Web Apps, Browsers, Servers, and the HTTP Protocol, OH MY! (A technical history lesson)

Joseph Wilm
January 19, 2014

Today we are inundated by dynamic, interactive, media-rich web applications with pixel perfect designs. Fortunes are spent each year on shaving a few kilobytes off of file size and a few milliseconds off of page load times to deliver an immersive experience across mobile, tablets, and desktop clients. Just two decades ago, the internet was a dramatically different landscape having little in common with modern web apps. Static Hypertext documents constituted the entirety of the web, HTTP only supported `GET` requests, and errors had to be inferred through the HTML or PLAINTEXT content of the response. This is the story of the web's transformation from static to dynamic and (almost) back again.

In the beginning

Prior to HTTP was Gopher - a simple protocol for distributing, searching, and retrieving documents. Invented at University of Minnesota, Gopher presents a file-like hierarchy of data in addition to being a gateway to usenet and ftp. A major strength of gopher was the ability to connect servers to share directories and resources. Alas, Gopher's rigid document structure was not flexible enough for demands of the web. Combined with licensing fees, implementations in browsers which also supported HTTP, and licensing fees from the University, Gopher quickly fell out of favor in lieu of HTTP.

The originally documented HTTP Protocol (Version 0.9, 1991) only supported `GET` requests, status codes did not exist, and content was entirely static. This was sufficient to deliver simple hypertext documents, but additional request types, and a robust set of behavior modifying switches would be needed to propel HTTP into the 21st century.

Despite the current simplicity of HTTP, script generated dynamic markup was already being served. CGI (*Common Gateway Interface*) was first being standardized by the NCSA in 1993. CGI allowed web servers to run arbitrary programs on the host server (such as a Perl or Python script). There were several issues with this strategy including overhead from creation of new processes for each request, and the ease of introducing security holes in CGI script.

A dynamic client and server

`POST`, `PUT`, `DELETE`, and other request types were introduced into browsers as early as 1992, but the world had to wait until 1996 for an official specification when the IETF released RFC 1945 *Hypertext Transfer Protocol – HTTP/1.0*. Of course, that didn't stop web browsers from implementing such features. The spec

described many features necessary to deliver today's modern application - namely two more request types, `POST` and `HEAD`, status codes, and a suite of powerful headers to enable compression, basic authentication, caching, and content-type switching.

Scripting languages were now in heavy use. FastCGI was being developed by Open Market, Inc. in the mid 90s to combat the overhead of standard CGI. Python has been available since 1991, and PHP and Ruby were released in 1995. All of these scripting languages were callable from both CGI and FastCGI. Combined with additional HTTP functionality, these scripting languages powered a dynamic web where users could both retrieve and update information on a server.

The Apache web server (first released as update to NCSA httpd in 1995) became the most popular server of the time. The modular architecture offered the ability to include interpreters and CGI handlers directly within the web server, thereby avoiding overhead costs associated with spawning additional processes for each request to generate dynamic markup. In the same year, the popular open source database *MySQL* was released. Together with PHP and Linux, these components would form the famous LAMP stack for web servers.

Primitive web applications became possible due to dynamically generated markup in concert with forward looking HTTP features. Such applications could could both retrieve and send data to the server, but every action would require a new request to the server to alter the page. *SSL* (secure sockets layer), introduced in 1995, would allow HTTP to run within an encrypted tunnel and therefore enable secure business transactions. The lack of SSL prior to then did not stop the Stanford Federal Credit Union from offering internet banking services as early as 1994.

Client side scripting was introduced in 1995 when Netscape shipped a language called *LiveScript* (known now as *JavaScript*) with their browser. The release of JavaScript was followed shortly by *Sun Java* and *Macromedia Flash*. These latter technologies were plugin based and in the long run are not able to compete with the power and security offered by JavaScript; although, flash is still widespread today - primarily for its versatility as a video player. Browsers could now dynamically update the page structure without making a request to the server, but they could not push or pull information from the server without requesting an entirely new page.

That limitation began to lift 1999 when Microsoft specified their `IXMLHttpRequest` which enabled Internet Explorer 5.0 to communicate with the server and not trigger a full page reload. Mozilla included a similar object called the `XMLHttpRequest` in december of 2000. Mozilla's object became the de facto standard until an official working draft specification was released in 2006 based on Mozilla's object. This new capability, combined with client side scripting and the *DOM* would usher in the *AJAX* era.

Let's examine that collection of technologies for a moment. JavaScript gives us access to a couple of tools to fetch data and update the page. The *DOM* allows us to call methods on objects from the `document`. DOM APIs were initially limited to handling events and updating the HTML. We have the `XMLHttpRequest` for retrieving serialized data from the server. CSS has been around since 1996, and CSS2 first showed up in 1998. Collectively, these technologies form the basis of modern web applications - even today.

Inching Closer

HTTP/1.1 finally standardized the full suite of HTTP request types, added connection reuse for transferring additional assets (scripts, images, etc), and cache-control headers (which enabled very intelligent caching) as of 1999. In 2000 Dr. Roy Fielding described *REST*, Representational State Transfer, in chapter 5 of his doctoral dissertation *Architectural Styles and the Design of Network-based Software Architectures*. This architecture, when applied to the HTTP protocol and URIs, yields scalable interactions, general interfaces, and testable components. It is the basis of modern URI design for API end points and mapping of HTTP verbs to actions.

Fielding's REST architecture, HTTP/1.1, AJAX, PHP/ASP on the server, and client side scripting enabled complex web applications, yet the limitations of the DOM APIs prevent them from being compatible with desktop apps for several years to come. In the mean time, we see the rise of web forums (vBulletin 1999, phpBB 2000), online banking is on the rise, Google has been returning extremely relevant results since 1998, and Amazon had its first profitable year in 2001.

Performance and scaling of current technologies are emphasized in the next few years. The HAProxy TCP/IP load balancer was released in 2000. The Nginx event-driven asynchronous web server and reverse proxy was first released in 2002 as an alternative to the thread-based Apache httpd. Memcached came out in 2003 as an in memory cache to reduce load on database servers. These tools are the first in a series of blows to the powerful LAMP stack. This trend continues in the following years with the rise of the full-stack server side web framework.

Python's Django is the first to the scene in mid 2005, and it is followed shortly by Ruby on Rails. Both frameworks offer traditional MVC architecture for generating dynamic markup to be rendered in the browser. These frameworks greatly simplified the development process by bundling an interchangeable web server component (allowing it to run within another web server like Apache or Nginx, or standalone), and an Object Relation Mapper for abstracting database interactions (and enable compatibility with multiple persistence engines), and built in templating solutions – just to name a few. Such tools enable developers to build sophisticated web apps in less time.

The era of apps

Apple, during a 2007 event which would change the web landscape dramatically, announced their new product called the *iPhone*. Up until now, web browsers existed only on traditional computers such as laptops and desktops. Suddenly, there would be a whole class of internet client which has dramatically different capabilities from a traditional browser - A tiny screen, battery powered, and driven by a touch interface. Of course, the web was not quite ready for this, but the release of the iPhone precipitated future HTML features like Media Queries to allow developers to design for varying screen formats.

HTML5 took the stage in 2008 and promised to enable true desktop-like capability to the browser platform. Initial improvements in browser capability included the Drag and Drop API and application cache. CSS3 starts to be deployed with features like transitions, gradients, 3d effects, and media queries. 2009 would bring us web workers for threading and IndexDB for a more traditional storage solution. The next year gave rise to audio and video tags which promise to eliminate Adobe Flash as a prerequisite for media enable applications (we are just starting to see this now with Vimeo's 2014 redesign of their video player). WebGL brought 3d graphics to the browser in 2011, and WebRTC promises to bring browser native real time video and audio streaming. The list of additional APIs goes on, but for further reading, please see the Mozilla Developer Network.

The suite of HTML5 APIs is powerful indeed, albiet unwieldy. This gives rise to a generation of client side JavaScript frameworks and MV* libraries to help structure browser code. These frameworks are used every day now to build amazing, powerful things on the web. One such library is Backbone.js which had its first commit entered in 2010.

Far from exhaustive

This timeline has only scratched the surface of what is a complex history for the web app era. Many databases have come and gone over this time period. We have seen in the last few years the rise of NoSQL and the resurgence of SQL databases. The wikipedia page *Comparison of web application frameworks* lists over 100 different server-side frameworks in several languages, and the todomvc page shows a multitude of client-side frameworks to match. Next generation browser features are being implemented every day, and we get front row seats to this history - right in our browser.