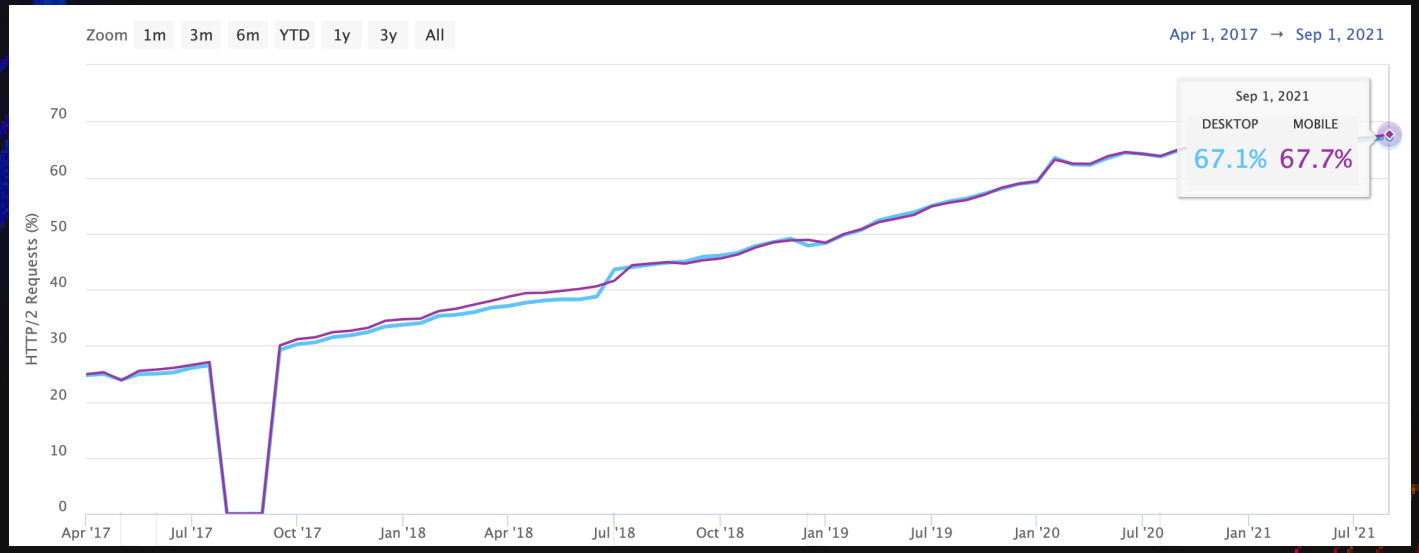
더는 `HTTP Request 감소`를 위한 다음같은 성능적 접근 유효성은 감소

- 여러개의 파일을 단일 파일로 번들링
- CSS Sprite

👨‍🏫 Request and Response Multiplexing

HTTP/2 점유율

2021/9 기준, 전세계 요청 중 67% 차지.



🤦 HTTP/2 Requests

네이티브 전환의 걸림돌

- JSX, TypeScript에 대한 트랜스파일링
- Tree-shaking과 같은 최적화 작업의 수행

만약, 도구 사용이 필요 없게 된다면?

- 👏 골치아픈 복잡한 설정에서 해방
- 🚀 개발 환경의 최적화와 성능 향상: Vite, Snowpack
- 👩‍🦲 작성된 코드가 트랜스파일러를 통해 변환되지 않으므로, 배포된 코드와 작성된 코드간 차이없어 디버깅 경험 개선

Wrap Up

2021년의 JavaScript & FE 생태계?

- WebAssembly 실사용 확산은 더디지만, 미래의 얼굴
- 프레임워크는 여전히 React 강세

  Who's Next?: Vue.js? SVELTE ? SOLID ?

- `TypeScript`, `JSX` 는 점점 de-facto
- IE11은 이제 그만
- BFF 또는 BE 영역의 FE 기술사용은 확장되며, 가속화
- PureESM: CJS ➡ ESM 전환
- Toolchain: non JavaScript for JavaScript
- PWA의 메인 스트림으로의 성장은 더디고 요원

  ➡ 결국, WebKit의 향방에 따라

고맙습니다.
Thank You.
Gracias.

Support us by,

billboard.js / naver/fe-news

NAVER ❤ FRONT-END