

If you have previous programming experience but are a newcomer to frontend JavaScript development, the array of jargon and tools can be confusing. Without getting bogged down in detailed discussion, let's take a general survey of the current "JavaScript landscape". This should orient you sufficiently to start your journey into frontend development.

Key terms are **bolded**. If you want to skip ahead to a repository with working boilerplate, I've put together a Github repository with my [recommendations](#).

How does client-side JavaScript work, and why use it?

Key terms: DOM (Document Object Model), JavaScript, async, AJAX

To write effective frontend code, you need a basic understanding of how HTML, CSS, and JavaScript fit together to create web pages.

When the **client** (usually a **browser**) accesses an HTML page, it downloads it, parses it, and then uses it to construct the **DOM** (Document Object Model). JavaScript can interact with and modify the DOM, which is how you produce **interactive** websites. [Here's a basic intro to the DOM](#).

How is JavaScript included in your page? That's a separate can of worms, but it [begins with script tags](#).

JavaScript execution [blocks](#) DOM construction. This means that spending lots of time executing JavaScript will make your page feel unresponsive. To avoid this, client-side JavaScript is often heavily **asynchronous**. (You may have heard of **AJAX**, which simply stands for asynchronous JavaScript and XML.)

If you're building an interactive website, you'll need to use JavaScript, and you'll probably deal with asynchronicity in one form or another.

What's a framework? Do I need to use trendy.js?

Key terms: React, Angular, Ember, Backbone, jQuery, Underscore, Lodash

"Framework" is a word with lots of meanings. The goal of a JavaScript framework is usually to reduce the amount of tedious work required to build an interactive webpage. Frameworks are basically scaffolding, designed to solve a particular kind of problem.

Many currently-popular frameworks are designed to address the problem of, "How do I create a single-page web application that supports complex user interactions, and manages all of my business logic on the frontend?" **Single-page applications**, or **SPAs**, are web applications that don't require a page refresh, where much of the product exists as a single "page" -- think about the Facebook homepage, or your Gmail inbox.

Related Vendor Content



Threat modeling, incident command, & DDoS mitigation



Free E-Book - Achieving Continuous Delivery for Databases



Microservice Databases: Migrating from Relational Monolith to Distributed Data (By O'Reilly)



Agile Suitability Filters: A Discussion of Suitability Models, Merits and Shortcomings



How to Create and Consume your First Predictive Model

Related Sponsor



Fastly's edge cloud platform powers secure, fast and reliable online experiences for the world's most popular digital businesses. [See for yourself.](#)

So which framework should you use? React? Angular? Ember? Do you even need a framework? Cue the choice paralysis!

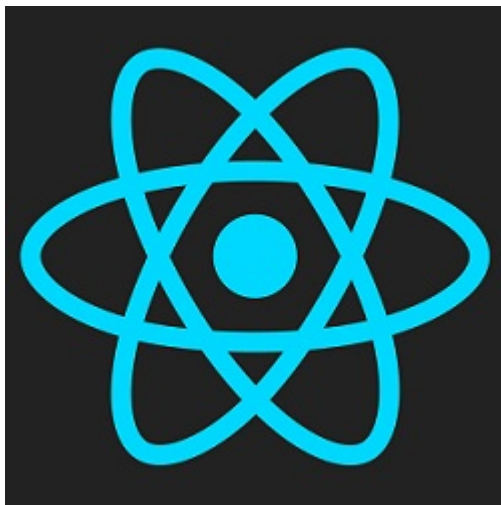
All of these projects are trying to help you write better web applications. There is no correct answer to which framework, if any, you should use.

If you're just getting started with JavaScript it might be better to use **no framework at all**, and try building a site with jQuery and little else. That will quickly become tedious, but you can do it, and it will teach you important things about how JavaScript works. [You can still practice good software engineering principles with jQuery.](#)

If you are working on a reasonably complex site, you'll probably find a framework helpful. Right now, [Angular](#), [React](#), and [Ember](#) are all popular and reasonable choices. [Backbone](#) is an older-style framework, and smaller in scope; it's also appropriate for many projects. The [starter kit](#) I've put together for this article uses React, but really, there's no wrong choice. To see for yourself how various frameworks compare, check out [TodoMVC](#), which implements the same checklist application using different frameworks.

JavaScript also lacks many standard library functions that other languages have built-in, like [padding strings](#) or [shuffling an array](#). Because of this, libraries such as [jQuery](#), [Underscore](#), and [Lodash](#) are often used to fill in the gaps. These libraries are conventionally imported and referred to using \$, _, and _ respectively, so if you see lots of dollar signs in a JavaScript file, that's almost certainly invoking jQuery.

If you're starting a new project, I recommend React or Angular, along with Lodash.



Should I be writing JavaScript, or something else? What kinds of JavaScript exist?

Key terms: *ES5, ES6, ES2015, CoffeeScript, TypeScript, ClojureScript, Babel, transpiling, compiling, MDN reference*

"JavaScript" is not really a single language. Each browser vendor implements their own JavaScript engine, and due to variations between browsers and versions, JavaScript suffers from some serious fragmentation. [CanIUse.com](#) documents some of these inconsistencies; you can also check the [Mozilla Developer Network documentation](#).

ES6, also known as **ES2015 / Harmony / ECMAScript 6 / ECMAScript 2015**, is the most-recent version of the JavaScript specification. It introduced new syntax and functionality. Fat arrows, ES6 classes, native modules, and template strings are all part of this version of JavaScript. [Treehouse has a good primer on ES6.](#)

Despite the fragmented environment in which JavaScript runs, it's nice to be able to use new language features. Thus, tools such as **Babel** will transform your shiny, standardized JavaScript into a version that's compatible with older platforms. This process is called **transpiling**. It's not much different from **compiling**. By using a tool like Babel, you don't need to worry as much about the headaches of whether or not a given browser will support the JavaScript feature you're using.

Transpiler tools don't just convert ES6 JavaScript to ES5. There are also tools to do the same thing for JavaScript-variants, such as **ClojureScript**, **TypeScript**, and **CoffeeScript**.

[ClojureScript](#) is a version of Clojure that compiles down to JavaScript. [TypeScript](#) is essentially JavaScript, but with a type system. [CoffeeScript](#) is very similar to JavaScript, but with shinier syntax; much of the syntactic sugar promoted by CoffeeScript has been adopted by ES6, rendering CoffeeScript significantly less useful. All of these compile down to regular JavaScript.

So what should you use? If you're new to frontend development, you should probably write ES6-style JavaScript. Most ES6 features have broad support across browser vendors. If you need to support older platforms, tools like Babel will compile your ES6 into ES5-compliant JavaScript for you. Stay away from shiny compile-to-JavaScript options such as ClojureScript for now, though once you have more frontend development experience, they're certainly worth exploring.



How do I use other people's code?

Key terms: *AMD, commonJS modules, ES6 modules, npm, Github*

The history of code sharing and modules in JavaScript is a bit complicated. I highly recommend reading Preethi Kasireddy's [JavaScript Modules: A Beginner's Guide](#) for more information.

For our purposes, the terms **module** and **library** are basically equivalent: they represent a chunk of self-contained code that we can use and re-use in our own projects. JavaScript modules are usually published via [npm](#), the node package manager. You can search for JavaScript modules on npm or on [GitHub](#).

There are a few competing ways of defining a module (sometimes referred to as module syntax). The main ones are **CommonJS**, **AMD**, and **ES6 Native Modules**. CommonJS takes a synchronous, server-oriented approach. By contrast, AMD (Asynchronous Module Definition) allows you to define and load modules in an asynchronous, non-blocking manner. CommonJS and AMD were both created in a world where JavaScript didn't natively support any concept of modules or dependencies.

ES6 added support for native JavaScript modules, which support both the declarative syntax of CommonJS and the asynchronous loading of AMD, among other features. Finally, modules are part of the actual language.

You'll likely come across all three types of modules in your work. For new projects, you should use ES6 native modules. Build tools, such as webpack (see below), can be helpful for working

with various types of modules in existing projects.

Do I need Node.js?

Key terms: *Node.js, npm, nvm*

Node.js is a tool for writing server-side JavaScript. Wait, but weren't we only talking about frontend JavaScript?

Because JavaScript modules are usually packaged and shared via **npm**, the node package manager, you'll want to have Node.js installed even if you won't be using it for server-side development. The best way to do this on OS X or Linux-based systems is via **nvm**, the [Node Version Manager](#), which facilitates managing different version of Node.js. Windows users should look at [nvm-windows](#).

What are my build tools?

Key terms: *grunt, gulp, bower, browserify, webpack, hot reloading, sourcemap*

Requiring each JavaScript dependency as part of your page, script tag by script tag, is slow. Therefore, most sites use so-called JavaScript **bundles**. The bundling process takes all of your dependencies and "bundles" them together into a single file for inclusion on your page. Optionally, some developers also use a **minifying** step, where all unnecessary characters are stripped out of your JavaScript without changing its functionality. This reduces the amount of data that the client will have to download.

Some tools also support features such as **hot reloading**, which will live-update your project in your browser when you save a file; and **sourcemaps**, which make debugging easier by providing a mapping from your bundled JavaScript back to its original form.

What I've just described is, essentially, a build process. Whether or not most JavaScript developers describe it this way, you're compiling your code into a production-ready format. "Front-end devops," or the process of managing your build and deployment tools and dependencies, is an increasingly complex endeavor.

Grunt, gulp, broccoli, brunch, browserify, and webpack are all tools for JavaScript builds. Comparing them is difficult, because they each focus on solving different problems. Many of them use different abstractions to talk about the same problems, and we don't really have a shared base of jargon yet.

In my experience, the configurations for these tools are often poorly understood, and thus get copy-pasted around between projects. For reference, here's the webpack configuration I put together for the starter repo:

```
var webpack = require('webpack');
module.exports = {
  entry: [
    './app.js'
  ],
  output: {
    path: __dirname + '/static',
    filename: 'bundle.js'
  },
  module: {
```

```
loaders: [
  {
    test: /\.js?$/,
    loader: 'babel-loader',
    query: {
      presets: ['es2015', 'react']
    },
    exclude: /node_modules/
  }
],
plugins: [
]
};
```

All told, this webpack configuration instructs webpack to:

- Start with app.js as the entry point
- Process all files ending in .js
- Use babel-loader to transform them, specifically handling ES6 transpilation (hence the es2015 query) and JSX (hence the react query)
- Bundle all the JavaScript into a file located in static/bundle.js

For new projects, I recommend **webpack**. It has strong adoption and handles large projects with complex dependency graphs well.



How do I test my code?

Key terms: *Mocha, Jasmine, Chai, Tape, Karma, Selenium, phantomjs*

Much as with any other kind of programming, your frontend JavaScript can [benefit](#) from testing. [Most](#) JavaScript developers say that they write at least some tests.

JavaScript lacks a built-in framework for running tests, so developers depend on external libraries. Much like JavaScript build systems, different testing tools focus on different aspects of the problem.

Mocha and **Jasmine** are two popular libraries, sometimes referred to as **testing frameworks**, that help you write tests. Their APIs are very similar; you describe how something ought to behave, then use assertions to test it.

```
describe('helloWorld()', function() {
  it('should greet me by name', function() {
```

```
// assertions go here
});
});
```

Mocha doesn't actually come with a built-in assertion library, so most developers combine it with **Chai**. Their assertion syntax is similar:

```
// Jasmine
expect(helloWorld("Bonnie")).toEqual("Hello, Bonnie");

// Chai
expect(helloWorld("Bonnie")).to.equal("Hello, Bonnie");
```

For running your tests, Mocha provides a command-line utility, while Jasmine does not. Many developers use **Karma**, which is a **test runner** that can run both Jasmine and Mocha-style tests.

That's all well and good for unit tests; for JavaScript-based integration tests, we'll need more tools. In frontend terms, an integration test often involves using a browser to actually render and load the page, simulating user interactions, and checking the result.

Selenium is a web driver that is often used for these tests. You'll need to equip Selenium with a **browser driver**, so that it can launch the browser. **PhantomJS** is a so-called **headless browser** -- it runs without a GUI -- that is often used in testing. Because they don't require a GUI, headless browsers are useful for automated tests. You may also find **Sinon** helpful; it provides a fake server, which can be useful for faking responses to AJAX requests.

You can also set up your JavaScript tests to run with your **continuous integration (CI)** system, such as **Jenkins** or **Travis**.

There are plenty of perfectly good choices for JavaScript testing tools. For new projects I typically choose Karma and Jasmine, with PhantomJS as my test browser, but Mocha + Chai is also a good choice.



So how do I get started?

I've pulled together a Github repo containing a basic frontend development setup here:

<https://github.com/bonniee/react-starter>

It uses:

- React
- Babel for transpilation
- Webpack for builds
- ES6 syntax (for React classes, and module exports)
- Karma + Jasmine + PhantomJS for tests

Let's break this down a bit more. React is the framework we're using, which makes it easier to build interactive websites. (You describe your UI, and React handles the rendering and DOM manipulation for you.) We'll write JavaScript in accordance with the ES6 specification. Webpack will use Babel to transpile our ES6 JavaScript code into ES5-compliant JavaScript; Webpack also manages our dependencies and module imports. Finally, we use Karma and PhantomJS to run our tests, and the Jasmine library to write them.



First, make sure you have a working version of [npm](#). Then, to install your dependencies for this repository and get started:

```
npm install
```

```
webpack
```

Then, to develop with it, run:

```
webpack --watch
```

This will instruct webpack to watch your project and recompile it when your JavaScript files change. To view your application, you'll need to launch a local server. On OS X or Linux you can use Python to do this easily:

```
python -m SimpleHTTPServer
```

...and you're off to the races! Make edits to `app.js` or `Hello.js` to add more React components, and use `npm test` to run your tests.

Of course, having a working repository of starter code is just half the battle. The world of frontend JavaScript development can be complicated, and there's a proliferation of tools and terminology, as well as new concepts to learn and problems to solve. Any one of the topics above could easily fill an entire blog post. Hopefully this article has helped illuminate some of the JavaScript landscape, and will help guide you as you learn more about frontend development.

Welcome to the community!

Note: Logo licenses

1. [React](#): Licensed under [React-docs license](#), Creative-Commons attribution
2. [Babel](#): [MIT license](#)

3. Webpack: [MIT license](#)
4. Karma: [MIT license](#)
5. Jasmine: [MIT license](#)
6. PhantomJS: [BSD license](#)






About the Author



Bonnie Eisenman is a software engineer at Twitter and a member of the hackerspace NYC Resistor, with previous experience at Codecademy, Fog Creek Software, and Google. She is the author of *Learning React Native*, a book on building native iOS and Android applications with Javascript, published with O'Reilly Media. She has spoken at several conferences on topics ranging from ReactJS, to musical programming and Arduinos. In her spare time, she enjoys learning languages, tinkering with hardware projects, and laser-cutting chocolate. Find her on Twitter as [@brindelle](#).

[Personas](#)[Development](#)[Topics](#)[React](#)[JavaScript Libraries](#)[JavaScript](#)[Agile Techniques](#)[Build systems](#)[Continuous Integration](#)[Agile](#)[Reactive Programming](#)[Web Development](#)[Dynamic Languages](#)

Related Editorial

-  Aurelia JavaScript Framework Hits 1.0, Looks to the Future
-  D3 JavaScript Visualization Library Hits 4.0
-  Ashley Nolan Surveys State of JavaScript Tooling in 2016
-  Pinterest's Switch to Universal JavaScript and React
-  Next.js Offers Simple Universal JavaScript Framework Based on React

Tell us what you think

Please enter a subject

Message

Post Message

Community comments

 Watch Thread

- ▶ **Thanks very much!** by Ilya Palopezhentsev Posted Apr 12, 2016 06:31
- ▶ **JavaScript Metastasis: A Warning for Newcomers** by Richard Eng Posted Apr 12, 2016 06:40
 - ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Will Hartung Posted Apr 13, 2016 11:00
 - ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Richard Eng Posted Apr 14, 2016 07:26
 - ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Richard Clayton Posted Apr 18, 2016 09:08

- ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Richard Eng Posted Apr 25, 2016 05:50
- ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Richard Clayton Posted Apr 25, 2016 06:52
- ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Bonnie Eisenman Posted Jan 10, 2017 12:20
- ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Jean-Jacques Dubray Posted May 19, 2016 04:32
- ▶ **Re: JavaScript Metastasis: A Warning for Newcomers** by Alex Bruno Cáceres Posted Jan 05, 2017 10:02
- ▶ **Great read!** by Sarah Alhawi Posted Apr 13, 2016 03:23
- ▶ **Fantastic Article!** by Chris Speer Posted Apr 13, 2016 10:50
- ▶ **really good article for overlook all javascript things.** by Uk Jo Posted Apr 15, 2016 12:53
- ▶ **A Complete Map Indeed!** by Shiva YB Posted Apr 20, 2016 12:15
- ▶ **Excellent timing - well written** by Guillaume Bilodeau Posted Apr 21, 2016 08:19
- ▶ **Terrific** by Immo Huneke Posted May 10, 2016 02:41
- ▶ **Javascript Ninja in Process.** by Chris Tian Posted Jun 07, 2016 02:33
- ▶ **Great resource** by Carlos Montoya Posted Aug 07, 2016 11:59
- ▶ **Thank you!** by Anup Tripathi Posted Aug 25, 2016 11:26
- ▶ **Ember.js is more ideal for corporate projects than Angular or React** by Zoltan Debre Posted Oct 12, 2016 09:55
- ▶ **Re: Ember.js is more ideal for corporate projects than Angular or React** by Bonnie Eisenman Posted Jan 10, 2017 12:24

Thanks very much!

Apr 12, 2016 06:31 by [Ilya Palopezhentsev](#)

It was very nice introduction for me! It was very hard for me to find all the disjoint bits of modern java script development and you've summarised it excellently.

 Like  Reply  Back to top

JavaScript Metastasis: A Warning for Newcomers

Apr 12, 2016 06:40 by [Richard Eng](#)

I really don't think you should recommend JavaScript. This is very much a dysfunctional programming language. I argue vociferously for using transpiled languages.

The JS web framework landscape is a huge mess. These frameworks make client-side web development far more complicated than is necessary. Using the transpiled languages that I recommend, jQuery is all you need to write large, maintainable applications that do everything you want to do, including all the fancy UI stuff you crave. And do it more easily without all the complications.

I highly encourage everyone to support fixing the JavaScript language before it's too late.

 Like  Reply  Back to top

Great read!

Apr 13, 2016 03:23 by [Sarah Alhawi](#)

This is really helpful - shows how all the tools and frameworks fit in together and their purpose.

 Like  Reply  Back to top

Fantastic Article!

Apr 13, 2016 10:50 by [Chris Speer](#)

Very helpful advice on navigating the current Javascript ecosystem. Thanks!

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

Apr 13, 2016 11:00 by [Will Hartung](#)

Then, Richard, you should do what Bonnie did and promote a quick start article and toolset to encourage adopting.

The curse of JS right now is variety ("choice paralysis", which is VERY real), and complexity. The integration of all the tools makes the JS landscape very difficult to navigate for the novice.

The transpiled languages do not reduce this complexity. So, having a package of start up tooling that lets folks wade in to the waters slowly is important to promoting adoption.

There is a lot of baby duck syndrome in development. Since most of these platforms mostly do the same thing in mostly similar ways, once a developer has adopted one, and gained momentum and experience in the platform, getting them to change their habit is much more difficult.

So, getting folks started early is better, and the easier it is for people to start, the more people will start.

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

Apr 14, 2016 07:26 by [Richard Eng](#)

Actually, I did write [a tutorial article series](#) for my favourite transpiled language: Amber Smalltalk. There are other good language choices as well, such as ClojureScript and Scala.js. Unfortunately, there is no one tutorial that can cover multiple languages; each has its own

way of doing things. But if everyone followed Amber Smalltalk, that wouldn't be such a bad thing! ;-)

Regarding choice paralysis, all the transpiled languages use jQuery, so that pretty much removes the paralysis with regards to web frameworks **and their associated toolsets**. I never really understood the need to go beyond jQuery, which is a very solid and well-supported framework that is widely used.

 Like  Reply  Back to top

really good article for overlook all javascript things.

Apr 15, 2016 12:53 by [Uk Jo](#)

one of best materials that I have ever seen!!

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

Apr 18, 2016 09:08 by [Richard Clayton](#)

All you need is jQuery? Probably the most absurd thing I've heard in a long time, particularly as a suggestion for newcomers.

 Like  Reply  Back to top

A Complete Map Indeed!

Apr 20, 2016 12:15 by [Shiva YB](#)

One of the best orientations for any newcomer to Front end development. I don't think anything is left out of this short article. Equips the developer with all the tools needed. Thanks very much!

 Like  Reply  Back to top

Excellent timing - well written

Apr 21, 2016 08:19 by [Guillaume Bilodeau](#)

This article could not have come at a better time. A long-time back-end Java developer, I have undertaken some front-end development tasks the day this article was published. It gave me clear pointers to the latest technologies and how they fit together, and allowed me to make some clear design decisions. Thank you!

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

Apr 25, 2016 05:50 by [Richard Eng](#)

Why is using jQuery absurd? It's the most widely used JS web framework in the world. It's very solid and extremely well-supported with tons of plugins. By using transpiled languages,

such as [these](#), you can do all the front-end stuff that you do with Angular, Backbone, React, Ember, Express, Meteor, Polymer, Knockout, etc. Transpiled languages universally rely on jQuery. Check out the super-cool examples I provide for ClojureScript, Dart, Haxe, and Scala.js.

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

Apr 25, 2016 06:52 by [Richard Clayton](#)

Why? Because it provides absolutely no structure to the application. I'm not saying that jQuery is bad, I'm saying that advocating that a beginner should take on a huge web app with only a utility library is irresponsible.

I don't even know why I'm arguing this. The reason all those other frameworks exist is to provide the features/structure jQuery doesn't offer (routing, templating/views, controllers and/or reducers, etc.).

 Like  Reply  Back to top

Terrific

May 10, 2016 02:41 by [Immo Huneke](#)

Having worked on various projects that involve front-end development (mainly doing the REST services and/or UI testing) it's refreshing to finally get a bird's eye view of the industrial landscape I have been trying to find my way around. Every corner I turn seems to reveal some new edifice of unknown function - it's really helpful to discover which ones are useful and complementary.

One thing I've noticed across the various projects is that if you use grunt or gulp (or, even worse, both) you still need npm anyway. With a bit of ingenuity, you can run all parts of your build using just npm, which simplifies and speeds up your builds including test runs.

The other thing that this article should definitely mention is the Q promise library by Kris Kowal (github.com/kris/kowal/q) - almost all web applications of any size rely on this (or Angular's equivalent) to make the asynchronous programming model tractable.

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

May 19, 2016 04:32 by [Jean-Jacques Dubray](#)

I would argue pretty much the opposite, JavaScript does require that you to be more opinionated about which programming model, wiring and architecture you choose (and stick to), but choosing for instance an Object Oriented facet of the language would probably result in losing 80% of the value JavaScript brings over Java.

I am obviously biased, but if you chose to learn JavaScript I would start learning the [SAM Pattern](#), or at a minimum I would chose a programming model where the View is a pure

function of the Model ($V = f(M)$) and the model is a single state tree (with Time Travel debugging tools). That's the future of Front-End architectures, and that's where you need start.

There are a lot of plain vanilla (and Framework based) samples on SAM's home page, for instance the TODOMVC using both [ES5 and ES6](#). Plain vanilla implementations are generally isomorphic and run nearly as is.

Last but not least, I would also take a look at AWS Lambda.

 Like  Reply  Back to top

Javascript Ninja in Process.

Jun 07, 2016 02:33 by [Chris Tian](#)

Thank you for posting this great article. Sure, wish I would have read this a few months ago, would have come in handy!!

 Like  Reply  Back to top

Great resource

Aug 07, 2016 11:59 by [Carlos Montoya](#)

Thank you so much for doing this, very valuable to have a big picture !

 Like  Reply  Back to top

Thank you!

Aug 25, 2016 11:26 by [Anup Tripathi](#)

Loved reading this in spite of understanding the landscape a little bit after a year being a full stack developer. Concise and comprehensive. Still working to understand whether to choose React over angular 2 or stick with trusty old jquery. Glad to see the choice paralysis is recognized by so many.

Trying to learn these, after working on jQuery for some length of time, made me think do we really need these frameworks? I tell myself that smarter people than me made these so must be for good reason. But still, doesn't make the adoption less painful. :P

 Like  Reply  Back to top

Ember.js is more ideal for corporate projects than Angular or React

Oct 12, 2016 09:55 by [Zoltan Debre](#)

Thank you for this great article. One thing, which is quite important. So many infoQ readers are from corporate area where a matured, with long term support, convention over configuration solution are more preferred. In this area Ember.js is the best choice, especially if the team is already knows some other MVC framework, and prefer OO patterns and clean code. Ember.js provide this on frontend world.

[Ember.js](#) has a huge ecosystem, thousands of addons, long term support releases, it is fully open source, and it is improved based on the real user requirement and not based on Google or Facebook requirement. Because of the conventions, development is super efficient and fast with Ember.js. Onboarding is minimal, because if someone already used Ember, they can join to any Ember project in a few hours.

It is easy to learn, the community built free, detailed [ember tutorials](#) also.

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

Jan 05, 2017 10:02 by [Alex Bruno Cáceres](#)

Sorry, but all languages are "transpiled"!

If you hate "transpiled" languages, then you can only develop in Assembly or binary.

 Like  Reply  Back to top

Re: JavaScript Metastasis: A Warning for Newcomers

Jan 10, 2017 12:20 by [Bonnie Eisenman](#)

You're right - those are important features/structures. However, for someone's *first* application, I think there's something to be learned from doing frontend development without the added complexity of a framework. It teaches you why you need those features.

I wouldn't advocate that a beginner build and maintain a "huge web app with only a utility library" in order to run a robust production app, but I think it's an instructive experience.

 Like  Reply  Back to top

Re: Ember.js is more ideal for corporate projects than Angular or React

Jan 10, 2017 12:24 by [Bonnie Eisenman](#)

That depends on your definition of "corporate" - by some definitions Google and Facebook are plenty "corporate". I don't have extensive hands-on experience with Ember, but my impression is that you're right, and for situations where you want convention over configuration it's a great choice.

 Like  Reply  Back to top