The Rich History of JavaScript

By Christian Gaetano

JavaScript wasn't always the grand language used to build massive framework systems that it is today. For a long time following its inception, JavaScript was mostly used for gimmicky website effects, like firework animations. It's come a long way from its archaic beginnings. The best way to see this dramatic, if gradual, improvement is to look at the ECMAScript standardization of JavaScript.

If you know what JavaScript is, then you also know what ECMAScript is. These two titles both refer to the same programming language. The colloquial name "JavaScript" is a strategic misnomer. Even though JavaScript syntax bears some resemblance to Java, the languages vary widely on their core principles. **Brendan Eich**, a former Netscape employee credited with creating JavaScript in 1995, coined the name JavaScript due to Java's immense popularity at the time. *Without this marketing ploy, JavaScript may not have been adopted by the community at large.* On the other hand, that original name has left web developers with a confusing conundrum in modern-day usage of the language. Even though "JavaScript" is now universally recognized, its etymology often remains a mystery.

> *If you know what JavaScript is, then you also know what ECMAScript is.*

The European Computer Manufacturers Association

Shortly after JavaScript's creation, the European Computer Manufacturers Association (ECMA), which puts forth standards for many modern technological protocols and programming languages, was tasked with standardizing the language. From this effort was borne the ECMAScript (ES) specification. Although related, ECMAScript and JavaScript are not synonymous. JavaScript is an *implementation* of the ECMAScript specification. (Other implementations of the ES specification exist, though they're used much less widely than JavaScript.) Nonetheless, because of its widespread usage, JavaScript is the "poster child" of ECMAScript. Generally, and especially in this e-book, any time you see a specific ECMAScript standard revision

mentioned, you can think of that as "how JavaScript implements this ECMAScript standard revision."

The recently-finalized ES2015 standard is still undergoing adoption by most major browsers and JavaScript engines. As expected, adoption is occurring feature-by-feature rather than holistically. While this can muddle compatibility issues, the upside is that it allows us to see that JavaScript is now far separated from its original identity. No longer a simple gimmick of a programming language, JavaScript has adapted to address a major issue that has fueled debate over its usefulness: eloquence. With new features like block level scoping and generator functions shipping in ES2015, JavaScript can now hold its own among the traditional "refined" programming languages like Java and C#. Impressively, JavaScript is coming into this role while retaining its usefulness as a procedural, customizable language as well. All this evolution culminates to offer a platform that is powerful and useful, not only for the client-side web, but also for server-side and native apps.

Understanding this history is important to understanding the current boom in JavaScript usage. The changes included in the recent ES2015 standardization and the advent of other new technologies—like local storage through the browser and web sockets—have been especially important to enabling the "Dawn of Frameworks" that necessitated this e-book.

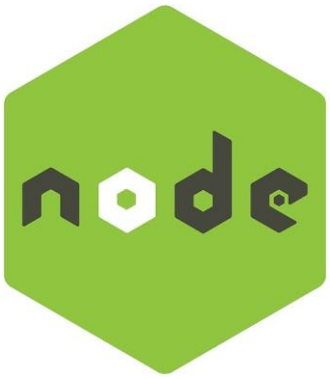
Where is JavaScript Now?

Although many have tried, it's difficult to generate reliable statistics on exactly what fraction of internet users have run some form of JavaScript. Fortunately, all modern web browsers support and enable JavaScript out of the box. Even browsers that ship with smartphones have JavaScript support that mirrors or outdoes their desktop counterparts. So pretty much *anyone* who has a semi-modern device has access to JavaScript content on the web. Even if you look back several years, you'll find that the majority of web browsers shipped with JavaScript support enabled by default. Even if this support is not complete (due to compatibility issues with new syntax, etc.), most features can be polyfilled or shimmed to make them work in older browsers. Overall, estimates put JavaScript accessibility near 100% of internet users. (This doesn't account for the relatively small group of people who voluntarily turn off JavaScript.) I'll be the first to admit that we, as developers, have a responsibility to make content accessible to *everyone* who uses the internet. Luckily, most of the major JavaScript frameworks provide a check and fallback mechanism for users who can't access JavaScript-driven apps. As a developer, you're still responsible for providing static fallback content, but even that can be generated using JavaScript on the server.

The major takeaway here is that putting your money on JavaScript support when deciding to use a

framework is a pretty safe bet to make. You can count on reaching virtually all internet users, and you can still use JavaScript to render a static fallback if you'd like.

The idea of using JavaScript to generate and serve static content brings up another interesting topic: server-side code. **With the advent of Node.js,** server-side JavaScript has become mainstream and commonplace. Node provides a powerful platform



for writing reliable and fast server-side applications using JavaScript, and its scalability has made it a viable option for large and frequently used server-side APIs. While it may not be obvious, this is yet another reason to use JavaScript frameworks. The availability of Node.js brings with it the possibility of a unified codebase. You can potentially integrate your entire development stack with the JavaScript language, building a backend with Node.js and a front-end with a JavaScript framework, to improve workflow and team communication.

Node.js has also introduced the web development world to **Node Package Manager** (NPM), which further facilitates JavaScript development.

With thousands of handy pluggable modules at your fingertips, the ability to share package configurations for a project, and an easy install process, NPM has revolutionized the development world. In the scope of frameworks, NPM makes starting up a new project and adding any popular framework a breeze. In fact, the "Installation" pages of almost all major JavaScript frameworks begin immediately with an npm install instruction.

The combination of widespread browser support, server-side platforms, and package management solutions makes *now* the best time to start using a JavaScript framework. As a matter of fact, some recent surveys indicate that over 70% of JavaScript developers have already used a front-end framework, and over 95% have at least *heard of* one of the frameworks discussed in this e-book. And if everyone else is using them, there must be something to this whole framework thing, right?

The widespread usage of JavaScript frameworks not only indicates that they work well; it also means that a vast array of resources and plugins have been vetted and improved by the large JavaScript community. It always feels better to start something new when you know others have done it before you. So rest assured: many have traveled and survived the path to framework enlightenment on which you're about to embark, and they've left some helpful tools and tips along the way.