Progress®

Choosing the Right JavaScript Framework for Your Next Web Application

by Brandon Satrom

WHITEPAPER

Table of Contents

The state of web application development has changed drastically in the past ten years. These changes have been spurred on by greater bandwidth for many, more powerful and feature-rich browsers. The result is a thriving web ecosystem with sites, apps and experiences that would have been hard to imagine a decade ago.

And yet, for many developers, this ecosystem often feels like a bazaar, stuffed with libraries and frameworks that do anything and everything one could imagine. For every need there are a dozen options to choose from, and with new stalls (libraries) opening every day, it's no wonder many of us pine for the days when starting a new project was as simple as dropping jQuery in a script tag.

The Current State of Web Frameworks

There's no doubt that the web has grown in complexity, and our development tooling options have grown along with it. Considering that the web is increasingly relied-upon as a key channel for mission-critical applications around the world, it should come as no surprise that our tools and frameworks have gotten bigger, and our choices more numerous. As the ecosystem grows and scales, so do our options.

But this doesn't change the fact that choosing the right tool for any job can be daunting.

Progress®

The Goal of this Whitepaper

That's where this whitepaper comes in. As an enterprise web developer or team, you're evaluating the web framework landscape because you either have a new project on the horizon, or you're looking to migrate or modernize an existing application. The goal of this whitepaper is to help you match your current needs and context against the capabilities, strengths and limitations of the most popular JavaScript frameworks available today.

We'll evaluate each framework against a few different dimensions and provide you some insight as to which framework is the right choice for your organization's next project. By the end of this document, you should have the knowledge you need to confidently select a tool and get started building that next great application.

The Contenders

Before we get started, we need to establish the libraries we'll review, and our evaluation criteria.

While there is a growing number of capable web frameworks out there, in order to keep this guide focused and digestible, we're going to limit our evaluation to the following three frameworks:

- Angular
- React
- Vue.js

Please note that in the case of Angular, this paper will stick exclusively to Angular version 2 and above, also known simply as "Angular." Angular versions <2, also referred to as "AngularJS," are outside of the scope of this paper since we're assuming you're evaluating newer, stable technologies for an upcoming development effort.

If after reviewing this paper, you're interested in exploring more web frameworks, you can expand your research and evaluate other contenders against the needs of your organization using the criteria below.

Progress®

Evaluation Criteria

Before we look at each framework in turn, I'll establish criteria for comparing each. As an enterprise, your team must consider a number of factors when choosing a software tool, from licensing to stability, support and even your ability to find developers with the right skills to be productive quickly. While not every criterion will apply to your company or project, this paper will attempt to capture every major factor to consider when selecting one of these tools, and you should apply the relevant factors when making your choice.

Category	Factor	What it is
Ecosystem	History and longevity	How mature is the framework? Why was it created?
	Popularity of framework	How widely used is the framework?
	Corporate support	Is there a corporate entity involved as a sponsor or interested party?
	Community and ecosystem	Is the framework supported by a large community? Is there a healthy ecosystem of plugins and libraries that extend core functionality?
Framework	Getting started experience and learning curve	How quickly can a new developer start using the framework? How hard is it to use as applications get more complex?
	Skills required	What skills does a developer need to have in order to be productive with this framework? Do they need to learn syntax or patterns that are specific to the framework itself?
	Completeness of offering	Does the framework provide everything "in the box" or do developers need to provide their own solutions to solve common problems?
	Performance factors	How does this framework perform in a complex application? What approaches does it take to help me make my apps run faster?
	Beyond the browser options	Can this framework be used in authoring non-browser apps, like mobile and desktop?

Progress®

Tooling	UI & component libraries	Are there UI & component libraries available for this framework?
	IDE & tooling support	Is there support for this framework in my IDE or other popular IDEs?
	Companion & CLI tools	What kind of tooling is available to help me create and manage apps with this framework?
Enterprise	Licensing	Under what license is this framework maintained? Does this license conflict with my enterprise's use of the tool?
	Support & upgrade paths	Do the maintainers of this library provide long-term support (LTS) versions? Are there enterprise support options available?
	Security	How do the maintainers handle security issues? How are security patches distributed?
	Talent pool & resources	How easy is it to hire developers who already know this framework, or who can learn it easily?

For each item on this list, I'll provide a brief explanation of what the criterion is, and why it may matter to your organization, before describing how each framework does (or does not) meet the criteria. Then, I'll conclude with some brief thoughts about how that criterion may factor into your decision.

Once we've gone through the list, I'll summarize with some general recommendations for making a choice, depending on key factors. You'll note that even though I am performing a side-by-side comparison, I'm not assigning a quantifiable number rating to any framework, given a factor. Your choice of framework ultimately depends on how these stack up against the criteria that are important to you, which can't be captured via a numeric rating, and may differ from project to project. As you read this paper and consider each factor, the content here should aid you, but the choice is ultimately yours to make.

Progress®

Ecosystem Considerations

In this section, we'll cover some of the ecosystem considerations surrounding each framework, including history, community, popularity, corporate involvement and notable apps built with this framework. There's comfort in selecting a mature tool that has momentum and broad interest, and this section will help you assess how each tool stacks up in that regard.

History & Longevity

What it is: How mature is the framework? Why and when was it created?

Why it matters: While the history and creation date of a framework is not a direct indication of its maturity, it does paint a picture of how the library started, and the chances it will stick around for the foreseeable future. Both are important for enterprises looking to create a long-lived application with one of these frameworks.

Angular

Angular is the oldest framework among those we cover here, first released in October of 2010. It's important to note, however, that AngularJS was completely rewritten from the ground up starting in 2014 and released as Angular 2 in October of 2016. This rewrite notably included a large number of breaking API and concept changes. Thus, when considering the history of Angular, it's important to note that the post 2.0 version of Angular has only been in the wild for a little over a year. That said, much of the first 4 years of AngularJS's history and real-world use factored heavily into the Angular rewrite, so it is fair to say that the library has a good deal of built-in maturity, post version 2. The current version of the library as of this writing is 5.0.1.

Progress®
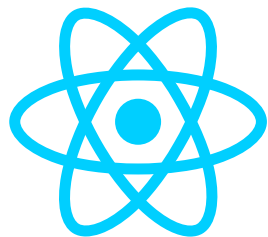
2013

React

React, sometimes referred to as React.js or ReactJS, was initially released in March of 2013. The current version as of this writing is 16.2. As with Angular, the React team recently rewrote the core of the library, with the first aspects appearing in the wild with the 16.0 release. Unlike Angular, however, this rewrite introduced a much smaller set of API changes and is considered an evolution of the Framework, as opposed to a new direction.

2014

Vue.js

Vue.js is the newest of the three frameworks we'll look at, first released in February of 2014. The library's creator, Evan You, started the project after working on AngularJS across several projects. At the time of writing, the current version is 2.5.9.

Motivations

Despite their similarities, these three frameworks differ in the initial problem they hoped to solve. It's important to be aware of these differences as they inform other criterion we'll review. Angular was first created to help teams and organizations address many of the challenges that were common when building Single-Page Applications, or SPAs. The motivation of the library was to create something that could address as many of these as possible, right in the box.

React, on the other hand, was designed to help developers create fast, simple and scalable UIs for complex web applications.

Finally, Vue.js was created to be a lightweight, progressive, incremental alternative to Angular. The author intended to provide something that was easy to get started with, but which could grow and scale as the complexity of your app grew.

Progress®

Verdict

All three frameworks are backed by a few years of history, and have matured to the point that they can be relied upon for enterprise applications. What's more, the rewrite of Angular, and more recently, React, is an indicator of ongoing maturation and stability of these libraries. That said, enterprises should carefully consider the history of those rewrites and breaking changes introduced before settling on a solution based on maturity alone.

Popularity

What it is: How widely used is the framework? What are some real-world examples where the framework has been deployed?

Why it matters: Popularity is the other side of the maturity coin from history. A popular library, meaning that it is used in real-world production applications, is an indicator that the library meets the needs of other organizations, and has had to improve in areas where it may not have, initially. Popular applications that have been deployed successfully by large organizations, especially those in a similar market as yours, indicate that a framework is worth serious consideration.

In this section, we'll take a look at the popularity of each framework along a few dimensions, both quantitative and qualitative. On the quantitative side, numbers like GitHub stars and monthly npm downloads provide some insight into how much a framework is used. However, these numbers don't differentiate between playground and real-world use, so we'll also look at some examples of real-world applications built with these technologies.

Framework	GitHub Stars	Avg. Monthly npm Downloads (6 months)	StackOverflow Tagged Questions
Angular 2+	~31,000	~1.93 Million	~87,000
React	~83,000	~5.85 Million	~67,000
Vue.js	~76,000	~0.78 Million	~12,000

All three frameworks have been implemented in production by notable brands, including those below:

Angular
- Google
- Wix
- Weather.com
- Healthcare.gov
- Forbes

React
- Facebook
- Netflix
- Paypal
- AirBnB
- Uber

Vue.js
- Alibaba
- Baidu
- Expedia
- Nintendo
- GitLab

Verdict

While all three libraries are quite popular, there's no doubt that React is the clear leader in terms of public adoption at scale. What's more, according to npm download trends over the last year, React's lead continues to widen, while Vue seems poised to pass Angular in the coming months.
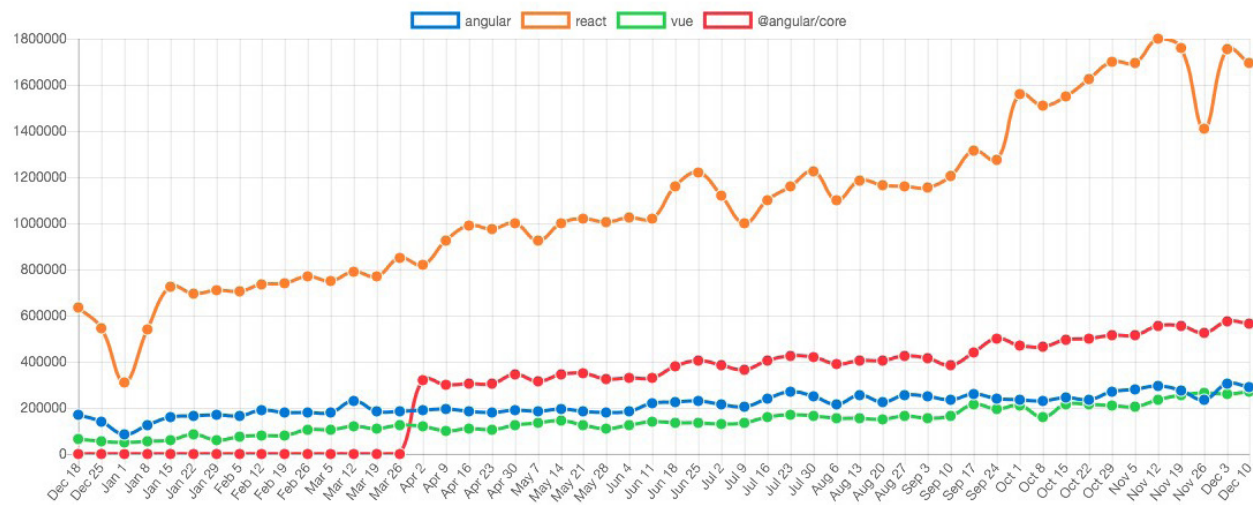
Source: www.npmjs.com
Graph created with: www.npmtrends.com/angular-vs-react-vs-vue-vs-@angular/core

All three libraries are also used by well-known brands, in production. If one of the companies listed above sits in a market or industry close to your own, this is a good indicator that the framework in question could be a good fit for your needs. If you want to search for other companies using these frameworks, you can use tools like StackShare and LibScore to find other public sites and apps that make use of React, Angular or Vue.

Corporate Support

What it is: Is there a corporate entity involved as a sponsor or interested party?

Why it matters: Adopting an open source library, even a popular one, can be a risky proposition for an enterprise. If a library is abandoned by its creators, organizations are left with no safety net, and must either take on the task of internal maintenance, or undergo a migration to a new solution. When a library is baked by a corporate sponsor, it serves as a telling indicator that the library has the support it needs to keep the maintainers engaged and involved, and minimizes the risk of adoption.

All three libraries have some form of third-party support, though that support differs for Vue. While Angular is stewarded and actively-maintained by Google employees, and the same is true of React through Facebook employees, Vue is largely backed by a handful of small- and medium-sized organizations that support core team member's time and effort via Patreon, a popular online membership model for creators.

Framework	Major Corporate Sponsor(s)
Angular 2+	Google
React	Facebook
Vue.js	None. Project is Patreon-supported by several companies

Verdict

While the involvement of Facebook and Google can provide some comfort to enterprises evaluating these tools, you shouldn't allow this to lull you into a false sense of security. Both Facebook and Google have notably abandoned popular tools and products in the past, and they could easily do so again if the needs of their business take them in a new direction. No matter which tool you choose, you should always have a "Plan B" on paper if the framework you've chosen is abandoned or moves in a direction that no longer meets your needs.

Progress®

Community & Ecosystem

What it is: Is the framework supported by a large community? Is there a healthy ecosystem of plugins and libraries that extend core functionality?

Why it matters: While this is somewhat of an extension of the popularity factor, it's possible to have a popular library or framework with a small or shrinking ecosystem. This is usually an indicator that the framework is trending in the wrong direction and, while still used in production applications, isn't seeing much innovation. jQuery is probably the best example of this in the web ecosystem.

While it can be hard to definitively rate a community quantitatively, we can look at a couple of factors to determine if our three frameworks have active ecosystems around them. The first is the number of npm libraries and packages available for developers who use each framework. In this realm, React has a healthy lead.

Framework	# of npm packages
Angular 2+	~18,322
React	~43,000
Vue.js	~8,174

The numbers above are directly indicative of the number of plugins and extensions that framework developers have at their disposal, and all three have sizable ecosystems, in that regard. Indirectly, we can look at these numbers as indicative of the number of "super developers" working with

each framework, assuming that a developer who publishes a utility or library to npm for others to use is more likely to fit this label than not. It's not an exact metric, but helpful as a guide.

Another factor is the size of the core team that maintains the framework. In theory, a large core team is an indicator that team members are involved in supporting and helping the community as they implement the framework for their applications. Here, Angular is the leader with a core team of 36, while Vue's core team is 16. The React team does not disclose its membership, though a review of core team meeting notes suggests that the team is around 6-10. Bear in mind, that the team size is also somewhat dependent on the size and scope of the library. As we'll discuss later, Angular is the largest codebase of the three frameworks, so a large core team makes sense.

Verdict

When it comes to the size of the team maintaining the framework, Angular is the clear leader. However, the plugin and library ecosystem of React is quite large, bigger than both Angular and Vue combined.

Progress®

Framework Considerations

Now let's turn our attention to a deeper dive into these frameworks themselves. This category will focus on features of each framework that should inform your decision, from the learning curve to the skills required, the completeness of the offering, performance, and options for using the framework for desktop and mobile apps. These factors should help you weigh your choice towards a framework that has a reasonable learning curve and matches the skills and context of your team and organization.

Getting Started Experience

What it is: How quickly can a new developer start using the framework? How hard is it to use as applications get more complex?

Why it matters: There are three critical dimensions to the getting started experience with a new framework: how quickly a new developer can pick up the framework and understand its core concepts, how complex it is to create a new greenfield application using the library, and how easily those concepts transfer as a real-world application gets more complex. A framework with a solid getting started experience is one that you can try out with little investment before you decide to adopt, that explains core concepts through a simple demo app and which elegantly scales up as complexity rises.

Angular

Angular has the most complex getting started experience of the three frameworks. The quick start requires installing the Angular CLI and touching a couple of files in order to get a simple app running, and the final app doesn't do much to introduce core Angular concepts. The tutorial, Hall of Heroes, is more comprehensive, takes a few hours to complete and does cover all of the major core concepts of the Framework. Neither guide has an online equivalent that developers can walk through before downloading code and doing local development.

Progress®

The Hall of Heroes tutorial uses the Angular CLI and shows off the same project structure you'd see in a real-world app, which does help lessen the learning curve when it's time to start your app. What's more, the guide does cover all of the major concepts you'd need to understand in order to develop any non-trivial application.

React

The React getting started experience comes in two flavors: a very basic Hello World application, and a more in-depth guide that walks the developer through creating a tic-tac-toe game. The latter explains all of the core concepts in React at a high level and can be completed in less than an hour by most developers. The tutorial is also available as an interactive CodePen workspace, so it can be used before downloading it or writing local code.

For project creation, the React team supports a tool called create-react-app, an npm utility that manages the complexity of React and other tools like Babel and Webpack, and provides a default build configuration for your app.

From a scale perspective, core React concepts stay the same for both simple and complex apps and the framework doesn't require special considerations for these. That said, React does not provide enterprise state management in-the-box, and some concepts like server-side-rendering do require additional work, so you'll want to take these into consideration during your evaluation.

Vue.js

Getting started with Vue.js requires only a single script tag in your HTML, making it the simplest experience of the three. Vue also provides an interactive JSFiddle example that can be used to try out the library before downloading it or writing local code. Both approaches are backed by a comprehensive tutorial that covers the basics of the framework.

For project creation, Vue provides an official CLI that helps scaffold new applications and manage builds, tests and deployment.

From the standpoint of scale, Vue is a progressive framework, meaning that certain concepts like routing, state management and server-side rendering are not covered by default. However, Vue does have official libraries for these and extensive documentation on using them when needed.

Verdict

If you're looking for the easiest framework to get started with in the shortest amount of time, Vue.js is the clear winner here, followed by React, which is slightly more complex, but easy enough for a developer to be comfortable with in a day or two. Of the three, Angular has the steepest learning curve, by far. As we'll see shortly though, its feature set is more comprehensive, so the investment of time may be worth it for your team, especially if you anticipate building a long-lived, complex app.

Progress®

Skill Requirements

What it is: What skills does a developer need to have in order to be productive with this framework? Do they need to learn syntax or patterns that are specific to the framework itself?

Why it matters: Even though each of our candidates are JavaScript frameworks, they are not alike in the style of JavaScript programming they encourage, or the unique concepts that you must learn to use them. Choosing a framework will require an understanding of the skills required for each, as well as the skill preferences of your team.

Framework	Language Concepts
Angular 2+	TypeScript
React	JS (ES6+), JSX, CSS-in-JS
Vue.js	HTML, JS (ES5+), CSS

Angular

Though a JavaScript framework, Angular 2 was written from the ground-up in TypeScript, a statically-typed superset of JavaScript from Microsoft. Because Angular is written in TypeScript, you're expected to use it for your Angular apps as well. Angular apps make heavy use of TypeScript language concepts not yet found in JavaScript, like decorators and static types. The use of TypeScript does add learning overhead to Angular, but developers more comfortable in static-typed languages like C# and Java are likely to find the language more familiar.

Angular itself also has a number of baked-in patterns and conventions that many enterprise developers will be familiar with, including Dependency Injection and heavy use of Services.

React

React requires developers to use modern JavaScript (ES6+) in their apps, and leverages Babel to compile your React code to cross-browser compatible JavaScript for execution. React uses many ES6 concepts in the framework itself, like string interpolation, arrow functions, and modules, and you'll need to understand these in order to use the framework.

From a framework convention standpoint, the biggest unique aspect of React is its use of JSX, an XML-like syntax used for authoring UI markup in JavaScript code, as in the example below.

```
// Using JSX to express UI components.
var dropdown =
  <Dropdown>
    A dropdown list
    <Menu>
      <MenuItem>Do Something</MenuItem>
      <MenuItem>Do Something Fun!</MenuItem>
      <MenuItem>Do Something Else</MenuItem>
    </Menu>
  </Dropdown>;
render(dropdown);
```

While JSX is not required to use React, it's uncommon to find code samples or tutorials that don't use it, so developers will need to have some comfort with the concept.

Finally, it's becoming increasingly popular in the React world for developers to take a "CSS-in-JS" approach to styling their applications. There are several approaches and libraries which support this, but the basic idea is that global stylesheets and CSS files are largely avoided and that each UI component in your application be responsible for styling itself. This approach is optional, but you should understand it when evaluating other libraries and plugins for your app that may take this approach.

Vue.js

Of the three frameworks, Vue is the least disruptive in terms of requiring the use of new languages or technologies. It works with ES5, though ES6+ is supported. It relies on HTML for UI composition and CSS for styling. It enables the use of Components, which are leveraged with custom HTML syntax, but this concept is common to all three frameworks and does not represent a steep learning curve.

From a concept standpoint, Vue will be most familiar to the traditional web developer is it relies on HTML, CSS and JS as they are traditionally used, with a few exceptions. Similar to Angular, Vue does make use of custom HTML attributes to handle data-binding, conditional and control-flow statements, though developers who are familiar with early MVVM libraries like Knockout or Backbone will find many similarities to those libraries.

Verdict

Your choice here will depend on the skills of your team. If your project team consists largely of C# and/or Java developers who are dubious of JavaScript, Angular is the clear winner. If, on the other hand, your team is made of traditional web developers who have built MVVM apps before, Vue will feel the most familiar. Finally, if your team prefers the cutting edge of JavaScript and standards, React would serve as a good fit.
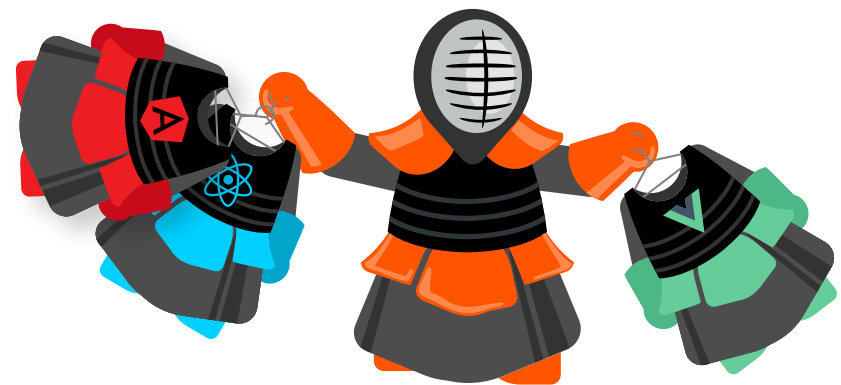
Completeness of Offering

What it is: Does the framework provide everything "in-the-box" or do developers need to provide their own solutions to solve common problems?

Why it matters: Each of these frameworks was built with a certain intent and to solve a certain class of problems, and this has dictated their evolution over the years. It has also lead to a conscious choice by each framework of what features to include 'in-the-box" versus those left up to the extension community or to your team. Depending on the complexity of your app, you may prefer a framework that ships with everything we need, or something lightweight that lets you add extensions for certain types of problems.

Framework	Completeness of offering	How to extend it
Angular 2+	Everything "in-the-box"	Community libraries, frameworks and utilities
React	Focused on UI-management only (including UI state)	Community libraries, frameworks and utilities
Vue.js	Essential features by default, official libraries for advanced functionality	Official libraries, community libraries, frameworks and utilities

For a complex application, Angular has everything in-the-box, from UI management, to complex state management, routing, end-to-end testing, and more. React, on the other hand, tries to stay UI focused, meaning that you're going to need to add in another library for advanced features like complex state management (Redux and MobX are two popular choices). Finally, Vue's approach is progressive, in nature. By default, only the most basic features are enabled so you can get up and running quickly. And when you need a feature like state management, Vue provides an official solution in Vuex.

Verdict

If you're looking for an all-in-one solution, Angular is the most complete of the bunch. However, this could be a drawback if you're building a smaller app, or one that you need to develop quickly and scale over time. In the latter case, your decision depends on whether you want official extensions from the project, as in Vue's case, or to leverage a much larger ecosystem of libraries as features, as is the case for React.

Progress®

Size & Performance

What it is: How does this framework perform in a complex application? What approaches does it take to help me make my apps run faster? How large is the framework itself?

Why it matters: A real-world application that performs poorly is an application that won't be used. When choosing a framework, you should be aware of the performance tradeoffs that go along with that choice, from library size to runtime performance, and even the tools the library provides to help you squeeze the most performance out of your app.
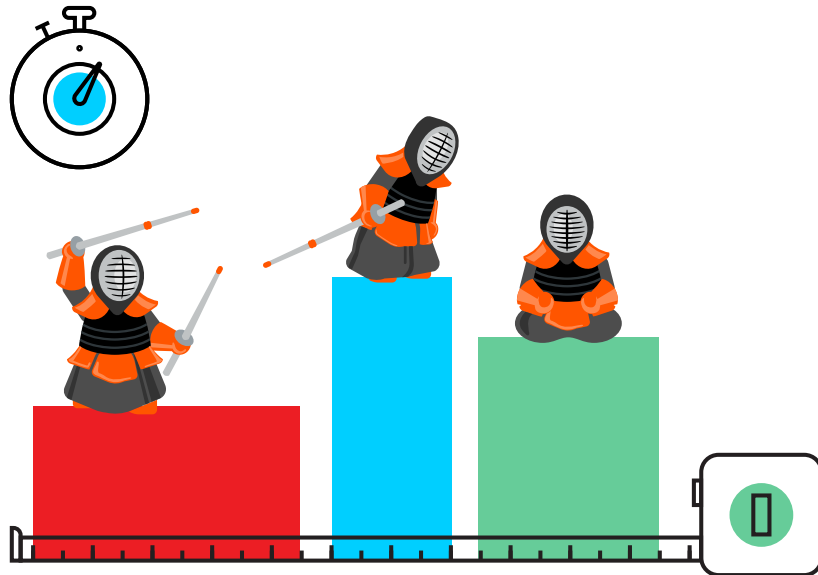
Library Size

Angular's "everything in-the-box" approach has a hefty price tag when it comes to library size, with the gzipped file coming in at 143K. Compare this to 43K for React and 23K for Vue.

Runtime Performance

While performance benchmarks can be gamed and shouldn't drive your decision, they can be useful when comparing the relative speed of one framework to the next. The images below contain some recent benchmarks across recent versions of all three frameworks, which tested each against common app use cases.

While the deviation between all three libraries is not significant, it's clear that Angular is the slowest performing of the three, with React slightly faster and Vue the fastest.

Duration in milliseconds ± standard deviation
(Slowdown = Duration / Fastest)

Name	angular-v2.4.9-keyed	react-v15.5.4-keyed	vue-v2.3.3-keyed
create rows Duration for creating 1000 rows after the page loaded.	197.3 ± 9.3 (1.2)	188.910.9 (1.1)	166.7 ± 8.6 (1.0)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	201.3 ± 5.9 (1.2)	201.0 ± 6.4 (1.2)	168.5 ± 5.0 (1.0)
partial update Time to update the text of every 10th row (with 5 warmup iterations).	12.8 ± 3.3 (1.0)	16.5 ± 2.3 (1.0)	17.3 ± 2.9 (1.1)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	4.9 ± 3.4 (1.0)	8.8 ± 3.4 (1.0)	9.3 ± 1.7 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	13.5 ± 1.1 (1.0)	14.7 ± 0.9 (1.0)	18.3 ± 1.5 (1.1)
remove row Duration to remove a row. (with 5 warmup iterations).	46.4 ± 2.0 (1.0)	47.2 ± 3.2 (1.0)	52.6 ± 2.7 (1.1)
create many rows Duration to create 10,000 rows	1866.7 ± 55.4 (1.2)	1852.4 ± 29.0 (1.2)	1587.5 ± 33.9 (1.0)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	365.1 ± 52.3 (1.1)	345.6 ± 10.4 (1.0)	399.5 ± 11.0 (1.2)
clear rows Duration to clear the table filled with 10.000 rows.	389.9 ± 39.0 (1.5)	398.4 ± 8.2 (1.6)	254.5 ± 5.0 (1.0)
startup time Time for loading, parsing and starting up	104.9 ± 8.6 (1.9)	70.0 ± 2.9 (1.2)	56.6 ± 2.5 (1.0)
slowdown geometric mean	1.17	1.12	1.05

Memory allocation in MBs ± standard deviation

Name	angular-v2.4.9-keyed	react-v15.5.4-keyed	vue-v2.3.3-keyed
ready memory Memory usage after page load.	10.6 ± 0.1 (1.4)	4.5 ± 0.1 (1.2)	3.8 ± 0.0 (1.0)
run memory Memory usage after adding 1000 rows.	10.6 ± 0.1 (1.4)	9.7 ± 0.1 (1.3)	7.5 ± 0.1 (1.0)

Source: www.stefankrause.net/js-frameworks-benchmark6/webdriver-ts-results/table.html

Progress®

Performance Tooling

Of course, performance is not just about the framework, but about the app you build using a framework. To that end, a solid framework will provide features that ensure that your final apps deliver the best experience possible to your end users. This includes features like:

- Ahead-of-time compiling - Removing a dependency on a run-time compiling by producing static assets in advance.
- Bundling - Creating the smallest code, style and markup files for production.
- Tree-shaking - Removing unused code and dependencies from your app.
- Lazy-loading - Only loading code when needed by the app, and not ahead of time.
- Server-side rendering - Representing your app as strings of HTML that can be indexed (for SEO purposes) and downloaded faster, improving the first load experience of your app.

As the heaviest library of the bunch, it should come as no surprise that Angular ships the most performance-focused tooling. That said, all three frameworks have great built-in tooling that helps ensure that the library itself doesn't adversely impact the performance of your app.

Verdict

Your choice here will depend heavily on the type of app you're hoping to build. A mission-critical back-office inventory management system may need all the features that Angular has to offer, with fewer worries about raw UI performance (although you should be careful not to assume that it's acceptable for enterprise apps to be slower than consumer apps), while an interactive consumer application should worry about how long it takes for a user on a slow connection to see the first content in your app. Either way, you should investigate the performance tooling that your framework of choice has to offer, and plan to use it from the start if performance is a key factor in your app.

Framework	Built-in Performance Features
Angular 2+	Ahead-of-Time compiling, Bundling, Tree-shaking, Lazy-loading, Angular Universal (Server-side rendering)
React	Focused on UI-management only (including UI state)
Vue.js	Essential features by default, official libraries for advanced functionality

Progress®

Beyond the Browser Options

What it is: Can this framework be used in authoring non-browser apps, like mobile and desktop?

Why it matters: It's becoming increasingly common for customers to demand apps that function in the browser, on a mobile device and even natively on the desktop. The problem is that each experience typically requires its own set of languages, tools and features, and most enterprises don't have the resources to triple the size of a development team for every project. In response, a bevy of web-to-native tools have arisen in the last several years that look to help your web team reuse its skills, while building apps that look, feel and in some cases are, fully-native on the desktop or a mobile device.

The following table lists some popular libraries for each framework.

Framework	Non-Browser Support
Angular 2+	NativeScript (Native Mobile), Ionic (Hybrid Mobile)
React	React Native, react-native-renderer (cross-platform iOS and Android)
Vue.js	NativeScript (Native Mobile), Weex (Hybrid mobile), Quasar (Hybrid & Desktop)

Verdict

If you're looking to take your web app beyond the browser, all three of our contenders have popular companion frameworks and libraries that let you reuse those web skills in creating mobile and desktop applications.
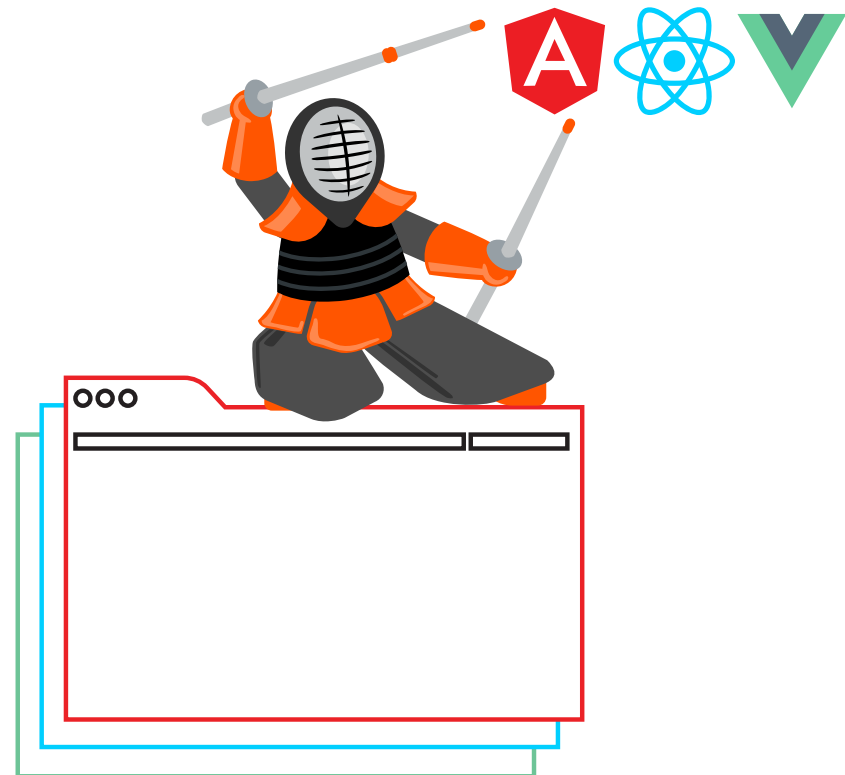
Tooling Considerations

In the modern era of web development, choosing a framework is about more than the framework and ecosystem around it. It's also about the tools surrounding those frameworks, especially UI and component tools, development environments (IDEs) and companion tools, including command-line (CLI) utilities. All of these tools are designed to make your team's life easier, and can greatly speed-up your development workflow if set up correctly. In this section, we'll look at each major area of tooling and see where each framework stacks up.

UI & Component Libraries

What it is: Are there UI & component libraries available for this framework?

Why it matters: For a complex web application, creating beautiful, responsive and well-performing UI can be challenging. Thankfully, there are a bevy of UI and component libraries that can help your team make that happen.

Verdict

Each framework is backed by excellent UI & Component libraries if what you're looking for is an all-in-one UI framework. For one-off components, pay attention to the size of the ecosystem as this is an indicator of the breadth of components you'll have to choose from for your app.

Framework	UI & Component Libraries
Angular 2+	Kendo UI, Angular Material, Bootstrap
React	Kendo UI, Material-UI, React-Bootstrap
Vue.js	Kendo UI, vue-material, bootstrap-vue

In addition to the libraries listed above, each framework's community has produced a litany of one-off components that you can use where a framework won't do. Check out the online resources for each framework (GitHub, npm, etc.) and make sure that there's a solid set of choices based on the app you're planning to build.

Progress®

IDE & Tooling Support

What it is: Is there support for this framework in my IDE or other popular IDEs?

Why it matters: Developers spend a lot of time in their IDEs, and nothing makes a developer hate a new tool more than finding that it's hard to use with their favorite code editor. Good IDE support will help smooth a developer's learning curve with a new tool or framework, and supercharge their productivity once they've gotten comfortable.
For this category, we'll take a look at some major IDEs on the market, and what kind of support they offer for each framework (built-in tooling, plugins, community extensions, etc.)

IDE	Angular Support	React Support	Vue.js Support
Visual Studio	3 marketplace extensions for snippets, starter kits and templates	2 marketplace extensions for snippets and starter kits	3 marketplace extensions for snippets and starter kits
Visual Studio Code	~100 extensions for snippets, templates and more	~100 extensions for snippets, templates and more	~46 extensions for snippets, templates and more
WebStorm	4 marketplace plugins for syntax highlighting, templates and more	4+ marketplace plugins for syntax highlighting, templates and more	1 formal framework integration plugin from the JetBrains team
Sublime Text	Sublime package curated by the Angular UI team	Sublime Text helpers curated by the Facebook team, though no longer maintained	Syntax highlighting extension curated by the Vue.js team
Eclipse	3 marketplace plugins to add official syntax, TypeScript and debugging support	No extensions available	No extensions available

Verdict

If your team's IDE of choice is Visual Studio, VS Code or WebStorm, each framework is well-supported via community extensions that will speed up your development. If you're using Sublime Text, syntax highlighting extensions do exist, but you'll miss out on productivity features like snippets and templates common in other editors. Finally, if your team is set on using Eclipse and you want built-in language productivity, Angular is your framework of choice.

Companion & CLI Tools

What it is: What kind of tooling is available to help me create and manage apps with this framework?

Why it matters: Even with a solid IDE and some good UI tools and components, there are aspects to building a web app that can bog developers down if they're not automated. This can be everything from scaffolding new components to running local servers, debugging and more. Companion tools and utilities can help automate all of these tasks and keep developers productive.

Companion tools and utilities can automate a lot of the grunt work involved in building and maintaining a web application. In this section, we'll look at the support each framework provides for the following project activities

- Project scaffolding - Is there a tool to scaffold a new app with all dependencies and project structure in place?
- Feature scaffolding - Is there a tool to scaffold new components, services and other project features?
- Linting & unit test automation - Does the tool provide the ability to lint your code and run unit tests?
- End-to-end test automation - Does the tool provide the ability to run end-to-end tests in a real or headless browser?
- Local development server - Does the tool make it easy to run a local server for development and debugging?
- Hot-reloading - Does the tool refresh your local app as you write code and make changes?
- Browser developer tools - Are there extensions in the browser for local development and debugging?
- Bundling and Deployment tools - Does the tool provide utilities for bundling code for deployment

Feature	Angular (via Angular CLI)	React (via create-react-app)	Vue.js (via vue-cli)
Project scaffolding	Yes	Yes	Yes, includes template support
Feature scaffolding	Yes	No	No
Linting & unit test automation	Yes	Yes	Yes

Progress®

End-to-end test automation	Yes	No	Yes, via Nightwatch
Local development server	Yes	Yes	Yes
Hot-reloading	Yes	Yes	Yes
Browser developer tools	Augury (3rd party) for Chrome	Official React Developer Tools for Chrome	Official Vue.js Developer tools for Chrome
Bundling and deployment tools	Yes	Yes	Yes

Verdict

Based on the list above, you can feel confident knowing that all three frameworks have solid tooling support for the types of development tasks you'll want to automate. Angular gets a slight edge when it comes to feature scaffolding common project assets, but aside from this, each framework provides a robust set of tools.
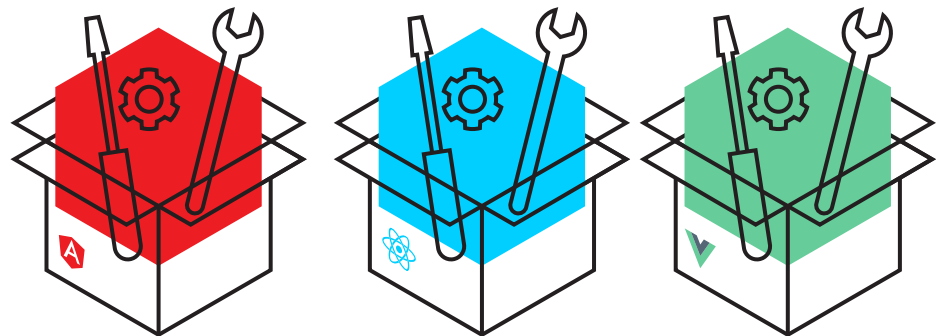
Enterprise Considerations

The final category of factors we'll look at are those considerations that uniquely affect enterprises. This is not to say that these things do not apply to small organizations, but rather that these factors can often yield a make or break decision on a library or product that otherwise checks all the right boxes for a company. Things like licensing structure, support options, security and the ability to hire developers with the right skills are all factors we'll explore in this final section.

Licensing

What it is: Under what license is this framework maintained? Does this license conflict with my enterprise's use of the tool?

Why it matters: Enterprises leveraging open source for their products should do so responsibly and be careful to use software with licenses that do not inadvertently violate source distribution clauses or extend coverage into your IP and source code.

As of this writing, all three of the frameworks we're evaluating are covered under a permissive MIT License, which puts very limited restriction on reuse and likely has ideal license compatibility with your internal licensing policies and that of other software your organization relies upon. An MIT License can be used within any proprietary software you create, provided that a copy of the license is included with the source. When you install one of these frameworks in your app via their CLI tools or npm, a license file will be included by default.

There is one caveat with React, however. Prior to the most recent releases, React was licensed under a BSD + Patents license. This was an intentional decision on the part of Facebook to ensure that none of their products inadvertently granted rights to any patents that they deemed valuable and essential to their business. While the BSD license is still permissive, many were concerned about the patent restrictions. After some discussion, Facebook decided to relicense React under the MIT license moving forward, which took effect with version 16 in November of 2017.

Verdict

All three frameworks ship with the permissive MIT license. If you decide to use React, be sure to stick with version 16 or above, as earlier versions are covered under a BSD + Patents license that may conflict with the legal standards of your organization.

Progress®

Support & Upgrade Paths

What it is: Do the maintainers of this library provide long-term support (LTS) versions? Are there enterprise support options available? What is the upgrade experience for this library?

Why it matters: Enterprise applications can be complex in scope and take time to implement. What's more, when in production, these apps must remain stable and predictable for a long period, depending on the objectives of the app. In these conditions, many enterprises want to ensure that the libraries they rely on will remain supported for a reasonable period of time, and that the maintainers provide clear upgrade paths, while being responsible about introducing breaking API changes.

When it comes to open source and support, enterprises are often wary of taking on large dependencies, and with good reason. When you rely on a large framework or library, you're saving the expense of developing that functionality yourself, but you're also taking a bet that the functionality will work, and be supported if it breaks.

For the purposes of this discussion, all three of these libraries, when adopted, become the critical engine that drives your application, and you want to ensure that your engine will run for the foreseeable future. So how does each library stack up when it comes to support, versioning, API deprecation, and more?

Angular

When Angular announced their shift from AngularJS (v1) to Angular (v2+), it came with a number of core improvements, and a bevy of API and dependency changes that made migration a monumental effort for most apps. For many, migrating from Angular 1 to 2 was more akin to a rewrite than a migration, which frustrated many in the development community.

The good news, however, is that this experience led to some needed changes to how Angular handles versioning and long-term support

Progress®

for its framework. Starting with version 2, the Angular team defined a comprehensive version and release strategy that includes semantic versioning, time-based releases and a clear deprecation policy for API changes. In short, the team ships patches (essential bug fixes) each week, 3 minor and one major update every six months.

And while Angular's API becomes more stable as it matures, breaking changes do happen from time to time. In those cases, the Angular team will announce the deprecation and include an upgrade path for developers using the deprecated feature. The deprecated feature will remain stable for more than six months and won't be removed until the next major upgrade. That means that minor updates will never ship breaking changes that will cause issues with your apps.

Finally, in early 2017, Angular announced a formal long-term support (LTS) policy that took effect in October of 2017, with the release of Angular 5. From that point forward, LTS versions of Angular will continue to receive critical fixes and patches for one year, even as the majority of development shifts to the new version.

React

In early 2016, React announced a formal versioning scheme and deprecation policy. As with Angular, the React team introduces API changes as new features in minor versions, and adds deprecation warnings (in your developer console and terminal), while still supporting the deprecated feature. These deprecated features are then removed in the next major version. Unlike Angular, however, the React team has made no public commitments around a consistent or regular release cadence for

patches, minor and major releases.

In addition, React does not have an LTS policy, and has made no public announcement around introducing one. Many claim that the migration hurdle from one version to the next is historically small and, therefore, LTS support is not needed.

Vue.js

With the release of Vue 2.0, the core team does provide a roadmap that lists priorities for the next version of the library, which for the 2.x version listed at the time of writing, includes a commitment to not introduce breaking changes. However, there is no formal statement about deprecation policies or any form of long-term support for the current version of Vue, although this may change as the library matures.

Verdict

When it comes to a well-defined, enterprise-friendly support policy and roadmap, Angular is the clear leader. If you need clear deprecation guidance and the comfort of an LTS plan for your app, Angular is the way to go at this point. That said, React has kept a quite stable API over the last few years, and has a solid deprecation policy that makes it easy for developers to upgrade. Since version 2, Vue.js has also been stable, but the lack of any formal commitments about release cadence, deprecation policies and LTS versioning may factor into your final decision about the framework.

Security

What it is: How do the maintainers handle security issues? How are security patches distributed?

Why it matters: Enterprises relying on open source software want to ensure both that a project has a safe process for reporting potential security vulnerabilities and that security issues are resolved and patches distributed quickly.

The security of public websites and applications has always been a concern for most enterprises, and has grown as more business is done online every day. When adopting an open source library the scale of those we discuss here, you want to ensure that there are policies in place to deal with security issues that arise.

Angular provides a comprehensive security guide in their docs, which not only provides an email address for securely reporting security vulnerabilities and a pointer to Google's corporate security policy, but a detailed list of pointers for implementing secure practices in your Angular apps. As mentioned above, the Angular team releases patches weekly, including security fixes, when appropriate.

React provides a pointer to a Facebook security bounty program in their contribution guide and does encourage developers to securely report issues, but otherwise provides no formal guidance or statements about how patches are distributed.

Vue provides no security email or security statement on their site or in their GitHub repository, as of this writing.

Verdict

If your organization is concerned about the security of your web framework, Angular provides the broadest assurances.
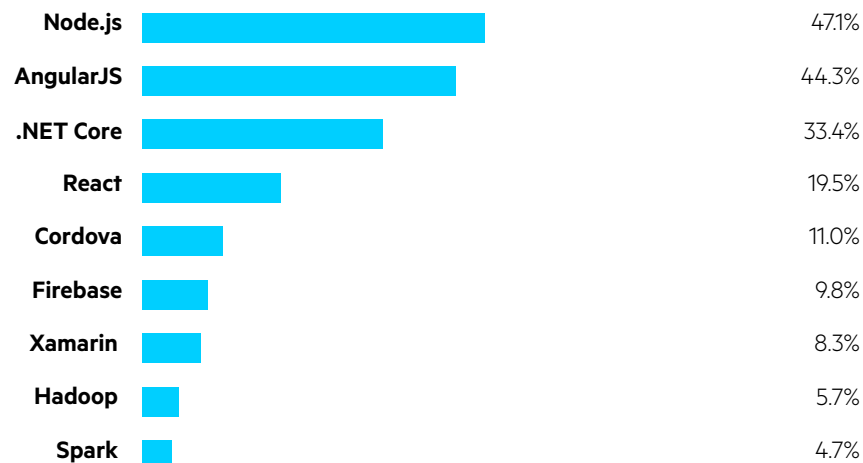
Talent Pool & Resources

What it is: How easy is it to hire developers who already know this framework, or who can learn it easily?

Why it matters: If you're going to bet on a web framework for your next app, you need to know not only that your developers can learn it, but you can hire developers as your development effort grows.

While this criterion is somewhat related to the popularity and ecosystem of these frameworks, those factors don't paint a complete picture of what the available developer resource pool looks like. Most importantly, popularity and ecosystem size is more an indicator of current state and not necessarily of future growth. And when you're picking a web framework to drive your development for a complex project or for years to come, you want to ensure that developers will continue to be interested in working with that library.

It's common, when evaluating talent pool size to look at surveys like the StackOverflow Developer Survey and draw some conclusions about what developers in the industry are working with, day-to-day. The most recent survey, for instance, indicates that far more developers are working with Angular than React, as depicted below.
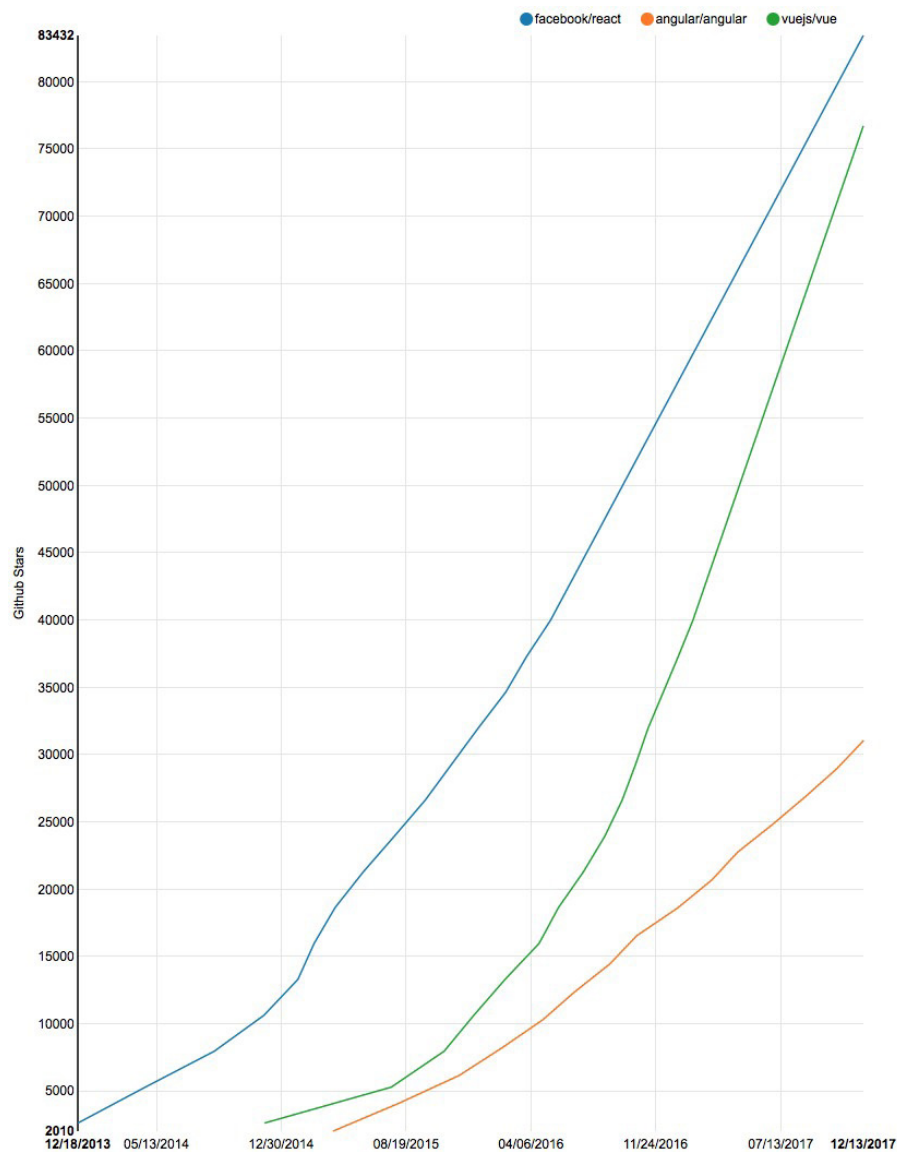
Node.js	47.1%
AngularJS	44.3%
.NET Core	33.4%
React	19.5%
Cordova	11.0%
Firebase	9.8%
Xamarin	8.3%
Hadoop	5.7%
Spark	4.7%

Source: Stack Overflow Developer Survey 2017

Progress®

And while this does paint a picture of what developers are working with today, it doesn't capture what they want to work on in the future. To determine that, we can look at the State of JavaScript survey, which asks developers to discriminate between frameworks they've used before and want to use again, versus those they do not. Here, as illustrated below, React is a clear leader among respondents, with Vue.js gaining in interest and popularity. And while interest in Angular 1 is understandably low, interest in Angular 2+ has been waning year-over-year, indicating that a large number of developers prefer the other two libraries on our list, or even another major framework.
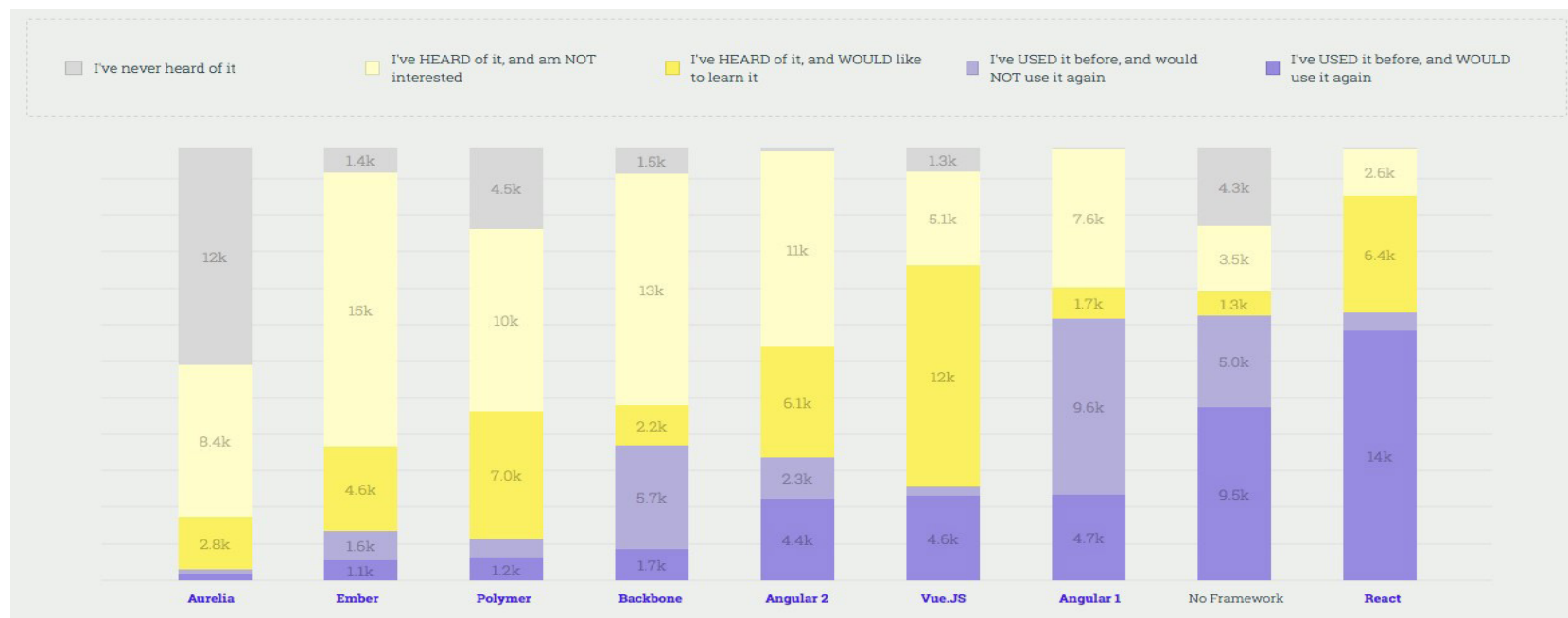
Source: The State of JavaScript 2017

Finally, while anecdotal, if you look at the growth of these libraries GitHub stars over the last 4 years, there's no denying that React and Vue have experienced a more hockey-stick-like curve to Angular's linear growth.

facebook/react   angular/angular   vuejs/vue

Github Stars

83432
80000
75000
70000
65000
60000
55000
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
2010

12/18/2013   05/13/2014   12/30/2014   08/19/2015   04/06/2016   11/24/2016   07/13/2017   12/13/2017

Source: github.com
Graph created with: www.timqian.com/star-history/#facebook/react&angular/angular&vuejs/vue

Verdict

Angular is still widely used, but data suggest that the rise of React and Vue may be coming at the expense of Angular. Depending on your hiring needs and the size and scope of your project, you may want to consider selecting React or Vue for this reason.

Progress®

Making a Choice

By now, I'm sure you agree that there are a lot of factors to consider when choosing between Angular, React and Vue.js for your next application. And while there are many dimensions above where each library stacks up well against the others, there are a few clear cases where one leads the pack.

More importantly, I'm guessing that one library stands out clearly to you given the context of your organization, the needs of your customers, the skills of your development team and the scope of your application. So, while I'm not going to declare one of our contenders the hands-down winner, I will provide some guidance around how your specific context might lead you to choose one of these over the others. This list is informed, of course, by all of the content above. As you review this, look for the conditions and factors you care about, and choose accordingly.

- If you need an all-in-one solution that has everything you need for a complex app, **Angular** is your best choice.

- If you want a high-performing UI framework to serve as a tentpole for a well-supported ecosystem of best-of-breed companion tools and libraries, **React** is the way to go.

- If you're looking for a solution that starts with some simple defaults and can be expanded as your app grows in complexity, **Vue's** progressive approach makes it a good choice.

- If you're looking for a library backed by a big corporate sponsor, **Angular** and **React** can give you that.

- If your engineering team is well-versed in C# or Java and prefers statically-typed systems, they'll be most comfortable in **Angular**.

- If your team consists of JavaScript ninjas who are always using the latest and greatest language features, **React** will be familiar to them.

- If your team prefers clean separation of HTML, JS and CSS, or works in a structure where designers and developers collaborate on components, **Vue** will fit that structure well.

- If your organization needs LTS support and strong security assurances, **Angular** is the best choice.

- If you're looking to grow your organization and you want to hire cutting edge developers motivated to work with new technologies, **React** and **Vue** seem to have captured developer mindshare for the future.

The bottom line is that your context is the key. There is no wrong choice between the three, as long as it is informed by the needs of your organization.

Choosing the right framework can seem like a risk with a good deal of complexity. Hopefully this guide has helped you explore the factors to consider when deciding on the right framework for your next enterprise app.

Progress®

Once the Choice Is Made, the Journey Begins

Once you have decided on a framework, it's time to start planning and we've got your back there as well. If you chose Angular, take a look at "Planning an Angular Application". If you fancy React better, there's "Planning a React Application" for you - and if you're set on Vue – stay tuned, we have the planning resource in the works!

About the Author

Brandon Satrom

Brandon is the founder of Carrot Pants Press, a maker education and publishing company, the founder and CEO of Tangible Labs and an avid tinkerer.

→ 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 and charts to basic buttons and menus. Try Kendo UI.

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 1700 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

Progress®