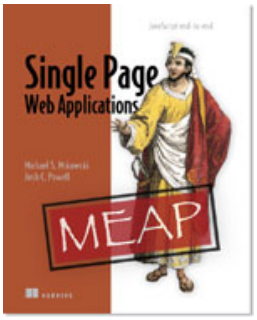
MANNING PUBLICATIONS



Single Page Web Applications
By Michael S. Mikowski and Josh C. Powell

This green paper based on Single Page Web Applications, defines Single Page Applications (SPAs) and explains user benefits of a well-written SPA.

You may also be interested in…

Single Page Web Applications

A Single Page Application (SPA) is an application delivered to the browser that does not reload the page during use. Like all applications, it is intended to help the user complete a task, like "write a document" or "administer a web server." We can think of an SPA as a fat client that is loaded from a web server.

A little history

SPAs have been around for a long time. Let's look at some early examples:

- *Javatic-tac-toe*—http://java.sun.com/applets/jdk/1.4/demo/applets/TicTacToe/example1.html. Hey, we didn't say this would be pretty. The task is to play a game of tic-tac-toe against your formidable and ruthless computer nemesis. The Java plugin is required (see http://www.java.com/en/download/index.jsp).
- *Flash Spacelander*—http://games.whomwah.com/spacelander.html. This is one of the earlier Flash games, written by Duncan Robertson circa 2001. The Flash plugin is required (see http://get.adobe.com/flashplayer/).
- *JavaScript mortgage calculator*—http://www.mcfedries.com/creatingawebpage/mortgage.htm. This calculator seems almost as old as JavaScript itself, but it works nicely. *No plugin is required.*

The astute reader—and even a few slovenly ones[1]—will notice that we have provided examples of three of the most popular SPA platforms: Java applets, Flash/Flex, and JavaScript. And those same readers may have noticed that *only the* JavaScript SPA works without the headaches and security concerns of a third-party plugin.

Today, JavaScript SPA's are the often the best choice of the three. But they took a while to become compelling, or even possible, for most uses. Let's take a look at why.

What took JavaScript SPAs so long?

Flash and Java applets had evolved nicely by the year 2000. Java was being used to deliver complex applications and even a complete office suite via the browser.[2] Flash had become the platform of choice for delivering rich browser games and, later, video. On the other hand, JavaScript was still pretty much relegated to little more than mortgage calculators, form validation, rollover effects, and pop-up windows. The problem was that we couldn't rely on JavaScript (or the rendering methods it used) to provide critical capabilities consistently on popular browsers.

Even so, JavaScript SPAs promised a number of enticing advantages over Flash and Java:

- *No plugin required*—Users access the application without concern for plugin installation, maintenance, and

[1] If you are currently eating potato chips off your chest, you are slovenly.
[2] Applix (VistaSource) Anywhere Office

OS compatibility. There is also no need to worry about a separate security model, which reduces development and maintenance headaches.[3]

- *Less bloat*—An SPA using JavaScript and HTML should use significantly fewer resources than a plugin that requires an additional runtime environment.
- *One client language*—Web architects and most developers have to know many languages and data formats—HTML, CSS, JSON, XML, JavaScript, SQL, PHP/Java/Ruby/Perl, and so on. Using a single language and environment for the browser is a great way to reduce complexity.
- *A more fluid and interactive page*—We have all seen a Flash or Java application on a web page. Often the application is displayed in a box somewhere and many details are different than the HTML elements that surround it: the graphical widgets are different, the right-click is different, the sounds are different, and interaction with the rest of the page is limited. With a JavaScript SPA, the entire browser window is the application interface.

As JavaScript has matured most of its weaknesses have been either fixed or mitigated and its advantages have increased in value:

- *The web browser is the world's most widely used application*—Many people always have a browser open and use it throughout the day. Access to a JavaScript application is just one bookmark click away.
- *JavaScript in the browser is one of the world's most widely distributed execution environments*—By December 2011, nearly a million Android and iOS mobile devices were activated every day. Each of these devices has a robust JavaScript execution environment built into the OS. Over a billion robust JavaScript implementations have shipped in the last three years on phone, tablet, laptop, and desktop computers around the world.
- *Deployment of JavaScript is trivial*—A JavaScript application can be made available to over one billion web users by hosting it on an HTTP server.
- *JavaScript is now very useful for cross-platform development*—We can thank converging implementation of standards across browsers, and mature libraries such as jQuery and PhoneGap that smooth over inconsistencies.
- *JavaScript has become surprisingly fast and can, at times, rival compiled languages*—This is thanks to the ongoing and heated competition between Mozilla Firefox, Google Chrome, Opera, and Microsoft. Modern JavaScript implementations enjoy advanced optimizations such as JIT compilation to native machine code, branch prediction, type-inference, and multi-threading.[4]
- *JavaScript has evolved to include advanced features like the JSON native object, native jQuery-style selectors, and more consistent AJAX capabilities*—Push messaging has become far easier with mature libraries like Strophie and Socket.IO.
- *HTML5, SVG, and CSS3 standards and support have advanced to the point where the rendering of pixel perfect graphics can rival the speed and quality produced by Java or Flash*.
- *JavaScript can now be used throughout a web project by using the excellent Node.js web server and data stores such as CouchDB or MongoDB which communicate in JSON, a JavaScript data format*—You can even share libraries between the server and the browser.
- *Desktop, laptop, and even mobile devices have become very powerful, with multicore processors and gigabytes of RAM*—Processing that used to be accomplished on the server can now be distributed to the client browsers.

JavaScript SPAs are becoming increasingly popular due to these advantages, and the demand for experienced JavaScript developers and architects has blossomed. Applications that were once developed for many operating systems (or for Java or Flash) are now delivered as a single JavaScript SPA. Startups have embraced Node.js as

[3] Can you say "same origin policy"? If you ever developed in Flash or Java, you almost certainly are familiar with this challenge.
[4] See http://iq12.com/blog/as3-benchmark/ and http://jacksondunstan.com/articles/1636 for some comparisons to Flash ActionScript 3

the web server of choice, and mobile application developers are using JavaScript and PhoneGap to create "native" applications for multiple mobile platforms using a single code base.

JavaScript is not perfect, and you don't have to look far to find omissions, inconsistencies, and other aspects to dislike. But this is true of all languages. Once you become comfortable with its core concepts, employ best practices, and learn what parts to avoid, JavaScript development can be pleasant and productive.

One destination, two paths

There are two common methods of developing a JavaScript SPA. JavaScript applications are written directly in *Native* JavaScript by the development team. *Generated* JavaScript applications are written in another language which is then converted to JavaScript, either at runtime or during a separate generation stage. We make this distinction because *generated* JavaScript is surprisingly popular.

Notable JavaScript generators include:

- *Google Web Toolkit (GWT)*—See http://code.google.com/webtoolkit/. GWT generates JavaScript from Java.
- *Cappuccino*—See http://cappuccino.org/. Cappuccino uses ObjectJ, a clone of the Object-C language from Mac OSX. Cappuccino itself is a port of the Cocoa application framework, again from OSX.
- *CoffeeScript*—See http://coffeescript.org/. CoffeeScript turns a custom language that provides some syntactic sugar into JavaScript.

Given that Google uses GWT for Gmail and other SPAs, we can safely say that generated JavaScript SPAs are quite widely used. This of course begs the question, why bother writing in one high-level language and then converting it to another? There are at least a few reasons, although most are not as compelling as they once were:

- *Familiarity*—The developers are already familiar with another language, and the generator and framework allows them to develop without having to learn the vagaries of JavaScript. The problem we see is that something eventually gets lost in translation. When this happens one has to inspect the generated JavaScript and understand it to get things to work right. We are more effective when we work directly in JavaScript instead of working through multiple levels of language abstraction.
- *Framework*—The generator and framework provide a cohesive system with matching libraries built for server and client. This is probably the strongest argument for using something like GWT.
- *Multiple targets*—Generator can write for multiple targets, such as one file for the Internet Explorer and one for the rest of the world's browsers. While generating code for different targets sounds nice, we think it's even more effective to deploy a single JavaScript source for all browsers. Thanks to converging browser implementations and mature cross-browser libraries like jQuery, it is now much easier to write a sophisticated SPA that runs across all major browsers without modification.
- *Maturity*—The developers consider JavaScript not sufficiently structured for large-scale application development. Yet JavaScript has evolved to become a much better language, with impressive strengths and manageable weaknesses. Developers from strongly typed languages like Java sometimes feel the lack of type-safety is unforgivable. And some developers from inclusive frameworks like Ruby on Rails bemoan the apparent lack of structure. Thankfully, we can mitigate these issues through a combination of code validation tools, code standards, and the use of mature libraries.

We believe JavaScript SPA's are usually the *native* best choice today.

Let's consider the primary benefit of an SPA over a traditional website: it provides a substantially more engaging user experience.

The user benefits of a well-written SPA

An SPA can deliver the best of both worlds—the immediacy of a desktop application *and* the portability and accessibility of a web site.

- An SPA can render like a desktop application: the SPA redraws the parts of the interface that needs to

change only as needed. A traditional web site, in comparison, redraws the entire page on every user action, resulting in a pause and a "flash" while the browser retrieves from the server and then redraws everything on the page. If the page is large, the server is busy, or the Internet connection is slow, this "flash" can take several seconds or more, and the user has to guess when the page is ready to use again. This is a horrible experience when compared to the rapid rendering of an SPA.

- An SPA can respond like a desktop application: the SPA minimizes response time by moving working ("transient") data and processing from the server to the browser as much as possible. The SPA has the data and business logic needed to make most decisions locally and therefore quickly. Only data validation, authentication, and permanent storage must remain on the server, for reasons we will introduce in later chapters. A traditional web site has most of the application logic on the server and the user must wait for a request/response/redraw cycle in response to much of their input. This can take several seconds, compared to the near immediate response of the SPA.
- An SPA can notify users of its state like a desktop application: when an SPA has to wait on a server, it can dynamically render *does* a progress bar or busy indicator so the user is not befuddled by a delay. Compare this to a traditional web site, where the user actually has to *guess* when the page is loaded and usable.
- An SPA is nearly universally accessible like a web site: unlike most desktop applications, users can access an SPA from anywhere they have a web connection and a decent browser. Today, that includes smart phones, tablets, televisions, laptops and desktop computers.
- An SPA can be instantly updated and distributed like a web site: the user *does not have to do anything* to realize the benefits—just reload the browser and it works. The hassle of maintaining multiple concurrent versions of software is largely eliminated.[5] The authors have worked on SPAs that have been built and updated multiple times in a single day. Desktop applications often require a download and administrative access to install a new version, and the interval between versions can be many months or years.
- An SPA is cross-platform like a web site: unlike most desktop applications, a well written SPA can work on any operating system that provides a decent HTML5 browser. While usually this is considered a developer benefit, it is extremely useful for many users who have a combination of devices—say Windows at work, a Mac at home, a Linux server, an Android phone and an Amazon tablet.

All of these benefits mean that you may want to make your next application an SPA. Clunky web sites that re-render an entire page after each click tend to increasingly alienate audiences as web users are become more sophisticated. The communicative and responsive interface of a well-written SPA along with the accessibility of the internet helps keep our customers where they belong—using our product.

Summary

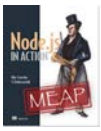
The single page application (SPA) has been around for quite some time. Flash and Java have, until recently, been the most widely used SPA platforms because their capability, speed, and consistency exceeded those of JavaScript and browser rendering. Recently, however, JavaScript and browser rendering have reached a tipping point where they have overcome their most troublesome deficiencies while providing significant advantages over other platforms.

An SPA can provide the best of both worlds—the immediacy of a desktop application *and* the portability and accessibility of a web site. The JavaScript SPA is available everywhere there is a web browser, works on desktop and mobile devices, runs on multiple OSs, and doesn't require any proprietary plugins. SPAs are easily updated and distributed, usually without requiring any action from the user. All of these benefits mean that you may want to make your next application an SPA.

[5] But not completely: what happens if the server-client data exchange format changes yet many users have the prior version of software loaded in their browser? This can be accommodated with some forethought.

Here are some other Manning titles you might be interested in:

Secrets of the JavaScript Ninja
John Resig and Bear Bibeault



Node.js in Action
Mike Cantelon and TJ Holowaychuk

CoffeeScript in Action
Patrick Lee

Last updated: July 11, 2012