Progress®

The Future of JavaScript: 2018 and Beyond

WHITEPAPER

Authors

Tara Z. Manicsic

Tara Z. Manicsic is a lifelong student, teacher, maker and, recently, a Google Developer Expert. She has spent her career using JavaScript on both back-end and front-end to create applications. In her free time she works in her community to educate and learn from other developers. Tara launched & directs the Cincinnati Chapter of Women Who Code and the Cincinnati branch of NodeSchool. Beyond code, she likes to make things with other materials (wool, solder, clay, etc.) and hikes any mountain she can get to with her trusty sidekick, #toshmagosh.

Alyssa Nicoll

Alyssa Nicoll is an Angular Developer Advocate & GDE. Her two degrees (Web Design & Development and Psychology) feed her speaking career: she's spoken at over 20 conferences internationally, specialising in motivational soft talks. She's a weekly panelist on Adventures in Angular and Angular Air, which have a combined following of over 16,000 listeners. She enjoys gaming, scuba diving, and has a toothless dog named Gummy. Her DM is always open.

Editor

TJ VanToll

TJ VanToll is a front-end developer, author, and a Principal Developer Advocate for Progress. TJ has over a decade of web development experience, including a few years working on the jQuery team. Nowadays, he spends his time helping web developers build mobile apps through projects like NativeScript.

Table of Contents

As JavaScript's role in the software world grows, so does the importance of knowing where the language is going. Every year we at Progress take a look at the state of the JavaScript ecosystem, including the language itself, as well its usage in libraries, frameworks, mobile applications and more.

This year is no different. The following guide is a comprehensive look at what JavaScript has done in 2017, and where we see the JavaScript ecosystem going in 2018.

We'll start by taking a look at the JavaScript language itself, in **JavaScript's Journey Through 2017 and Into 2018**. You'll learn what features JavaScript enabled in the last year, what features are up and coming, and what features we see playing a critical role in your applications in the next few years.

Next, we'll return to JavaScript in the browser to discuss the rise and fall of popular JavaScript frameworks in **Libraries and Frameworks and Futures, Oh Yes!** We'll look at how these frameworks have done in 2017, and make some predictions about the JavaScript framework world for 2018.

JavaScript's Journey Through 2017 and Into 2018

by @tzmanics

Another year has gone by and JavaScript is still going strong. This is probably not a surprise to any of us. Hopefully, option overload and the complexity of coding in JavaScript has not made anyone throw their computer out the window. In reviewing JavaScript in the past year, there are a few topics that stand out that I'll be covering:

- ECMAScript Goodies
- How to JavaScript in 2018
- Package Manager Rumble

There are other things I could cover, if you think of any, add a comment, let's discuss! Okay, here we go (deep breath).

ECMAScript Goodies

Let's check in with ECMA International, Technical Committee 39! It turns out the 6 in ES6 does not stand for the number of years it takes for a release. I kid! Since ES6/ES2015 took so long to release (6 years, hence my jab) the committee decided to move to a yearly small-batch release instead. I'm a big fan of this and I think the momentum keeps things moving and JavaScript improving. What presents did we get for ES2017 and what's on our list for ES2018?

*You can learn more about the TC39 process of proposals here

ES2017

In January, at the TC39 meeting, the group settles on the ECMAScript proposals that would be slated as the features of ES2017 (also referred to ES8, which probably should be nixed to avoid confusion). This list included:

Major features

- Async Functions
- Shared Memory and Atomics

Minor features

- Object.values/Object.entries
- String padding
- Object.getOwnPropertyDescriptors()
- Trailing commas in function parameter lists and calls

Async/Await

Proposed by: Brian Terlson

I'm starting here because it was first on the list and my level of excitement is pretty high for this nifty addition. In ES2015 we got promises to help us with the all too familiar condition commonly known as...(are you really going to make me say it?) CALLBACK HELL.

The async/await syntax reads entirely synchronously and was inspired by TJ Holowaychuk's Co package. As a quick overview, async and await keywords allow you to use them and try/catch blocks to make functions behave asynchronously. They work like generators but are not translated to Generator Functions. This is what that looks like:

```
// Old Promise Town
function fetchThePuppies(puppy)
  return fetch(puppy)
    .then(puppyInfo => puppyInfo.text())
    .then(text => {
      return JSON.parse(text)
    })
    .catch(err => {{
      console.log(`Error: ${err.message}`)
    })
}

// New Async/Await City
async function fetchThePuppies(puppy)
  try {
    let puppyInfo = await fetch(puppy)
    let text = await puppyInfo.text()
    return JSON.parse(text)
  }
  catch (err) {
    console.log(`Error: ${err.message}`)
  }
}
```

This doesn't mean you should go in and replace all promises in your code with *async/await*. Just like you didn't go in and replace every function in your code with arrow functions (one hopes), only use this syntax where it works best. I won't go too into detail here because there are tons of articles covering async/await. Check them out (yes, I did add a link of a async/await blog post for each of those last words in the previous sentence, you're welcome). In the upcoming year we will see how people are able to make their code more, readable and efficient using async/await.

Shared Memory and Atomics

Proposed by: Lars T. Hansen

Wait, did we enter a theoretical physics class? Sounds fun, but no. This ECMAScript proposal joined the ES2017 line up and introduces SharedArrayBuffer and a namespace object Atomics with helper functions. Super high-level (pun intended), this proposal is our next step towards high-level parallelism in JavaScript.

We're using JavaScript for more and more operations in the browser relying on Just-in-Time compilers and fast CPUs. Unfortunately, as Lars T. Hansen says in his awesome post, A Taste of JavaScript's New Parallel Primitives from May 2016:

But JS JITs are now improving more slowly, and CPU performance improvement has mostly stalled. Instead of faster CPUs, all consumer devices — from desktop systems to smartphones — now have multiple CPUs (really CPU cores), and except at the low end they usually have more than two. A programmer who wants better performance for her program has to start using multiple cores in parallel. That is not a problem for "native" applications, which are all written in multi-threaded programming languages (Java, Swift, C#, and C++), but it is a problem for JS, which has very limited facilities for running on multiple CPUs (web workers, slow message passing, and few ways to avoid data copying).

SharedArrayBuffer

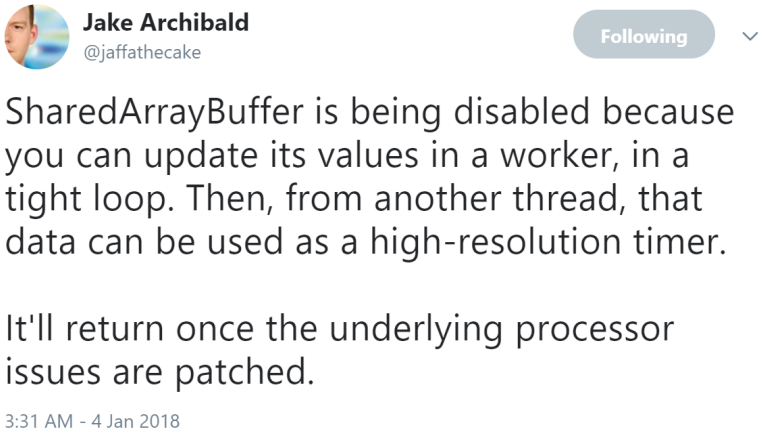
This proposal provides us with the building blocks for multi-core computation to research different approaches to implement higher-level parallel constructs in JavaScript. What might those building blocks be? May I introduce you to SharedArrayBuffer. MDN has a great succinct definition so I'll just plop that in right here:

The SharedArrayBuffer object is used to represent a generic, fixed-length raw binary data buffer, similar to the ArrayBuffer object, but in a way that they can be used to create views on shared memory. Unlike an ArrayBuffer, a SharedArrayBuffer cannot become detached.

Basically, one of the first ways we were able to run tasks in parallel was with web workers. Since the workers ran in their own global environments they were unable to share, by default, until communication between the workers, or between workers and the main thread, evolved. The SharedArrayBuffer object allows you to share bytes of data between multiple workers and the main thread. Plus, unlike its predecessor ArrayBuffer, the memory represented by SharedArrayBuffer can be referenced from multiple agents (i.e. web workers or the web page's main program) simultaneously. You can do this using postMessage to transfer the SharedArrayBuffer from one of these agents to the another. Put it all together, and what do you got? Transferring data between multiple workers and the main thread using SharedArrayBuffer so that you can execute multiple tasks at once which == parallelism in JavaScript. But wait, there's more!

SharedArrayBuffer Update

Before we move on it's important to note some current hold-ups for SharedArrayBuffer. If you've been paying attention to the news lately you may be aware of the processor chip security design flaw causing two vulnerabilities: Meltdown and Spectre. Feel free to read up on it but just know that browsers are disabling SharedArrayBuffer until this issue is resolved.



Atomics

Okay, the next stop on this parallel train: Atomics, which is a global variable that has two methods. First, let me present you with the problem the Atomics methods solve. When sharing a SharedArrayBuffer betwixt agents (as a reminder agents are the web workers or the web page's main program) each of those agents can read and write to its memory at any time. So, how do you keep this sane and organized, making sure each agent knows to wait for another agent to finish writing their data?

Atomics methods wake and load! Agents will "sleep" in the wait queue while waiting for another agent to finish writing their data, so Atomics.wait is a method that lets them know to wake up. When you need to read the data you use Atomics. load to load data from a certain location. The location is based on the methods two parameters a TypedArray, an array-like mechanism for accessing raw binary data (what SharedArrayBuffer is using), and an index to find the position in that TypedArray. There is more to it than what we've just covered but that's the gist of it.

For now, Atomics has only these two methods. Eventually, Hansen (our lovely author of this proposal and explainer of parallel things) says, there should be more methods, like store and compareExchange, to truly implement synchronization. Again, we are at the beginning stages of parallelism in JavaScript and this proposal is providing us with the building blocks to get there.

Phew! Although that was quite a lot to think about, that was still a high level overview. This update may not be used by most developers in the next year but will help advance JavaScript to benefit everyone. So, thank your brain for getting you this deep and check out these fantastic resources to dive in more!

- Dr.Axel to the rescue!
- A Taste of JavaScript's New Parallel Primitives from Lars T. Hansen

Object.values/Object.entries

Proposed by: Jordan Harband

Object.values()

I have actually benefited from the useful addition of Object.values recently when pulling Philips Hue light information from an observable. It allowed me to iterate through my data's values because it returns an array of the object's properties.

```
// land before `Object.values()`
const lights = {{ id: 1, on: true, color: ‘blue’}, { id: 2, on:
false, color: ‘red’ }}
this.lights = Object.keys(data).map(key => data[key])

// the time is now aka WITH `Object.values()`
this.lights = Object.values(data)

// both return
// [{ id: 1, on: true, color: ‘blue’}, { id: 2, on: false, color:
‘red’ }]
```

Fancy, right?

Object.entries{}

This method takes what Object.values() does one step further. Looking at an object, a data structure of key-value pairs, each of those pairs is an entry. When you call Object.entries() it is returning an array containing an array for each of those entries.

```
Object.entries({ name: 'Toshmagosh', age: 12 })
// [[ "name", "Toshmagosh" ], [ "age", 12 ]]
```

It's really quite straight forward, but if you ever want to dive in more, JavaScript.info has a good rundown.

String Padding

Proposed by : Jordan Harband, Rick Waldron

I didn't think I was going to have to say this again but, left pad. Yes, the left pad debacle of 2016 raised the attention of the JavaScript community enough for TC39 to add string padding. To be fair though, it was about time for JavaScript to have some native methods to handle Strings. Welcome padStart/padEnd to the family, which currently was just a lonely String.prototype.trim (est. ES5)!

You are all smart people so you probably can surmise what each of these methods do. So, I'll just show you some examples instead of using my words.

```
// padStart adds padding until string reaches provided length
'puppies'.padStart(22)
// "               puppies"

// or provide a filler instead of blank spaces
'nachos'.padStart(11, 'yum')
// "yumyunachos"

// padEnd works the same but adds to the end of the string
'Carlos Santana'.padEnd(30, '*-^')
// "Carlos Santana*-^*-^*-^*-^*"

// Emoji trickiness
'🍔🍟🍩🌭'.padEnd(8)
// "🍔🍟🍩🌭" // no pad? yup, because the length === 8, emoji you so
funny
```

To review this, both methods take an integer parameter that is telling them how long the final length of the string, including the padding, should be. If you pass a number shorter or equal to the original length of the string, nothing will change. Bravo, on wasting your time (just kidding). You can also pass a string and padStart/padEnd will repeatedly add each item of that string to the start or end of the original string until the length matches the passed length parameter. As you can see in my example above, since I wanted a length of 11, padStart added ` 'yum' then the 'yu' and stopped. Emoji are very important so I wanted to remind you of their tricky string lengths, more information in this handy blog post.

Developers will, more than likely, take advantage of these methods which will also let them remove libraries they were using to accomplish string manipulation. There are more methods in the pipeline: trimStart/trimEnd is currently at stage 2 (out of the 4 stages, here's the process break down again). This will let us trim or remove starts and ends of strings. Fun fact: this proposal started out with trimLeft and trimRight but has been updated to trimStart and trimEnd to stay consistent with padStart and padEnd. Yay, consistency! It also helps with any confusion whether a language is read right-to-left or left-to-right.

Object.getOwnPropertyDescriptors()

Proposed by: Jordan Harband & Andrea Giammarchi

This is the plural version of Object.getOwnPropertyDescriptor which returns a descriptor of the property that's directly on an object, i.e. not on its prototype chain.

```
let popcorn = { action: 'pop', butter: true }
let popcornAction = Object.getOwnPropertyDescriptor(popcorn,
'action')

// popcornAction is {
//   value: "pop",
//   writable: true,
//   enumerable: true,
//   writable: true
// }
```

So, using the plural version you are able to capture all of an object's non-inherited (or own) property descriptors. Using the delicious example above:

```
let popcorn = { action: 'pop', butter: true }
let popcornProperties = Object.getOwnPropertyDescriptors(popcorn)

// popcornProperties is {
//   action: {
//     value: "pop",
//     writable: true,
//     enumerable: true,
//     writable: true
//   },
//   butter: {
//     value: true,
//     writable: true,
//     enumerable: true,
//     writable: true
//   }
// }
```

Why do we need these methods? Well the proposer, Jordan Harband, puts it well here:

There is not a single method in ECMAScript capable of simplifying a proper copy between two objects. In these days more than ever, where functional programming and immutable objects are essential parts of complex applications, every framework or library is implementing its own boilerplate in order to properly copy properties between composed objects or prototypes.—Jordan Harband

With this addition you can now use getPrototypeOf and getOwnPropertyDescriptors with objectCreate to copy object and easily give it the same prototype and property descriptors. Before, this was most often done using Object.assign, which would grab an object's properties and symbols instead of descriptors. That approach left the risk of discarding possible accessors.

```
// just to give you a bit of an idea
const toshmagosh = {
  cuteLevel: 11,
  breed: 'Blue Pomeranian',
  treatTime: treat => {
    console.log(`Do you want a ${treat}?`)
  }
}

const newPuppy = Object.create(
  Object.getPrototypeOf(toshmagosh),
  Object.getOwnPropertyDescriptors(toshmagosh)
);

// newPuppy
// {cuteLevel: 11, breed: "Blue Pomeranian", treatTime: ƒ}
```

Appreciation Pause

Now, there are a lot of great people that are and have been on TC39, I would like to thank them all for their work. Doing this list has also put someone's name in writing multiple times: Jordan Harband. So, I just wanted to take a quick pause to Jordan, who is currently on a solid 1,325 day GitHub contributions streak as of December 1, 2017. Thanks for all you do for JavaScript, Jordan!!

Trailing Commas

Proposed by: Jeff Morrison

I must admit, I think trailing commas looks super sloppy and I have never been a big fan.

```
let why = [
  'really?',
  'must you?',
  'yuck😆',
]
```

That being said, I get it. I have had many occasions where I add an item to an array, a key value pair to an object, or delete an item and have had to remove or add a comma. I am one with the concept of minimizing how many keystrokes you must use. keysleft.com says I only have 213,407,968 and I just blew through 117 in this sentence alone! I've also heard the argument for the benefits this will add to checking your git diffs since you would only need to edit one line when adding function parameters, array items, etc. TBTH I'll probably take on this convention from now on. Okay, TC39? You win!

What's to Come in 2018

There is an awesome, emoji-laden table of proposals and what stages they are in located here. You can also see the finished proposals including one that is already ready for 2018 publication!

Dr. Axel is here to keep you up-to-date with the happenings of ES2018: http://2ality.com/2017/02/ecmascript-2018.html.

Usually, at this point we're looking at Stage 4 proposals, which are proposals that will definitely be added in the next release, and Stage 3 proposals, which are proposals that have a good chance of being included in the next release.

So far, the stage 4-ers are:

- Template Literal Revision proposed by Tim Disney. Currently, the escape sequence in template literals is problematic for embedding languages like domain-specific languages. This proposal will remove the restriction on escape sequences, to understand more click here

- s (dotAll) flag for regular expressions by Mathias Bynens. This proposal is all about emoji! Okay, not entirely, but it introduces the /s flag into regular expressions to make up for the dot's (.) shortcomings (like not matching with non-BMP character such as emoji). There is more to it though, so check out Mathias's proposal.

The list of stage 3-ers is a bit longer so I will give you this helpful link to see the table and an image of it down below. How nice is that.

Stage 3

	Proposal	Author	Champion	Tests
🚀	`Function.prototype.toString` revision	Michael Ficarra	Michael Ficarra	✅
	`global`	Jordan Harband	Jordan Harband	✅
	Rest/Spread Properties	Sebastian Markbåge	Sebastian Markbåge	✅
	Asynchronous Iteration	Domenic Denicola	Domenic Denicola	🚧
	`import()`	Domenic Denicola	Domenic Denicola	❓
	RegExp Lookbehind Assertions	Gorkem Yakin, Nozomu Katō, Daniel Ehrenberg	Daniel Ehrenberg, Mathias Bynens	✅
	RegExp Unicode Property Escapes	Mathias Bynens	Brian Terlson, Daniel Ehrenberg, Mathias Bynens	✅
🚀	RegExp named capture groups	Gorkem Yakin, Daniel Ehrenberg	Daniel Ehrenberg, Brian Terlson	✅
🚀	`s` (`dotAll`) flag for regular expressions	Mathias Bynens	Brian Terlson, Mathias Bynens	✅
	Legacy RegExp features in JavaScript	Claude Pache	Mark Miller, Claude Pache	✅
🚀	`Promise.prototype.finally`	Jordan Harband	Jordan Harband	🚧
	`BigInt`	Daniel Ehrenberg	Daniel Ehrenberg	🚧
	Optional `catch` binding	Michael Ficarra	Michael Ficarra	✅
	`import.meta`	Domenic Denicola	Domenic Denicola	❓
	Private methods and accessors	Daniel Ehrenberg	Daniel Ehrenberg, Kevin Gibbons	❓
	`Array.prototype.{flatMap,flatten}`	Brian Terlson, Michael Ficarra	Brian Terlson, Michael Ficarra	❓
	Numeric separators	Sam Goto, Rick Waldron	Sam Goto, Rick Waldron	🚧
	Class Public Instance Fields & Private Instance Fields & Methods	Daniel Ehrenberg	Daniel Ehrenberg, Jeff Morrison	❓

How to JavaScript in 2018

After discussing what changed in the standard used to create JavaScript, let's talk about how we USE JavaScript. Last year many people, including myself, were talking about JavaScript fatigue. Yes, the ways to write a JavaScript application have not really slimmed down, BUT with a lot of command-line tools doing much of the heavy lifting, transpiling becoming less crucial and TypeScript trying to minimize type errors, we can relax a little.

Command-line Tools

Most libraries and frameworks have a command-line tool that, with one command, will spin up skeleton projects for us to quickly create whatever our little hearts desire. This will often include a start script (sometimes with an auto re-loader), build scripts, testing structures and more. These tools are relieving us of a lot of redundant file making when we create new projects. Let's look at few more things some command line tools are taking off our plates.

Webpack configurations

Configuring your webpack build process and really understanding what you were doing, was probably one of the more daunting learning curves of 2017. Thankfully, they had one of their core contributors, Sean Larkin, running around the world supplying us with great talks.

Many frameworks nowadays not only create the webpack config files for you, but even populate them to the point that you may not even have to LOOK at it Vue's CLI tool even has a webpack-specific template giving you a full-featured Webpack setup. Just to give you the full idea of what command line tools are providing, here's what this vue cli template include, straight from the repo:

- npm run dev: first-in-class development experience.
  - Webpack + vue-loader for single file Vue components.
  - State preserving hot-reload
  - State preserving compilation error overlay
  - Lint-on-save with ESLint
  - Source maps
- npm run build: Production ready build.
  - JavaScript minified with UglifyJS v3.
  - HTML minified with html-minifier.
  - CSS across all components extracted into a single file and minified with cssnano.
  - Static assets compiled with version hashes for efficient long-term caching, and an auto-generated production index.html with proper URLs to these generated assets.
  - Use npm run build --reportto build with bundle size analytics.
- npm run unit: Unit tests run in JSDOM with Jest, or in PhantomJS with Karma + Mocha + karma-webpack.
  - Supports ES2015+ in test files.
  - Easy mocking.
- npm run e2e: End-to-end tests with Nightwatch.
  - Run tests in multiple browsers in parallel.
  - Works with one command out of the box:
    - Selenium and chromedriver dependencies automatically handled.
    - Automatically spawns the Selenium server.

The preact-cli, on the other hand, takes care of the standard webpack functionality. Then if you need to customize your webpack configurations you just create a preact.config.js file which exports a function that makes your webpack changes. So many tools, so much help; developers helping developers.

Babel On or Off

Get it? Sounds like Babylon. I crack myself up. I'm not exactly tying Babel to the ancient city of Babylon, but there has been talk of possibly removing our reliance on transpiling. Babel has been a big deal for the past few years because we wanted all the shiny that ECMAScript was proposing but didn't want to wait for the browsers to catch up. With ECMAScript moving to yearly small releases browsers may be able to keep up. What is a JavaScript post without some of the awesome kangax compatibility charts.

These images of these charts aren't legible because I wanted to showcase just how green they are! For full detail click the links below the images to inspect the charts further.

Image source: http://kangax.github.io

Compatibility for es6

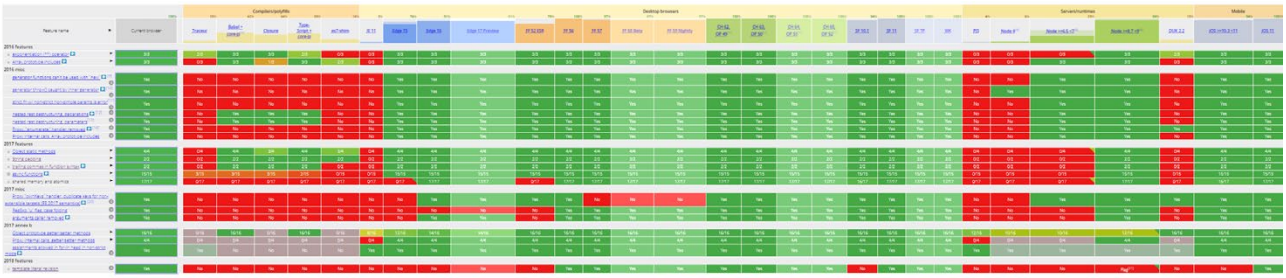


Image source: http://kangax.github.io

Compatibility for 2016+

In the first graph those red chunks on the left are compilers (e.g. es-6 shim, Closure, etc.) and older browsers (i.e. Kong 4.14 and IE 11). Then the five mostly red columns on the right are the server/compilers PJS, JXA, Node 4, DUK 1.8 and DUK 2.2. On the lower graph that red section that kind of looks like a bad drawing of a dog looking at a messed up exclamation point are servers/runtimes with only Node 6.5+ having green streaks. The makeup of the left red square are the compilers/polyfils and IE 11. More importantly, LOOK AT ALL THAT GREEN! In the most popular browsers, we have practically all green. The only red mark for 2017 features is on Firefox 52 ESR for Shared Memory and Atomics.

To put some of this into perspective here are some browser usage percentages from Wikipedia.

Usage share of all browsers for November 2017

Source	Chrome	Safari	UC	Firefox	IE	Opera	Samsung Internet	Edge	Android	Others
StatCounter	55%	14.76%	7.99%	6.1%	3.88%	3.8%	2.97%	2.06%	1.77%	1.68%

Usage share of all browsers for November 2017

Source	Chrome	Safari	IE	Firefox	Edge	Android	Opera	UC	Yandex	AppleMail	Amazon Silk	Others
Wikimedia	48%	20.8%	8.2%	6.6%	2.1%	1.3%	1.3%	0.6%	0.4%	0.2%	0.1%	10.5%

Usage share of mobile browsers for November 2017

Source	Chrome	Safari	UC	Samsung Internet	Opera	Android	Firefox	IE Mobile	QQ Browser	Others
StatCounter	49.58%	18.76%	15.19%	5.94%	5.54%	2.59%	0.68%	0.49%	0.29%	0.94%

Usage share of tablet browsers for November 2017

Source	Safari	Chrome	Android	UC	Opera	Others
StatCounter	61%	24.16%	11.04%	0.99%	0.99%	1.82%

Usage share of desktop browsers for November 2017

Source	Chrome	IE	Firefox	Edge	Safari	Others
NetMarketShare	60.61%	12.04%	11.42%	4.21%	3.85%	7.87%

Usage share of desktop browsers for November 2017

Source	Chrome	Safari	Firefox	IE	Opera	Edge
W3Counter	59.2%	14.3%	9.3%	5.4%	4%	2.7%

Usage share of desktop browsers for November 2017

Source	Chrome	Firefox	IE	Safari	Edge	Others
StatCounter	64.02%	12.55%	8.47%	6.08%	4.29%	4.59%

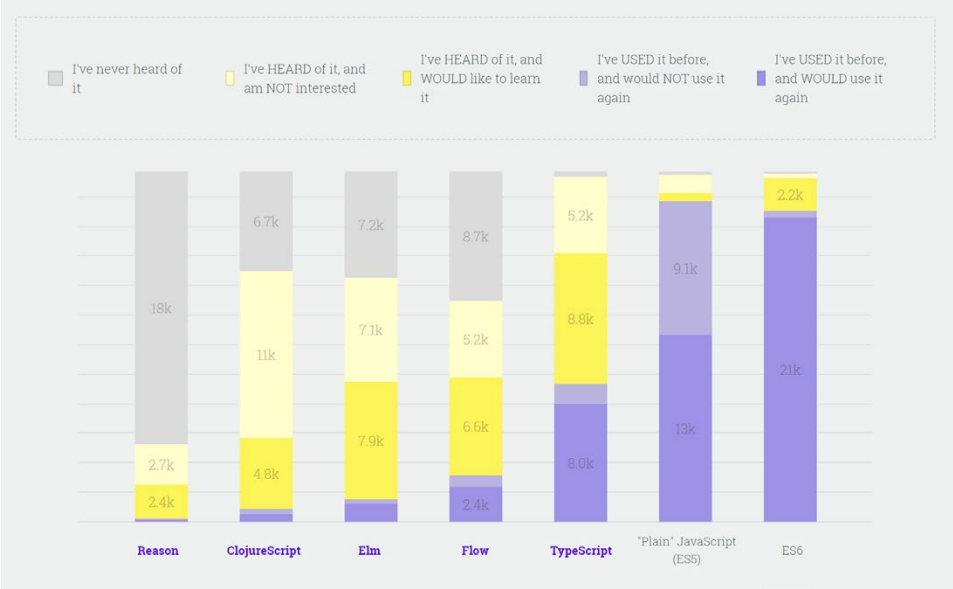
Note: Percentages are found by only looking at the "Browsers, non mobile" list.

Okay, turning off Babel may be a long ways aways because when it comes down to it we want to make a concerted effort to be accessible to as many users as we can. It is interesting to consider that we may be able to get rid of that extra step. You know, like before, when we didn't use transpilers.

TypeScript Talk

If we're talking about how to JavaScript we must talk about TypeScript. TypeScript came out of the Microsoft office five years ago but has been the cool kid in town in 2017. There was rarely a conference that didn't have a "Why We Love TypeScript" talk; it's like the new dev heartthrob. Without writing a sonnet to TypeScript let's talk a bit about why developers are crushing hard.

For everyone who wanted types in JavaScript, TypeScript is here to offer a strict syntactical superset of JavaScript which gives optional static typing. Pretty cool, if you're into that kind of thing. Of course, if you take a look at the newest results from the State of JavaScript survey, it seems that a lot of people ARE, in fact, into that kind of thing.



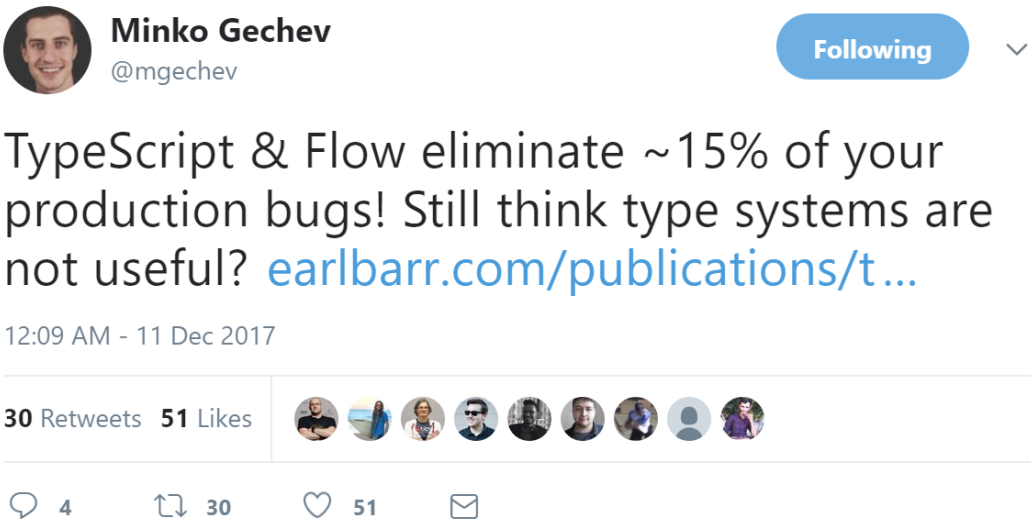
From State of JavaScript

To hear it straight from the source, check out this quote from Brian Terlson:

Speaking as someone who proposed types for JavaScript in 2014: I do not believe types are in the cards for the near future. This is an extremely complex problem to get right from a standards perspective. Just adopting TypeScript as the standard would of course be great for TypeScript users, but there are other typed JS supersets with pretty significant usage including closure compiler and flow. These tools all behave differently and it's not even clear that there's a common subset to work from (I don't think there is in any appreciable sense). I'm not entirely sure what a standard for types looks like, and I and others will continue to investigate this as it could be very beneficial, but don't expect anything near term - HashNode AMA with Brian Terlson

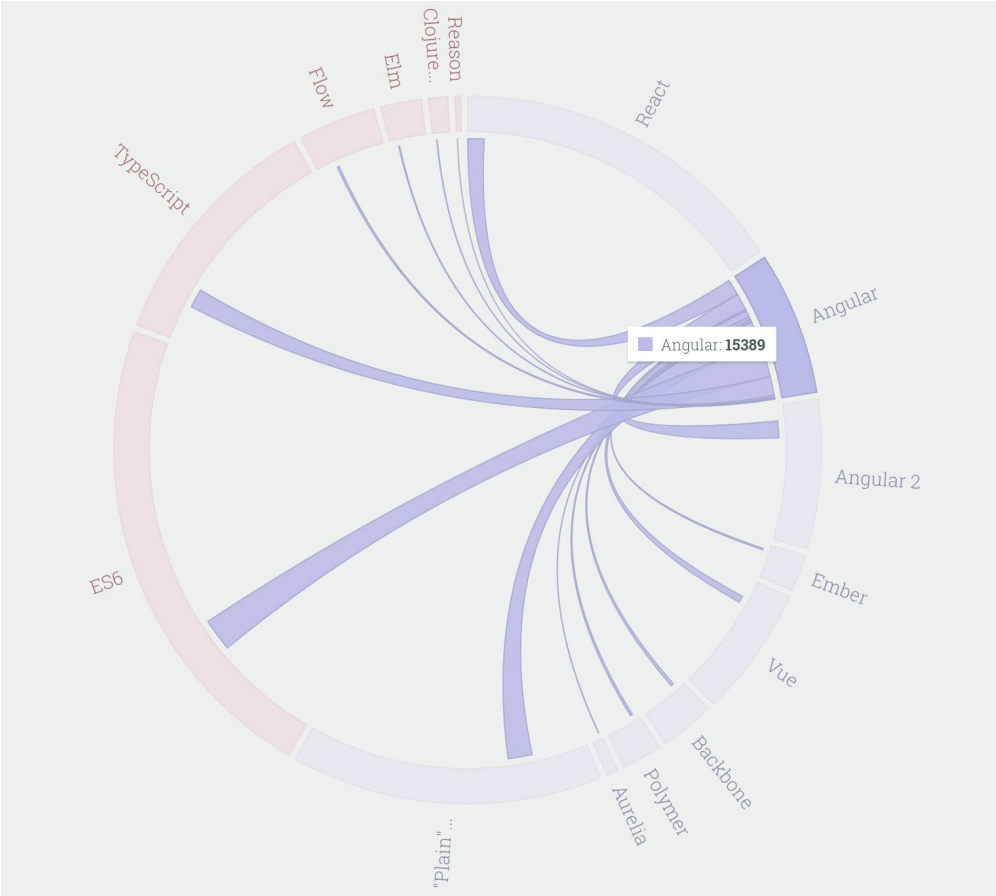
TypeScript ❤️s Flow

In 2017, you have probably seen many blog posts discussing the TypeScript + Flow combo. Flow is a static type checker for JavaScript. Flow, as you can see in the State of JavaScript survey chart list above, has about as many people interested as they do uninterested. More interesting is the stats showing how many of the people surveyed haven't heard of Flow, yet. As people learn more about Flow in 2018 maybe they will find it as beneficial as Minko Gechev does:



Angular ❤️s TypeScript

One may notice that all the code samples in Angular documentation are written in TypeScript. At one point, there was an option that you could choose to walk through the tutorial in JavaScript or TypeScript but it seems Angular's heart has been swayed. Looking at the chart below connecting Angular to JS flavors we can see that there is actually a tiny bit more users connecting Angular to ES6 (TypeScript: 3777, ES6: 3997). We'll see if all of this affects Angular in 2018.



From State of JavaScript

Undoubtedly, the way we JavaScript (used as a verb here) will evolve in 2018. As programmers we like to make and use tools that make our lives easier. Unfortunately, that can sometimes lead to more chaos and too many choices. Thankfully, command line tools are relieving us of some grunt work and TypeScript has satiated the type-hungry who were sick of type errors.

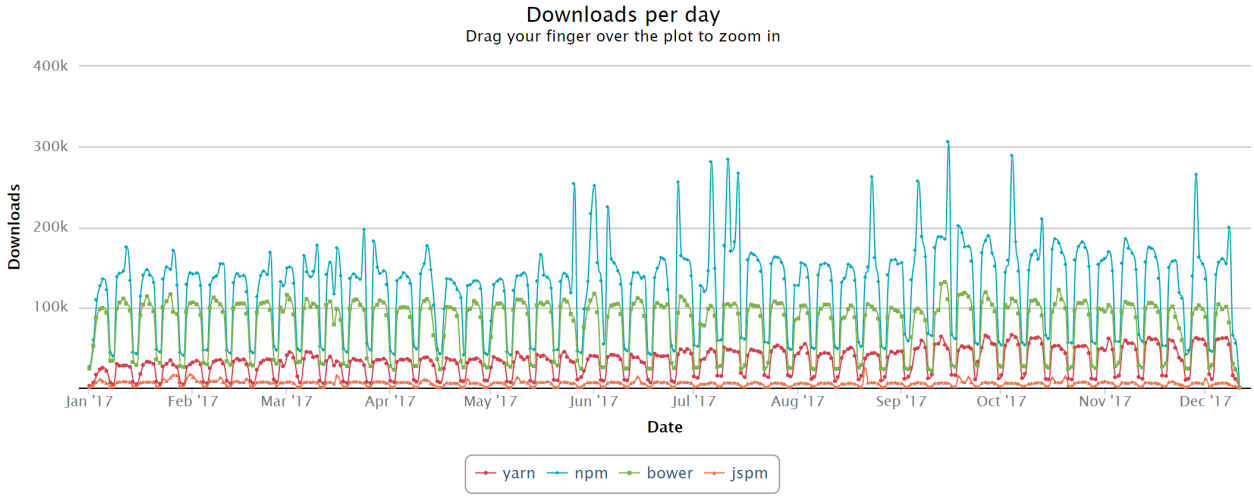
Package Manager Rumble

Along the same lines of how to use JavaScript is the discussion of package managers. Modules help us utilize tooling we and other developers make because WHY would you spend time re-writing something that already exists and works well?? If that question has not popped into your head or been repeated in a team meeting at least once in 2017...you might be doing it wrong. Just sayin'.

Thankfully, we have teams creating better and better experiences for us to install and organize these modules. Npm, Yarn and Bower are still the leaders of the pack...age management tools but I also wanted to throw in jspm. With close to two million installs this year, jspm is still going strong. Now this isn't going to be a package manager brawl, despite the heading of this section, I'll give you the info and you can decide what it means to you I'm not going to lie though, I use npm and like their team and what they do a ton. So, if I come across as biased, it's probably because I am.

The Digits

Let's first take a look at the comparative installs for the year. There seems to be an almost even exponential growth between each of these package managers. npm still has a large lead over yarn but is less than double the installs of Bower. One of the first things that caught my eye is the obvious pattern of hill-like install stats. Although, jspm looks to be skimming that bottom like, it reached nearly two million installs this year.



https://npm-stat.com/

Npm

45,073,457 installs

It's pretty clear to see that there were many aspects of Yarn that users liked: the speed, the lockfile...

TODO: what other aspects?

Just kidding! Although that really was my note, thank TJ VanToll for recognizing the comedic spin. Jokes aside, Yarn got a ton of attention last year because of its Facebook backing and solutions to npm users sore spots like slow installs and errors caused by package version inconsistencies.

In response, npm released version 5, which was packed with fun things. One of the main focuses of this release was increasing their speed, which, of course, prompted amazing blog post titles like, "npm@5—Yarn killer?" by Nikhil John. With this update npm is noticeably faster.

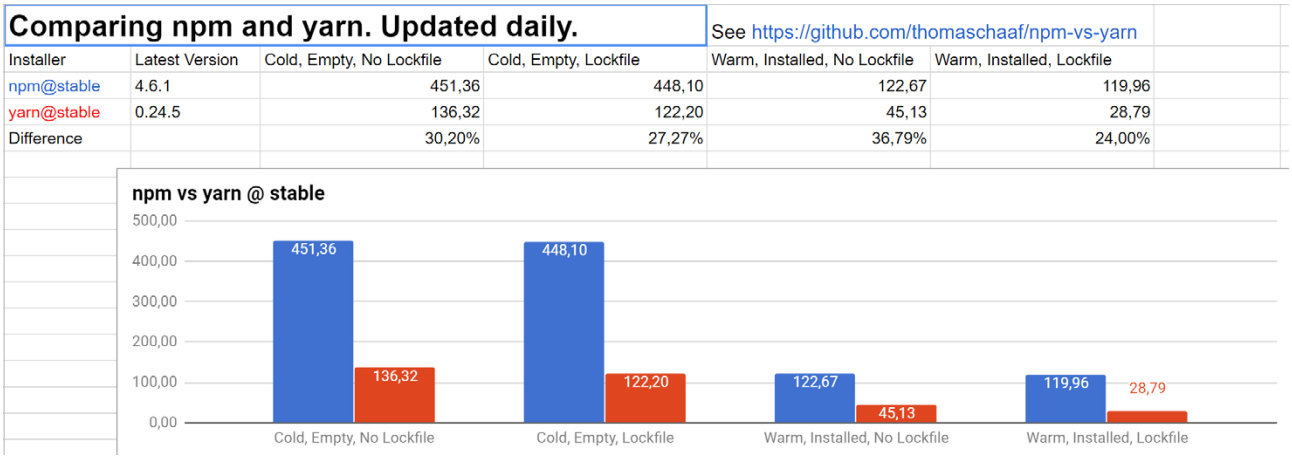
Look at that speed!

This update also included a Package.lock file which has the same benefits of the yarn.lock file, keeping your package versions consistent, and removed npm-shrinkwrap. They brought on a --save default to any package you install, which saves you those, oh-so-important keystrokes. One of my favorite additions is npm's npx package runner. One nifty thing that npx allows you to do is use packages on a per-project basis instead of having to save packages to your machine globally. There is much more to it though, check out the awesome Kat Marchán's post to learn more. There are also more features im general on version 5, you can check out their blog for more information.

Yarn

11,851,948 installs

Even with the updates in npm 5, Yarn is still faster. Oh, you want to see the speed comparison updated on the daily? Well, Thomas Schaaf has just the thing. That's right, here he has a Google doc with daily speed comparison updates.



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	4.6.1	451,36	448,10	122,67	119,96
yarn@stable	0.24.5	136,32	122,20	45,13	28,79
Difference		30,20%	27,27%	36,79%	24,00%

https://github.com/thomaschaaf/npm-vs-yarn

Yarn is on version 1 and stays fast by caching packages and using parallel operations. Caching downloaded packages also means that you have them available whether or not you have a network connection. Yarn also focused on security using checksums (basically, the outcome of an algorithm comparing information you generate and information provided by the package to make sure they match) to verify packages before you execute its code.

There has been some hesitance to adopt Yarn because it is a newer technology but since it's created and backed by Facebook, it makes the choice less risky than most young technologies. Although npm seems to have nearly four times as many installs as yarn, it is important to note that yarn does not recommend installation via npm.

Note: Installation of Yarn via npm is generally not recommended. When installing Yarn with Node-based package managers, the package is not signed, and the only integrity check performed is a basic SHA1 hash, which is a security risk when installing system-wide apps.

For these reasons, it is highly recommended that you install Yarn through the installation method best suited to your operating system.

From the yarn installation guides.

Tune in next year to see what happens for yarn in the year 2018, dun dun duuuuun.

Bower

28,133,666 installs

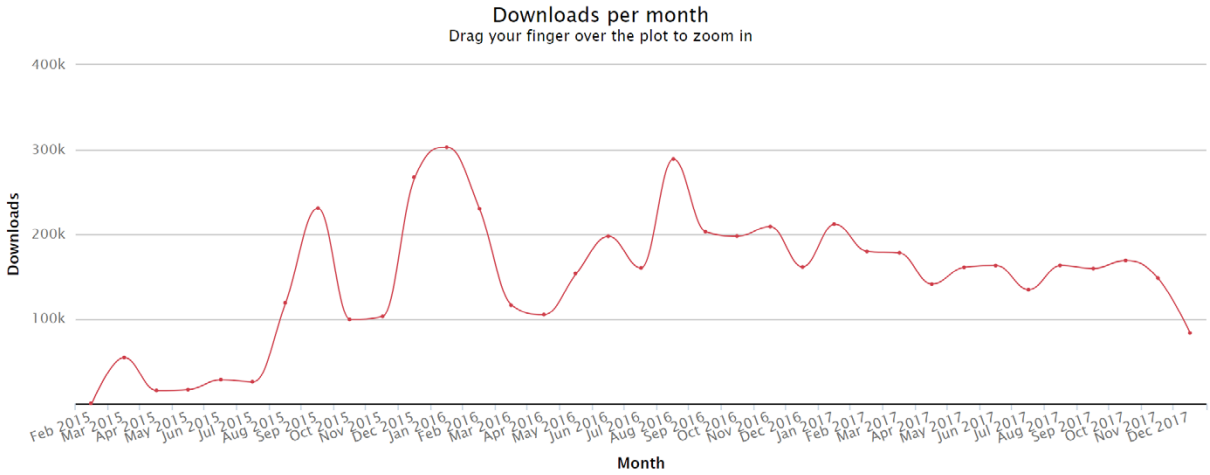
This was a much larger number of installs than I would have ever guessed, coming in second overall and doubling yarn's numbers. Bower is still the most popular front-end specific package manager BUT, while it is still being maintained, the Bower team is recommending users switch to using Yarn and Webpack. In October of this year, Adam Stankiewicz made a post on the Bower blog on how to migrate off of Bower pointing to his repo, bower-away, that he had created in July. Yet, this year's install numbers show Bower with over double the amount of installs of Yarn, so we'll see how that goes. If you feel like cozying up for a long read, check out this closed issue discussing whether or not to deprecate Bower.

One thing they may not be considering is how many users are installing Bower based on a tutorial they're following and are never actually visiting their page. Since, this message to the public is pretty recent we can take look at the numbers next year to see the impact it had.

jspm.io

1,941,913 installs

In their words "jspm is a package manager for the SystemJS universal module loader, built on top of the dynamic ES6 module loader." It can load any module format (ES6, AMD, CommonJS and globals) straight from any registry, like npm and GitHub. jspm does not seem to have much GitHub love in the form of forks and stars but there is consistent activity throughout this year. With nearly two million downloads this year and consistently staying between ~150k and 200k monthly downloads throughout the year, it seems like jspm has staying power.

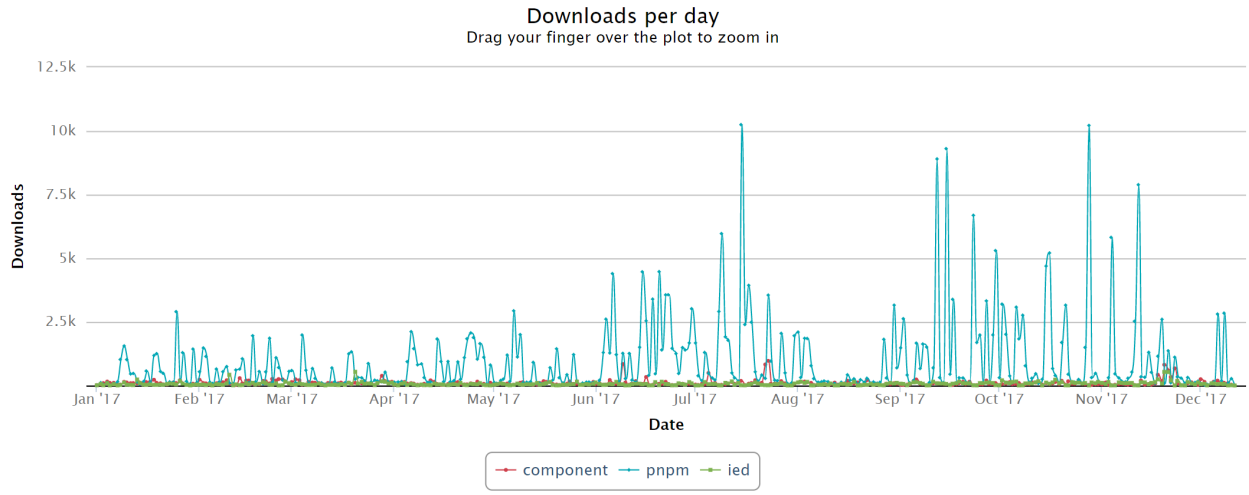


provided by npm stats

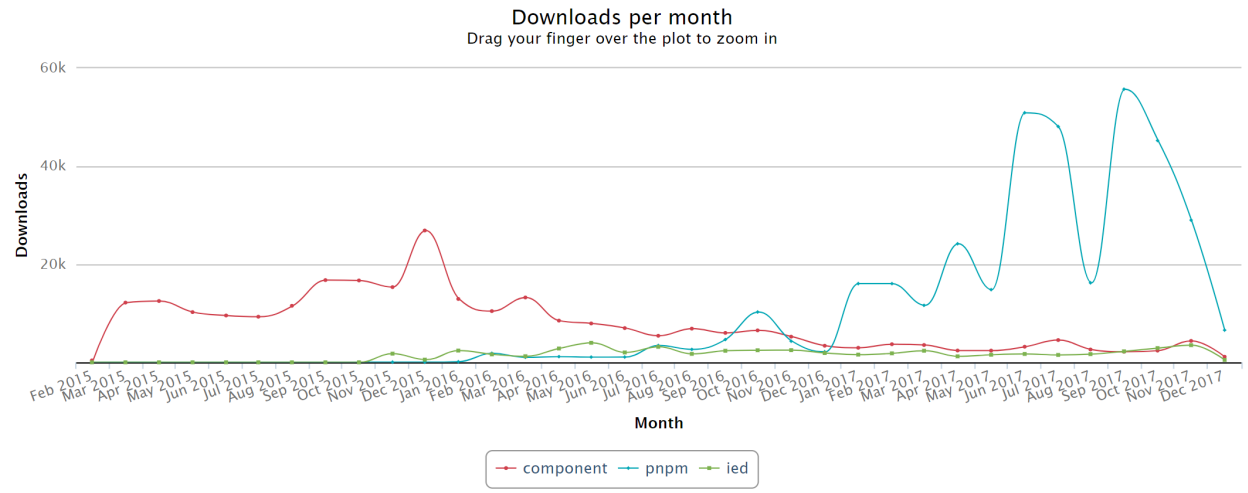
The Other Ones

Okay, chunking these all together may seem harsh but, let's be honest, people aren't using them as much as npm, Bower, yarn or jspm.

Which other ones you may ask? Today we're going to look at three that are currently doing the best in installs this year: component, pnpm and ied. If we take a look at the charts, provided by npm stats (yes, just like yarn, these can all be installed using npm), pnpm is towering over the other two. I also wanted to show a chart looking at monthly downloads starting at February 2015. In this chart it looks as if component and ied have hit their peak and are slowly dying down whereas pnpm is on an upward trajectory. Let's briefly dig into each project.



provided by npm stats



provided by npm stats

pnpm - 334,497 installs: By far the most installs of these "others" package management libraries and is the youngest of the bunch having its first commit in January of 2016. It focuses on speed leveraging disk space efficiency and is actively being worked on. It currently seems to be worked on actively, having a commit every few days or so.

component - 35,340 installs: This project is deprecated and hasn't had a commit in 2 years, yet still has over 35,000 installs this year.

ied - 22,522 installs: Touts being "like npm, but faster" and had its first commit back in August of 2015. It is specifically for Node, has some killer ASCII art but hasn't had a commit in over a year.

Only the future can really say what will happen to these brave "other" libraries. Although, it's probably safe to say that component and ied may eventually fade away never to make it into the top package manager section. It is the open-source world though, so never say never.

So the package manager battle wages on but when it comes down to it, we have options for really great package management tools. Isn't that the way it should be? You tell me. I'm just happy to have a great way to install all the the things I need to build all the weird app ideas I have in my head!

Related: Look at this great list of package managers.

Conclusion

In 2017, ECMAScript continued its small but impactful deliveries, the package manager race continues to make our experiences better, we have some great tools to make JavaScripting a tidbit easier and we have more ways to utilize the advancements of the modern web. 2017 was pretty crazy but look at all these bright spots we have in our JavaScript world. That's right, I'm an optimist! There are bound to be many more things to talk about in a year's time but, for now, let's be thankful JavaScript has survived another year without burning everything to the ground.

Libraries and Frameworks and Futures, Oh Yes!

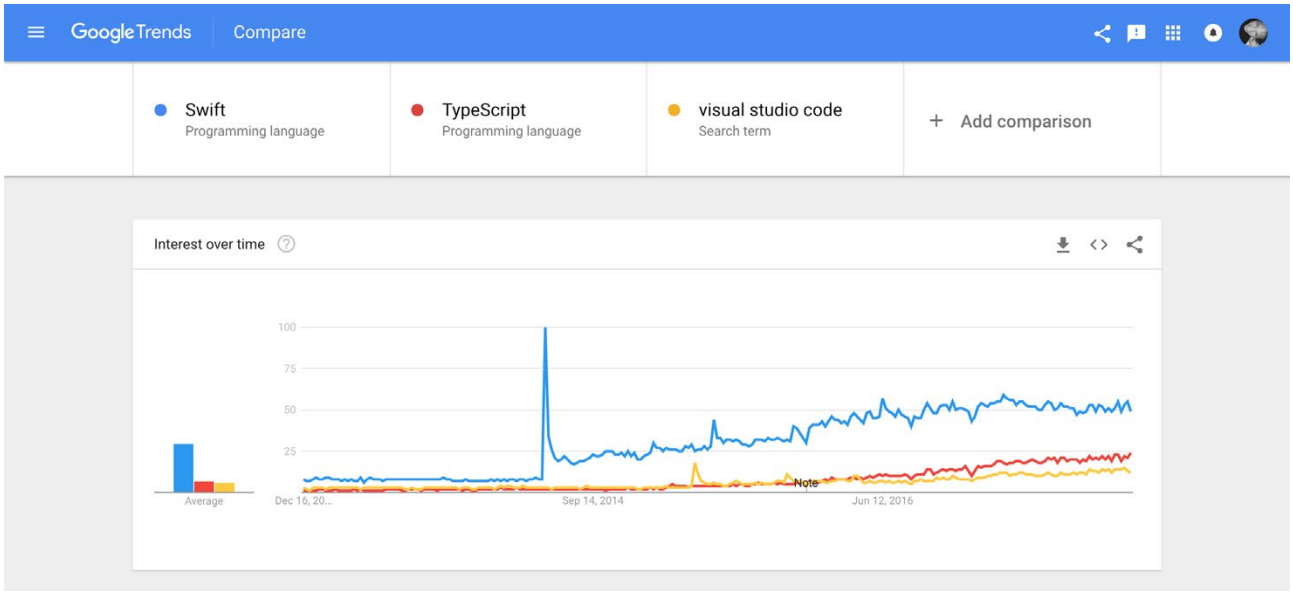
by @AlyssaNicoll

Here at Progress we take a yearly look at the JavaScript framework landscape and try to guess where things are going. Last year, we focused on frameworks such as React, Angular, Ember, Vue and Backbone. This year, we are going to take at look at all the same frameworks, save Backbone. If there is anyone less hot than Ember, that would be backbone. Here are the frameworks and libraries we'll be taking a look at.

We'll also look at Kendo UI. We certainly aren't biased because we make it and hope it does well, certainly not. Looking back can help us determine how each of these frameworks impacted web development in 2017, as well as where they are likely headed.

Open Source Predictions For 2018

We all know that Google is the driving source behind Angular and Facebook is the creator and maintainer of React. In 2017 we predicted that open source software, controlled by large corporations, aka “Corporate Open Source”, would become more prevalent. While no new huge contenders have entered the ring in 2017, OS projects that started in 2016 or earlier remain strong (like .NET and UNIX). Whereas others have really taken root and spread, like VS Code, TypeScript, and Swift.



source: https://trends.google.com/trends/explore?date=today%205-y&q=%2Fm%2F010sd4y3,%2Fm%2F0n50hxv,visual%20studio%20code

As you can see in this Google Trends chart over the past 5 years, all three of those OS projects are on the incline in 2017.

I made a nifty list of OS goodies and the earliest dates I could find on their Github repos:

Microsoft

- vscode - OS in July 2016
- dotnet - OS in Sep 2016
- TypeScript - OS in March 2015
- azure-sdk-for-node - July 2015

Apple

- Swift - Dec 2015
- UNIX

We had hoped that there would be more open-source offerings for JavaScript developers from Microsoft and Apple. However, in October of 2017, Apple did open source the kernel that drives iOS and macOS.

"[Apple's iOS and macOS kernels] are now available on GitHub, representing the first time that Apple has released such integral code into the public domain."
— *The Inquirer*

Angular

For this girl's opinion, Angular is still pretty hot *cough cough* *points to Tesla's hiring page*. Many large companies made the switch from AngularJS to Angular. It would take something just shy of a miracle to switch them off this robust framework.

In November, the latest version, Angular 5.1, was released. Stephen Fluin Angular's Developer Advocate, wrote about all the juicy deets in this blog post.

I pinged Stephen on slack and asked what he thought Angular's biggest feat was in 2017. Here is what he had to say:

"In 2017 we've successfully balanced stability and innovation. making your applications smaller and faster without making you rewrite your code.

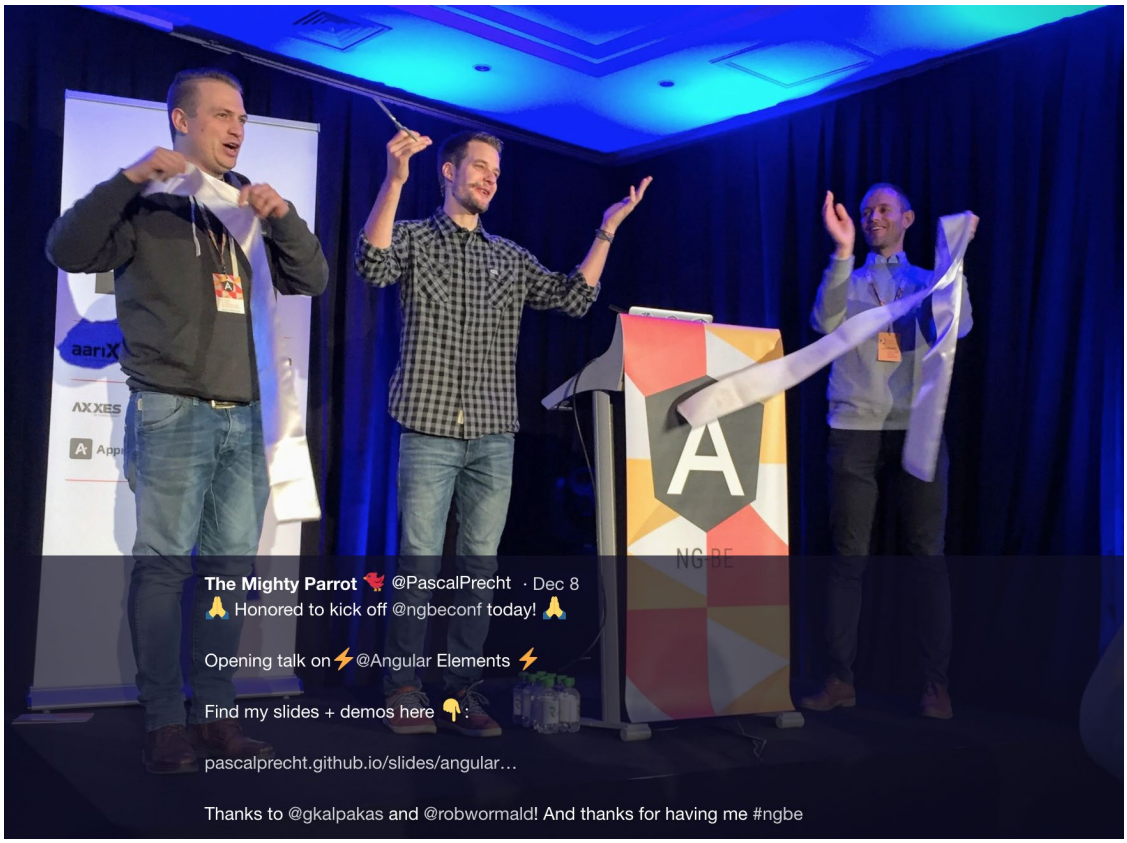
It can go much further, like imagine in 3 years Web Assembly is good enough that we want to use it. Because we are a full platform, we could start shipping part of your apps as Web Assembly for you, without you having to do anything. Or ES2015 modules, or web components, or any of the 'modern web'."

I'm extremely pumped to see what Angular has in store for 2018, 19, and beyond. It feels like we are finally getting past those awkward teenage years, where we are still figuring out who we are and how we fit into the world. Now we know what kind of framework Angular is and needs to be and we are well on our way.

As Rob Wormald put it:

"Angular is ideal for building complete applications and our tooling, documentation and infrastructure have been primarily aimed at this use case..."

For one thing, I think Angular elements are going to be HUGE. Rob gave a talk on them in November at Angular Connect and Pascal Precht just gave a keynote on them @ngbeconf.



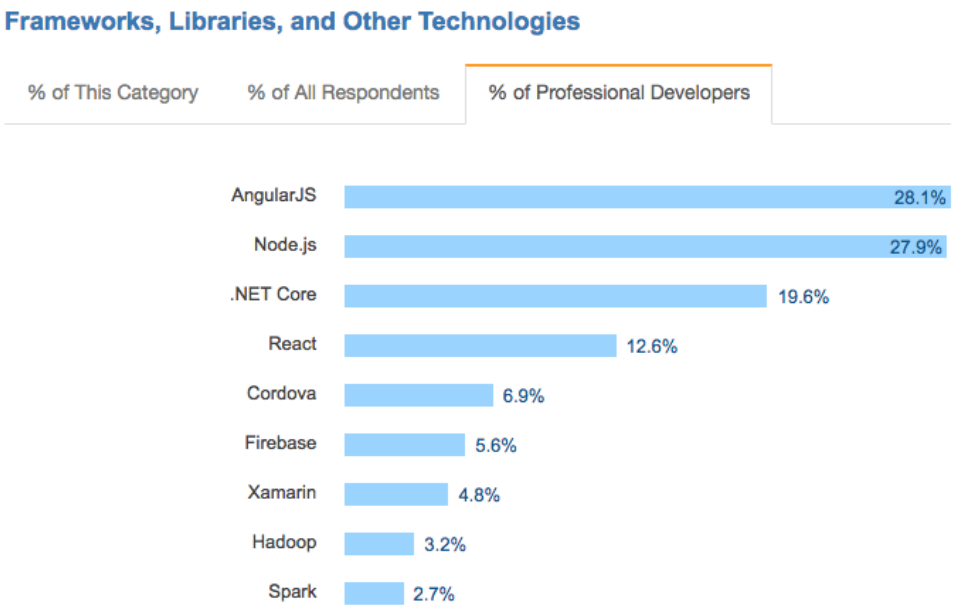
Source: https://twitter.com/PascalPrecht/status/939102123102474240

Angular elements are simply **Angular Components wrapped in Custom Elements**. These will bridge the gap and allow you to use Angular Components anywhere, without the full Angular environment. Need an Angular Component included in your React project? No problem, Angular Elements have you covered!

Not only is the Angular team looking to solve these issues in 2018, but we also have so much to look forward to in the coming years. They hope to have a solution for everything you could possibly need in a typical Angular development environment. The end of 2018 should not look too strange from it's beginning for Angular. Since Sep. 2016 they have started a release cycle that allows time for breaking changes to be deprecated, before they are officially changed. This friendlier system means developers have time now to work on updating (approximately 6 months, sometimes longer) before breaking changes become official. We foresee some cool things from Angular Elements (expect more on that at this years ng-conf), as well as some updates to the CLI and Angular Material. The Angular team is also in cahoots with the new in-browser editor team 'StackBlitz'. We predict that all of the Angular Material and Angular doc examples will be switched from Plunkr to StackBlitz in 2017.

AngularJS

Stack Overflow Dev Survey 2017

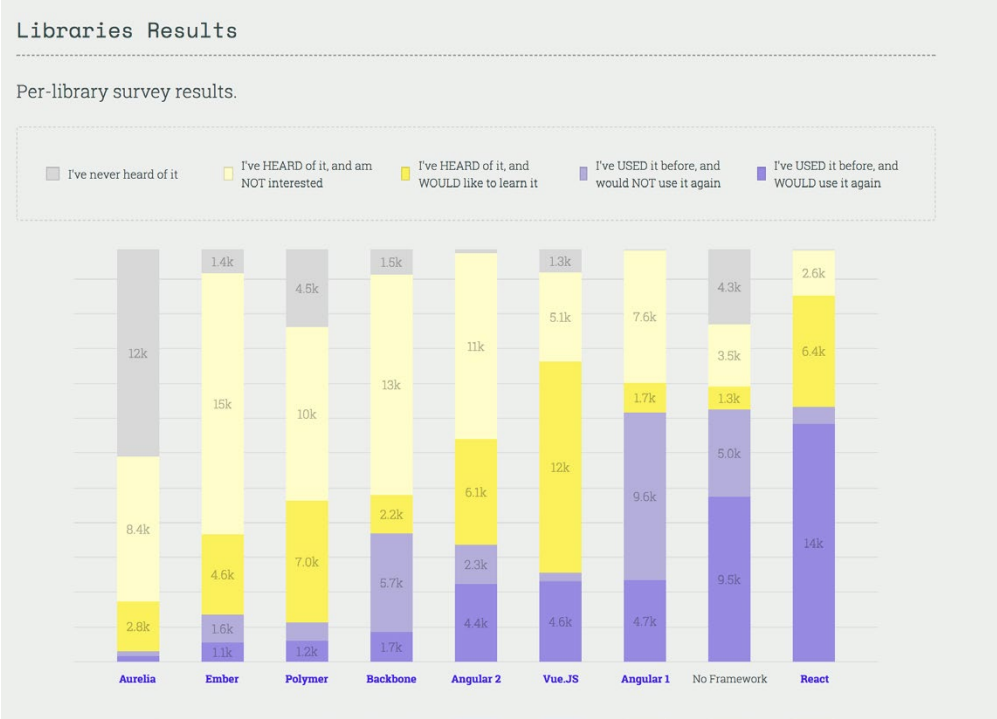


27,612 responses; select all that apply. Shown as a percentage of the respondents who chose at least one language, framework, database, or platform.

"Node.js and AngularJS continue to be the most commonly used technologies in this category."

Source: https://insights.stackoverflow.com/survey/2017#technology-frameworks-libraries-and-other-technologies

Looking at Stack Overflow's 2017 survey would also make you think AngularJS is still crushing react. However, in the survey results from "The state of JS in 2017", we see a different story. In this chart, React is clearing conquering all in the `I've used it before and would use it again` category.



The current state of AngularJS

At the last Google Developer Expert Summit that I attended, they again expressed their plans to continue updating AngularJS, only so long as the traffic to its docs outnumbered the traffic to the Angular.io docs. AngularJS's days are numbered.

That being said, there are still so many companies using AngularJS, without any plans of upgrading. Why, you might ask? Some just do not have the resources needed to dedicate dev hours towards upgrading all the directives to components and then make the massive swap over to Angular. Let's be real peeps, this is no simple upgrade. For others, they are hesitant (still) to adopt TypeScript.

At first, the Angular team said they would support multiple languages, not just Typescript. However, a few months into Angular being released, they swiftly back tracked and removed any references to other languages in their docs. So for some companies, believe it or not, they won't upgrade because they cannot give up their beloved CoffeeScript. ;)

So where does that leave you? If Angular is too large of a change for your company to swallow, and AngularJS is bound to be deprecated one of these days, where does that leave your company, clients and code? Some companies are still biding their time, while they don't have to make a decision just yet, while others are choosing the jump to VueJS.

Vue.js

VueJS has been on the rise and mentioned as the next hot new thing, even Nasa is hiring VueJS devs! It is VERY similar to AngularJS, and does not require a compiled language change like TypeScript. (Yay, the people can still have their CoffeeScript and the dev world shudders).

What is Vue?

You guessed it! Vue is yet another WONDERFUL JavaScript framework. Vue is simple to get started, scales to large cases easily, has everything you need end to end to build small to large scale apps. It boasts these fun features, which after reading, you should think to yourself... AngularJS?

AngularJS •cough• I mean, Vue.js Features

- Reactive Interfaces
- Declarative Rendering
- Data Binding
- Directives
- Template Logic
- Components
- Event Handling
- Computed Properties
- CSS Transitions and Animations
- Filters

No surprise there though, Vue was created by an ex-Googler, Evan You, who after using AngularJS on a number of projects, was just playing around, seeing if he could strip down AngularJS to the core parts he liked. Thus, in 2013, VueJS was born.

"For me, Angular offered something cool which is data binding and a data driven way of dealing with a DOM, so you don't have to touch the DOM yourself. It also brought in all these extra concepts that forced you to structure the code the way it wanted you to. It just felt too heavy for the use case that I had at that time.

I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight without all the extra concepts involved? I was also curious as to how its internal implementation worked. I started this experiment just trying to replicate this minimal feature set, like declarative data binding. That was basically how Vue started."

Evan You for "Between the Wires"

You can read more about Vue's origin story here. The Vue.js 2 core library is very small (17 kB). This means that using Vue in your site should be pretty fast and load in browser quickly. This ALSO means, that learning Vue should be relatively easy and it shouldn't take you long to get started with the framework.

"I read thru it's docs and knew everything i needed to know in less than 30 min. I couldn't finish the first page of Angular in 30min." — Anonymous Dev

React

I'm still surprised by all the apps I continually run into and after inspecting under the hood, I realise they are created with React. React, while not being the newest of "hot-nesses", is still very prevalent and often brought up as a web dev go-to framework. Don't believe me? Check out these killer circle charts from "The State of JS 2017 survey" results. React's big.

source: https://stateofjs.com/2017/front-end/worldwide/

What is React? Should I use it?

"React is very popular and will likely continue to be now that Facebook has resolved the licensing controversy. It has become the first choice alternative to Angular and only continues to gain traction." –Joe Eames, JavaScript and Angular expert source

React, unlike Angular, is not a framework, but rather a library. React is a JS library for building UI on web apps. It provides a declarative method of defining UI components, which as they claim:

"Declarative views make your code more predictable and easier to debug." — React site

They also enforce component based architecture. An encapsulated component in react should manage its own state and multiple components can be combined in your quest to build your apps UI. Below is a handy chart that compares some features of React vs. Angular.

Attribute	Angular 2	React
Churn	Reduced	High
Tooling	High	High
Code Design	JS into HTML	JavaScript Centric
JavaScript "Fatigue"	Less	More
DOM	Regular DOM	Virtual DOM
Learning Curve	Medium	Low
Packaging	Medium	Strong
Abstraction	Strong	Strong
Debugging General	Good JS/Good HTML	Good JS / Bad HTML
Debug Line NO	No	Yes
Unclosed Tag Mentioned?	No	Yes
Fails When?	Runtime	Compile-Time
Binding	2 way	Uni-Directional
Templating	In TypeScript Files	In JSX Files
Component Model	Strong	Medium
MVC	Yes	View Layer Only
Rendering	Server Side	Server Side

source: https://blog.techmagic.co/angular-2-vs-react-what-to-chose-in-2017/

React updates in 2017

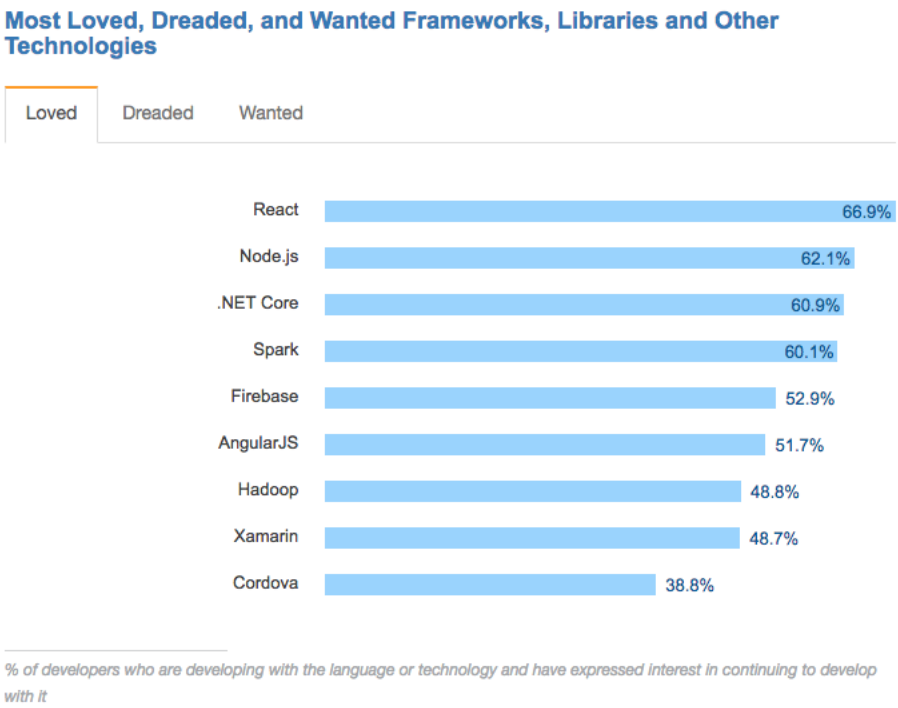
In September the React team announced the release of React v16.0! Some long requested features/changes made it into this release, including improved server-side rendering, error boundaries, support for custom DOM attributes and fragments.

With the new return types (fragments and strings) you can now return an array of elements from a component's render method. Like with other arrays, you'll need to add a key to each element to avoid the key warning.

React did really well on not only the stackoverflow dev survey for 2017 but also The State of JS survey and npm trends for the past year. These aren't fully comprehensive representations of the web dev world as a whole, but they are an interesting window into it. We predict React continue to grow in popularity in 2018.

Stack Overflow's Dev Survey 2017

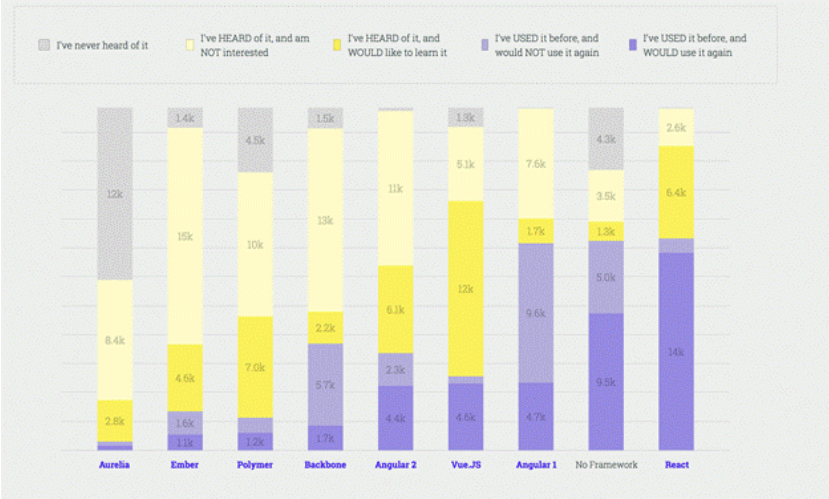
On the Stack Overflow survey, React scored highest for `Most Loved` among developers.



"React is the most loved among developers, whereas Cordova is the most dreaded. However, Node.js is the most wanted."

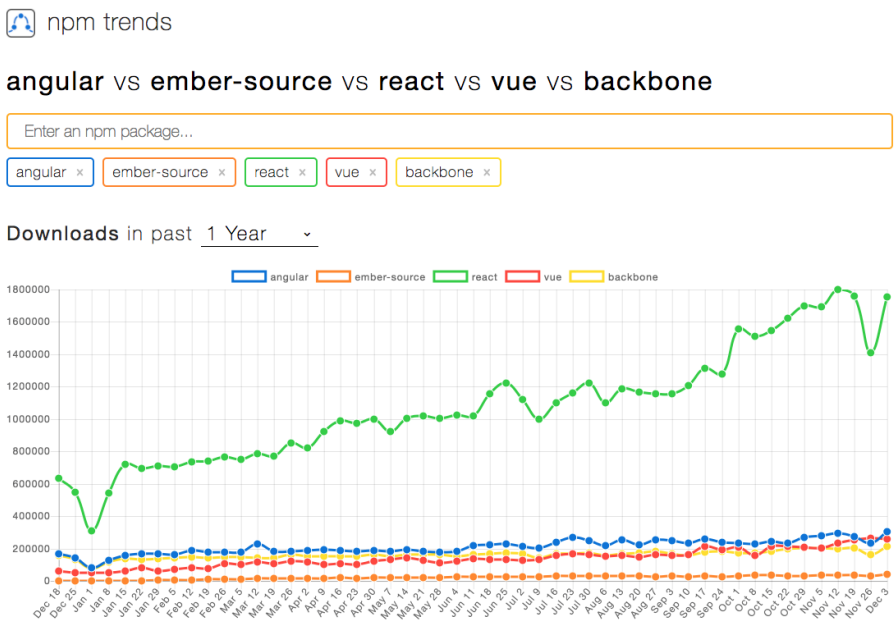
The State of JS 2017 Survey

React did the best out of ALL the frameworks on `The State of JS 2017` survey by Sacha Greif. 14k people said they have used React before and would use it again. The next leading framework in that category was Vue.JS, which only got a measly 4.6k votes.



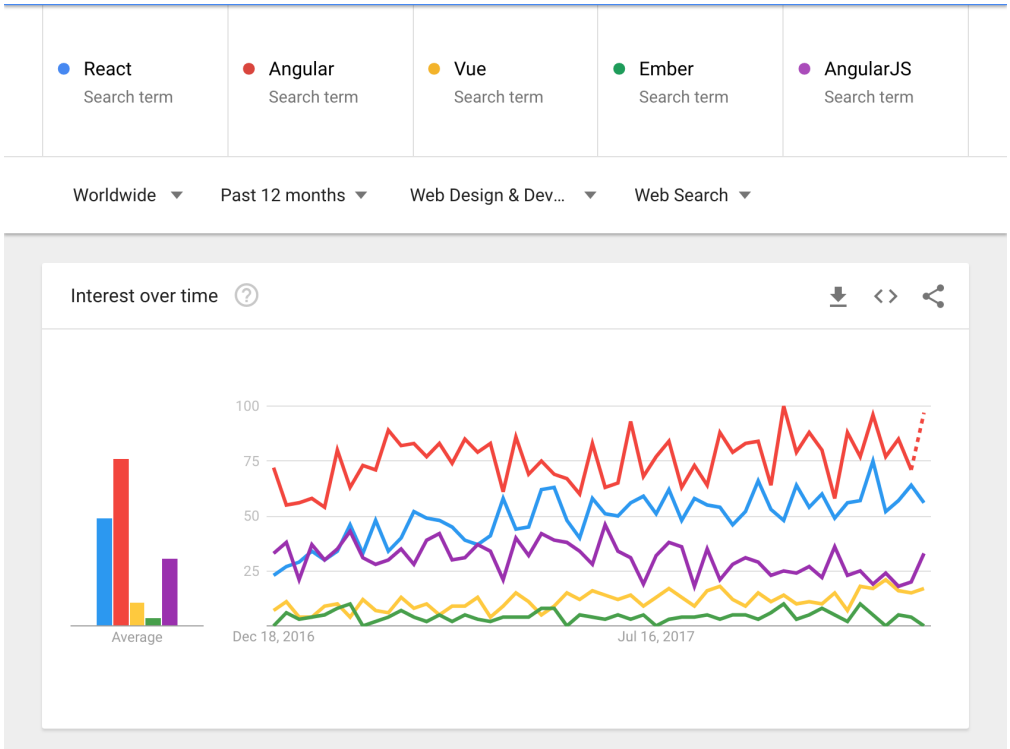
Source: https://stateofjs.com/2017/front-end/results

On npm trends, React is the most downloaded module, when compared Ember, Angular, React, and Vue, and Backbone.



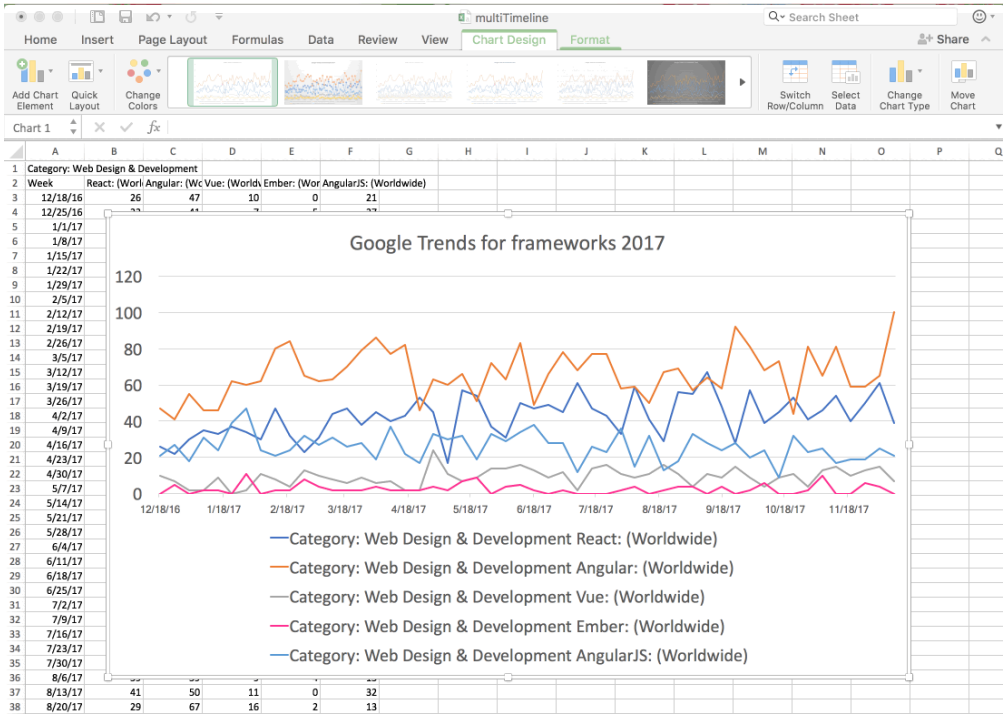
Ember

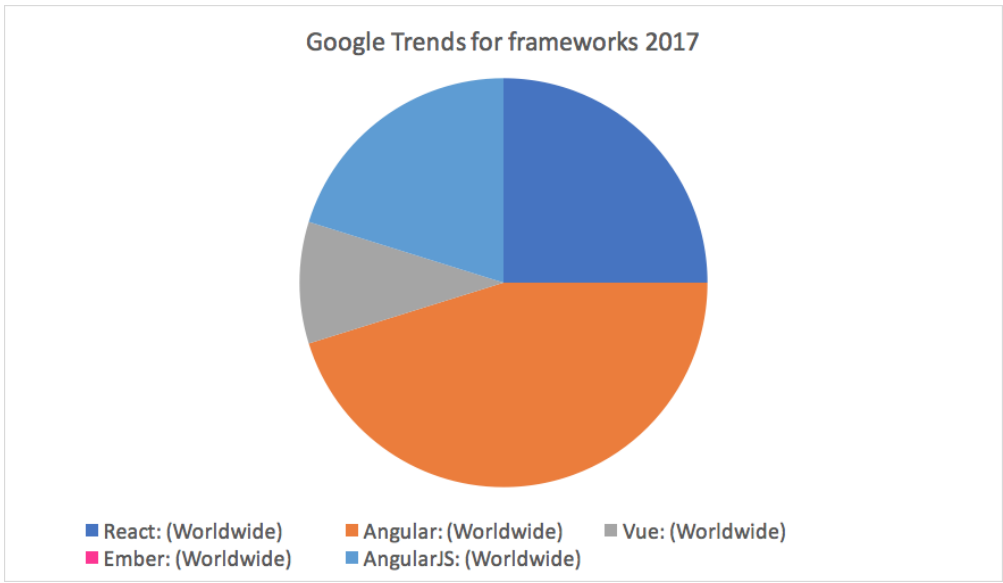
Ember. What to say about Ember? I used Google Trends to generate some data on the four top runner Frameworks and got this glorious chart below. Ember is that little green line that seems to be flatlining there at the bottom.



source

Now that chart doesn't do a whole lot for me, other than showing Ember as a dead thing, which we all knew. Jk, please don't send Tomster after me. So I decided to turn it into a nifty pie chart (exported said data into excel), that might help shed more light on the situation:





There really is no one way to know which frameworks are doing the best. This is just one peek into the enigma that is the web dev world. However, just going off of this pie chart, it still looks like Angular is on top, followed by React. Whereas Ember doesn't even get a piece, its dataset is THAT small. Other surveys (see earlier) don't show Ember doing too well either. This does not mean that developers are done using Ember, it just means that the survey-taking-type devs are not in love with Ember right now. That's the beauty of surveys, they only shed light on the part of the demographic willing to take them.

For example, the site builtwith.com measures how many sites across the web are built with specific technologies such as JavaScript frameworks. And for Ember specifically you can

Get a list of 31,099 websites using Ember which includes location information, hosting data, contact details, 15,116 currently live websites and 15,983 sites that used this technology previously.

31k sites, not too shabby, even if the framework is trending downward.

"One of the problems with Ember is that it targets a different type of developer, different from any other target demographic. It targets people who like Python, Rails or CoffeeScript. Ember is a different way of writing code. It's very structured, there's a way to do everything, everything is standardized — all things that are abnormal to JS devs. People like the freedom of JS, with Ember, you cannot break the rules." — Zach Nicoll, Front End Web Dev

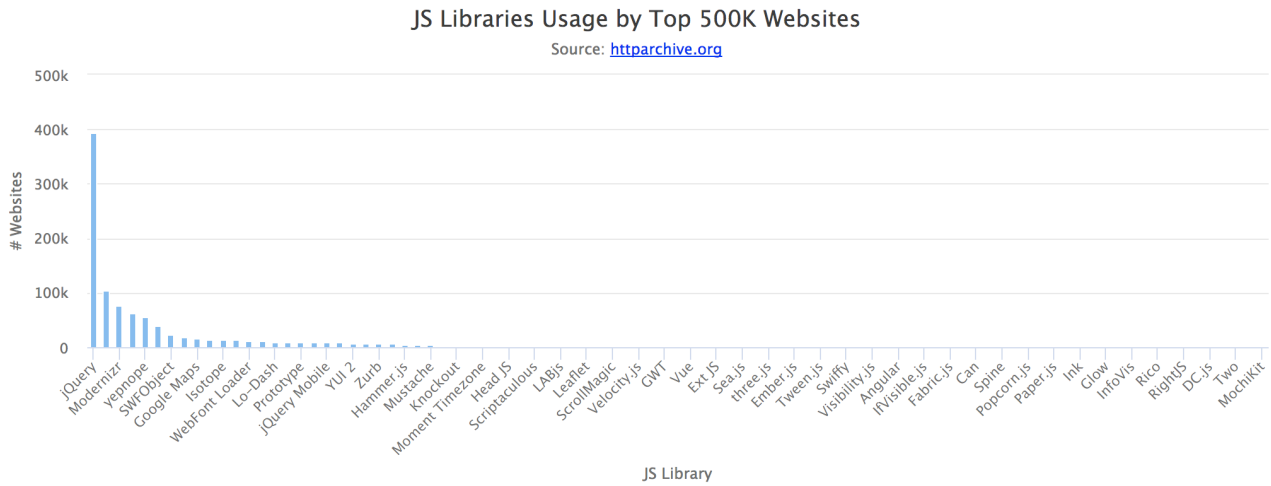
So after interviewing him on his love of Ember, I understand a little more why it might be trending downward (at least on the scales we have to measure it today). I predict that Ember, like Angular, isn't going anywhere. Whether or not it's popular in 2018, I can't say, but it will still be one of the forerunner frameworks.

Much of the Web Still Runs on jQuery

In March this year, 3.2.1 was released with bug fixes like this fella:

Ensure we get proper values for width and height on elements with display "inline".

The Internet keeps chugging along and so does jQuery. Like 90% of the Internet runs on jQuery. We all know this, so the chart below shouldn't shock you.



Source

This CSS Tricks article goes over new vanilla alternatives to jQuery methods. As JavaScript gets better and advances, jQuery should inevitably become deprecated.

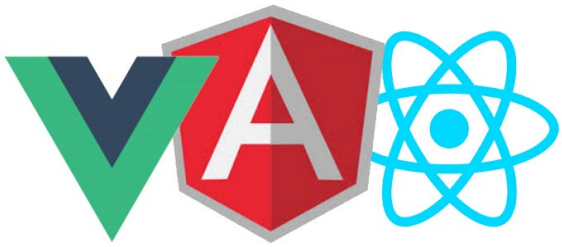
> jQuery:

```jQuery
var $elem = $(".someClass") //select the element
$elem.remove(); //remove the element
```

> Without jQuery:

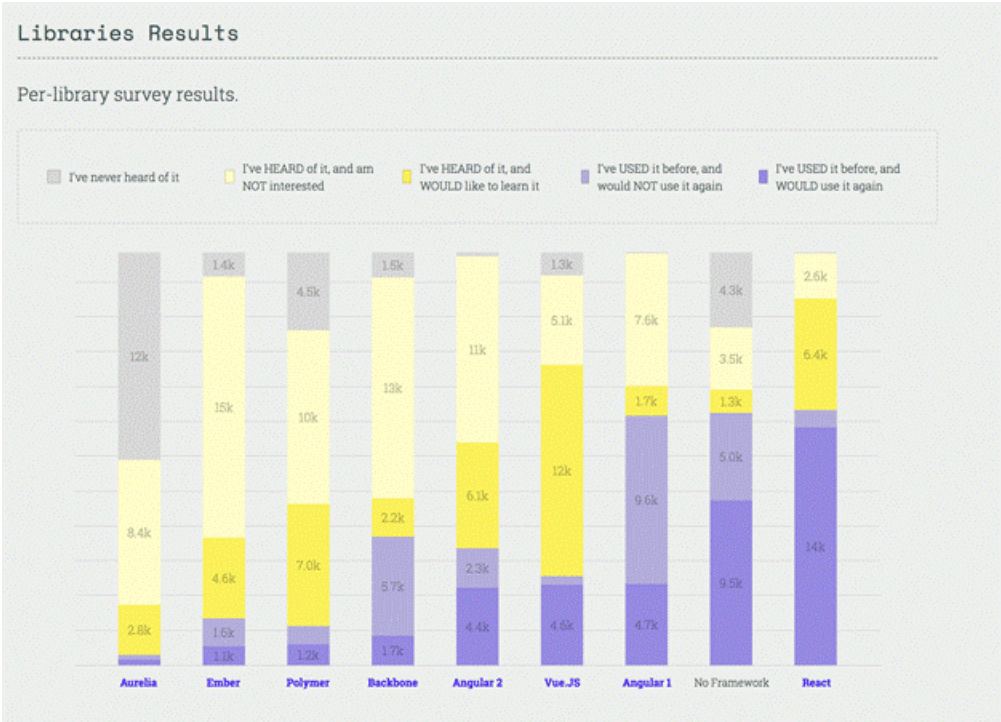
```JS
var elem = document.querySelector(".someClass"); //select the element
elem.remove() //remove the element
```

However, as a friend of mine likes to say "you might wanna take that with a bucket of salt", especially since jQuery has been around since the dawn of time. It would be very hard to imagine anything over throwing that dynasty, anytime soon.

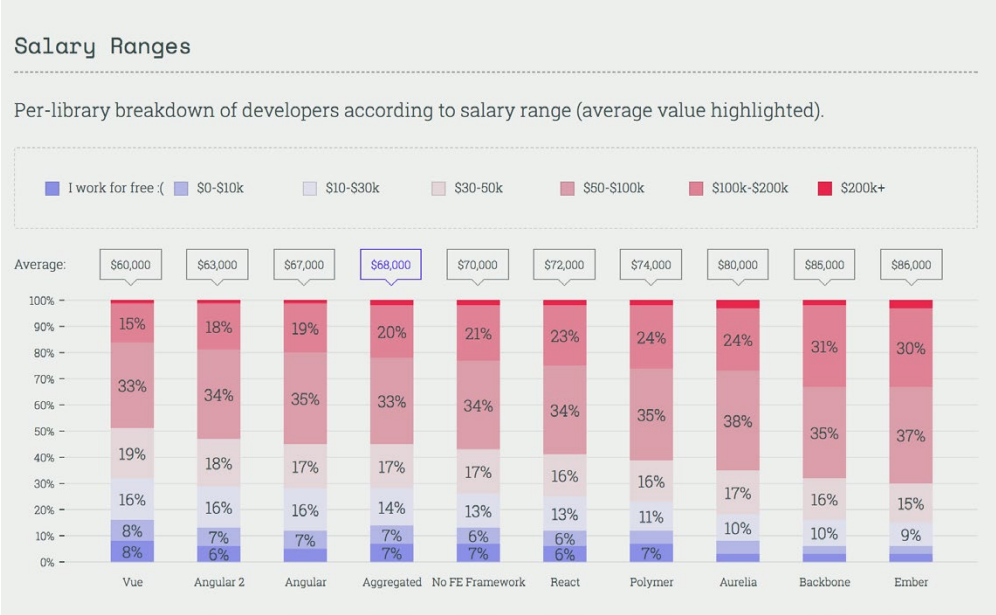


The State of JS in 2017 Survey Results

Now that we have taken a look at the big JS frameworks out there, let's shift our attention to some recent survey data so we can pontificate our futures and the meaning of all this. Earlier this month, Sacha Greif released his survey results, in which he surveyed thousands of developers about their framework preference, salary, and more. I think I've found an interesting correlation on the recent survey results released.



In the above chart, backbone and Ember are pretty much on the dead side, whereas React is RIDICULOUSLY popular. Now take a look at this chart, telling about salary breakdowns according to framework.



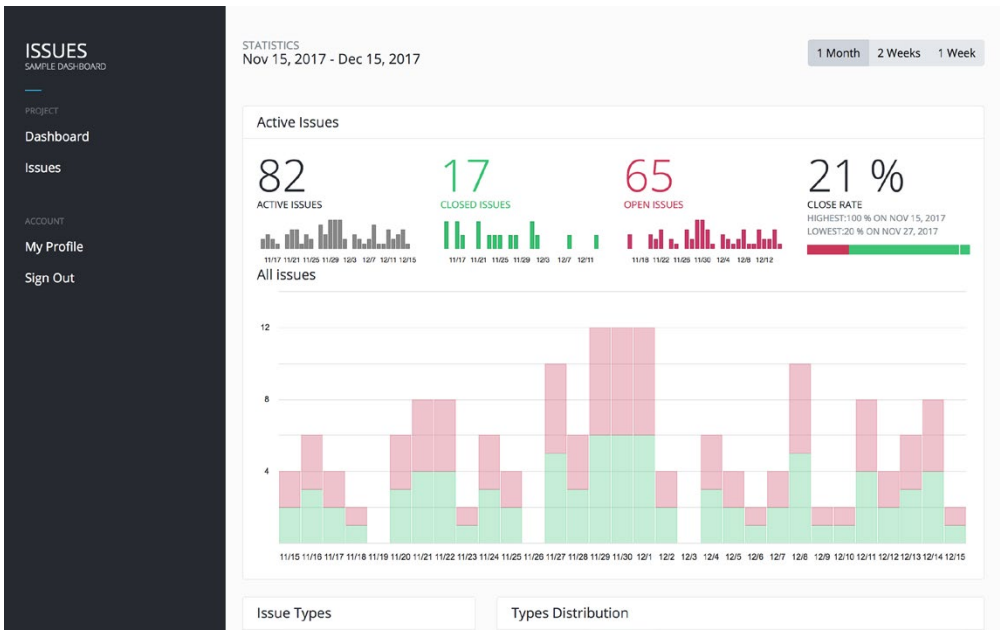
I find it really interesting that Ember and backbone, two on the lowest on the popular chart, are two of the higher paying salaries. Whereas the more popular languages (Vue, Angular, React) are on the lower end. It seems to me that the old "supply and demand" theory is at it again!

Kendo UI

2017 was huge for Kendo UI. Looking ahead, 2018 is shaping up to be even bigger. Here's a quick look into the crystal ball!

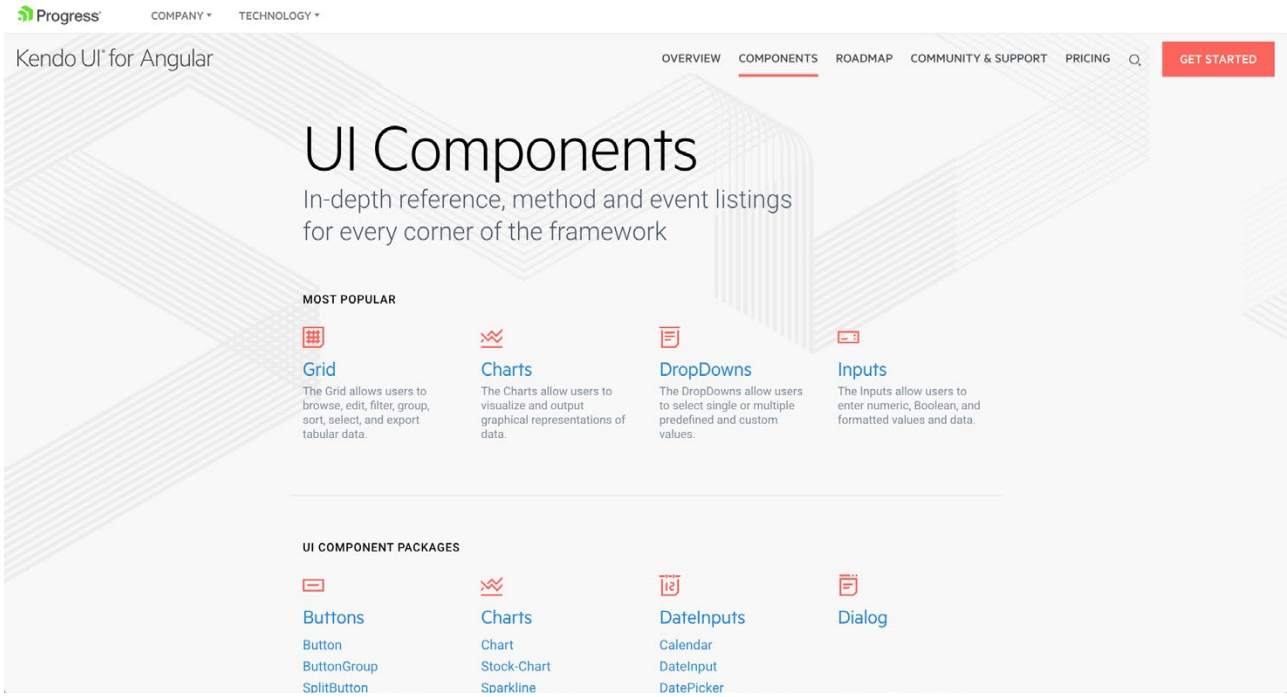
Wrappers for React and Vue were introduced in September of 2017. These wrappers give React and VueJS devs access to the majority of our components. In 2018, ALL jQuery-based components in Kendo UI will have wrappers for React and VueJS. We're SUPA pumped!! This means that Kendo UI will fully support the big four (jQuery, Angular, React, and Vue).

React's continuing popularity bodes well for Kendo UI in 2018. We just released, this January, a set of native components for React. In Kendo UI, you have one of two options. Wrappers, that provide our components or truly native components that are written in the framework you are using. (As mentioned above, we currently support jQuery, Angular, React, and Vue.) These React native components will make use of features such as the virtual DOM, useful for complex components like the Grid. They include form component suites, with such goodies as DropDowns and powerful (and did we mention sexy?) input elements. They will also support themes for Material Design and Bootstrap v4.



Checkout Kendo UI's dashboard component for React!

We're also upping the Angular support in Kendo UI to include popular components such as the TreeView, Window, Splitter, and Gauges. Many features for our very popular Grid component will also be added.



Check Out The Angular Components Kendo UI Already Offers!

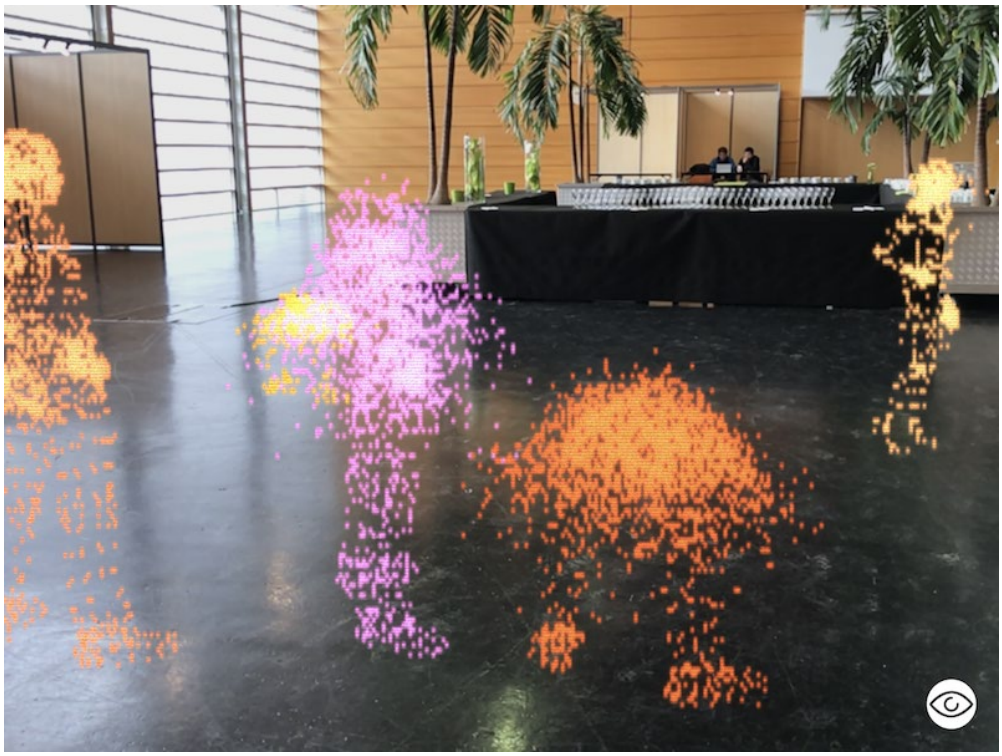
Prediction Time

On the safer side, we predict that the weak JavaScript frameworks will get weaker, and the strong will get stronger. People (and companies) will continue to use frameworks they know and love. Angular, React, Ember, and Vue will all still be in the game come 2019. I predict that Vue will continue to light up and be used, but at an individual scale, rather than a large company scale.

AR/VR

This month Mozilla announced the WebXR Viewer app was released for download on the iTunes store. This app is an augmented reality viewer that lets you make and view AR experiences created with webxr-polyfill Javascript library and ARKit.

This app is not intended to be a full-fledged web browser, but rather a way to test, demonstrate and share AR experiments created with web technology.

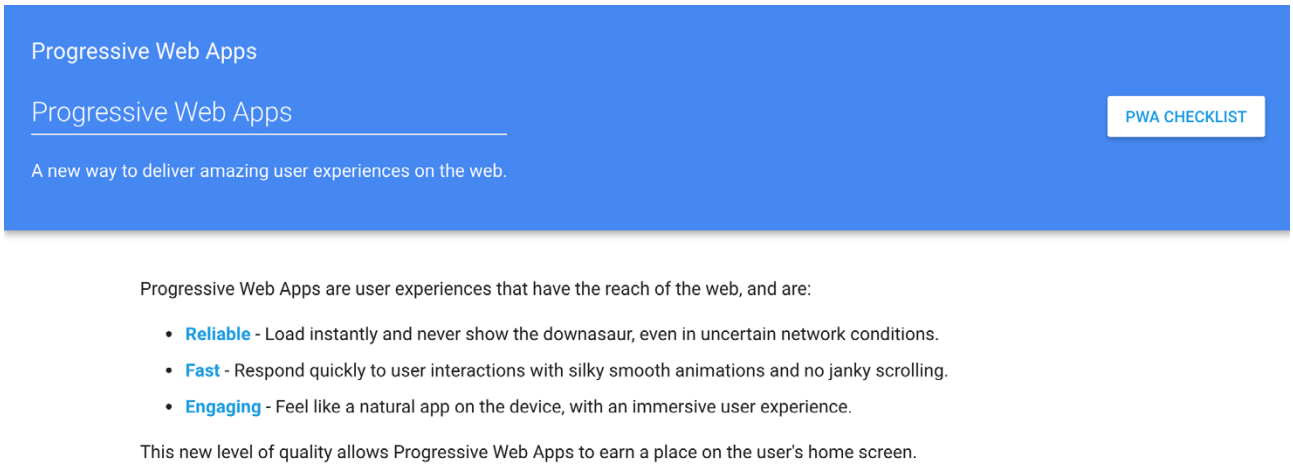


Mozilla isn't the only company in the AR/VR pie, however. Google also released their own AR app called WebARonARCore APK on Android. Mozilla has also been working on integrating into three.js graphics library and A-Frame framework. These are both really popular libraries/frameworks on the AR scene and it will only make it easier and easier to use AR/VR if big companies like Mozilla and Google support it.

On our more ballsy-prediction-scale, we predict that libraries and frameworks will start diving into the AR/VR scene more thoroughly and that new AR/VR libraries and integrations with existing libraries will start to pop up. We hope that 2018 will have some fun and interesting reveals.

PWA

Progressive web apps are where it's at right now. Building your app, no matter the frameworks or libraries used, so that it can work offline is super in right now. But there are more to progressive web apps than just "working offline". Here is the Google standard for what it takes to be a PWA.



Check out Google's PWA checklist today!

Gartner predicts that *"by 2020, progressive web apps will have replaced 50% of general-purpose, consumer-facing mobile apps"*. I don't know about 50%, but I do predict more and more big name companies implementing progressive web app features on their sites in 2018 and beyond.

TJ, our friend from the NativeScript team, actually just wrote a blog post about them. [insert link] In his post, TJ outlines the benefits to switching your app to a PWA, why PWAs have been so successful, and compares PWAs versus JavaScript-driven native approaches. Check it out for more nitty gritty on PWAs.

Here's to 2017

We here on the Kendo UI team understand that all of these surveys and tracking trends are not the full picture. If we have misrepresented or even forgotten your favorite framework ping me or my colleagues on the twitters! We'd love to hear your thoughts and promise to keep an open mind, now, in 2018, and forever more! We hope you've enjoyed our summary of 2017 and wish you all the best of luck in your coding endeavours in 2018!

Happy Coding <3 from the Kendo UI team authors

😆Tara Z. Manicsic🐺

@Tzmanics

developer, electronics & clay & textile tinkerer, all-around good time...er

Alyssa Nicoll

@AlyssaNicoll

Google Developer Expert • Intl Speaker • Egghead.io Teacher Panelist weekly on @AngularAir and @AngularPodcast • FWC Titan ;)

TJ VanToll

@tjvantoll

Web Developer · Tech Author · Principal Developer Advocate at @ProgressSW · Work on @NativeScript

About Kendo UI – Our Complete JavaScript UI Component Library

Kendo UI allows you to quickly build eye-catching, high-quality, high-performance responsive web-based apps integrated into your technology of choice (jQuery, Angular, React, or Vue). Kendo UI offers a large library of popular components from sophisticated grids, charts and scheduler to buttons and menus. Access the library's 70+ customizable UI components to speed up your development time by up to 50%.

Get started with a free trial

About Progress

Progress (NASDAQ: PRGS) offers the leading platform for developing and deploying mission-critical business applications. Progress empowers enterprises and ISVs to build and deliver cognitive-first applications that harness big data to derive business insights and competitive advantage. Progress offers leading technologies for easily building powerful user interfaces across any type of device, a reliable, scalable and secure backend platform to deploy modern applications, leading data connectivity to all sources, and award-winning predictive analytics that brings the power of machine learning to any organization. Over 1,700 independent software vendors, 80,000 enterprise customers, and 2 million developers rely on Progress to power their applications. Learn about Progress at www.progress.com or +1-800-477-6473.

Worldwide Headquarters

Progress, 14 Oak Park, Bedford, MA 01730 USA
Tel: +1 781 280-4000 Fax: +1 781 280-4095
On the Web at: www.progress.com
Find us on facebook.com/progresssw twitter.com/progresssw youtube.com/progresssw
For regional international office locations and contact information,
please go to www.progress.com/worldwide

Rev 2018/02 | RITM0010943

Progress