Michael Wanyoike
Technical Writer/Digital Marketer
Oct 16 • 10 min read

# History of front-end frameworks


## Introduction

Have you ever tried building a frontend web interface using just plain HTML5, CSS3 and JavaScript? Well, it's actually not that hard these days. Provided the required features are not complex, you can finish a small project relatively easily. For medium to large projects, you'll need at least one framework to deal with complexities introduced by user requirements.

Today, a beginner developer can build more complex frontend interfaces much faster than what was possible 20 years ago.

In this article, we'll look at how modern frontend frameworks grew to provide

this level of sophistication.

## Base technologies timeline

Let's first start by taking a very quick look at the basic technologies that make up the web.

The history of HTML, CSS and JavaScript is very rich and can easily fill up a whole book. I will highlight a fraction of the notable moments that took place over the last 30 years.


Tim Berners-Lee at CERN

The first HTML specification was made public in late 1991 by Tim Berners-Lee. It only supported text at the time and consisted of just 18 tags. Later, CSS was proposed by Håkon Wium Lie in October 1994.

About 2 years later, HTML 4.0 was published which was the first specification to support CSS. Internet Explorer 3 became the first commercial browser to support CSS. This was before CSS level 1 had become a World Wide Web Consortium recommendation.

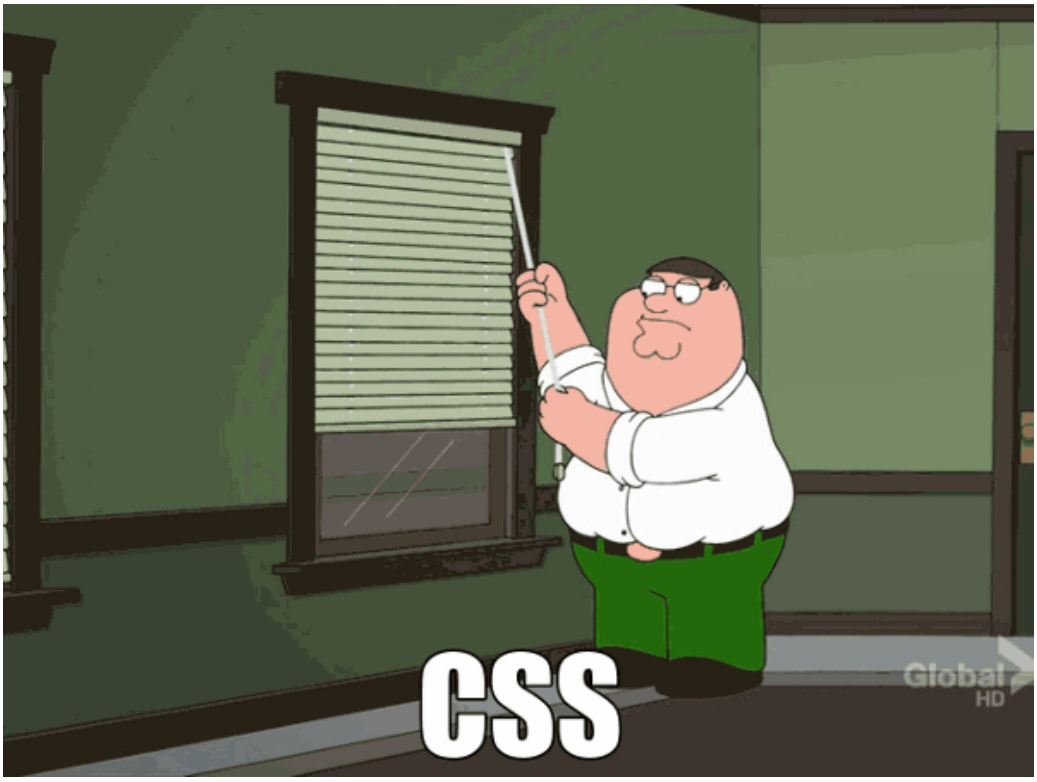
In 1995, a new browser scripting language called *Mocha* was created by Brendan Eich in just 10 days. It got renamed to *LiveScript*. A few months later it got renamed again to *JavaScript* as we now know it. Ecma International is

currently responsible for developing JavaScript specifications. HTML and CSS specifications are under the World Wide Web Consortium, also known as W3C.

## The trouble with CSS

I would like first to make an important note about CSS before we dive into frameworks.

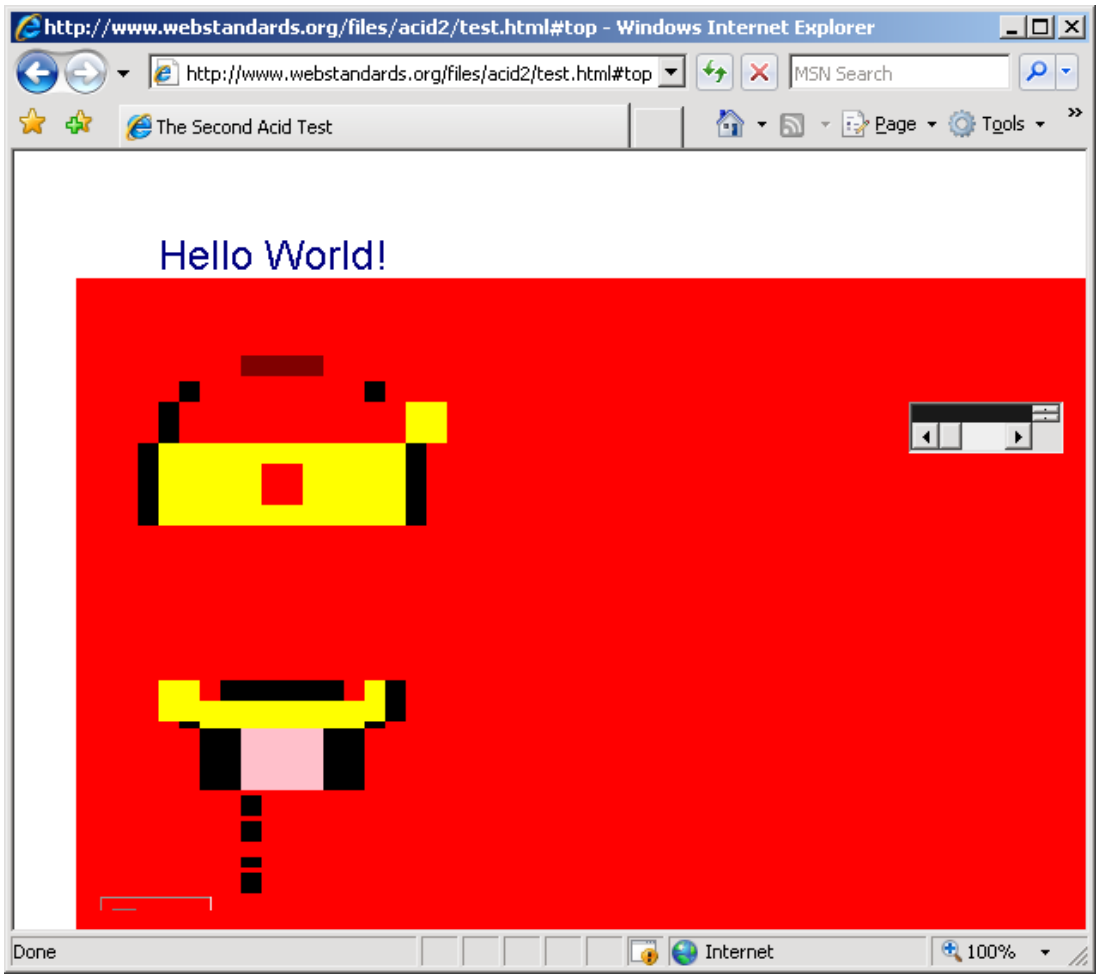
During the early years of the web, most browsers were not compliant with the standardized CSS specifications. It was very frustrating to develop a website that looked fine in one browser, and a total mess in another browser.




Supporting multiple browsers was a nightmare. At the time, Internet Explorer was the most popular browser, yet it had poorly implemented several CSS features such as the *box model*. Various CSS hacks were created to fix various problems with specific browsers.

A group known as the Web Standards Project created a series of CSS tests called the ACID tests. When CSS is implemented correctly in a browser, the following image should appear:

However, this is what appeared on Internet Explorer 7:




IE7 ACID2 screenshot

You don't want to see how bad it was in previous versions. All major browsers were affected in some way or another. Thanks to the efforts of the Web Standards Project and the online communities, a majority of CSS issues have now been resolved in major browsers.

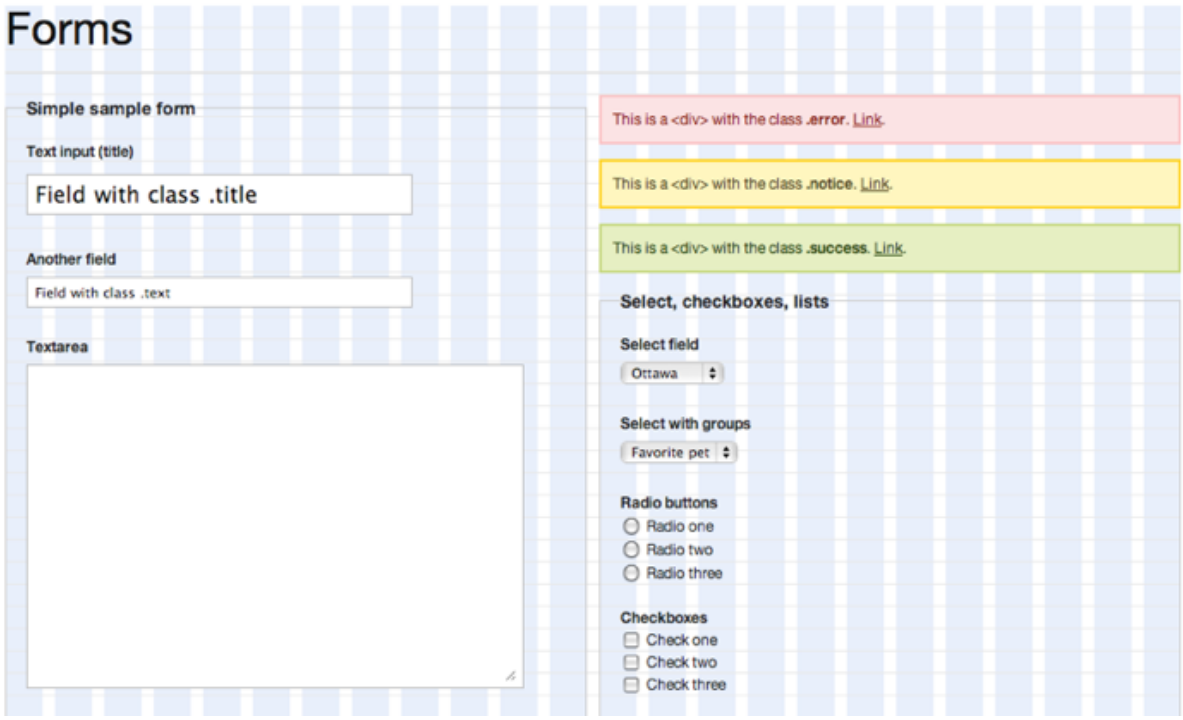
When CSS 2.1 became a *Proposed Recommendation* in April 2011, it took about 13 years to get there. CSS level 2 was the *W3C Recommendation* as from 1998. The reason it took this long is due to a few CSS features that were delaying the completion of the new revision. As for CSS3, the entire level 2.1 specification was broken down into modules. Each module is now developed independently. The breaking down of the specification has immensely helped browser vendors to catch up faster with the latest recommendations at a modular level.

Today, we have some modules in draft at level 4. Others are at level 3. If a new CSS feature gets added, it becomes level 1. You can visit the W3C CSS work page to see the current status of each module. You can also visit css3test.com to

check how well your browser supports different CSS modules.

## CSS frameworks

Around the mid-2000s, CSS libraries and frameworks began cropping up. These frameworks introduced a grid system to help web designers layout their content. The frameworks supported most browsers(one less headache for designers). Some of the earliest CSS frameworks included Blueprint, 960, YUI Grids and YAML.



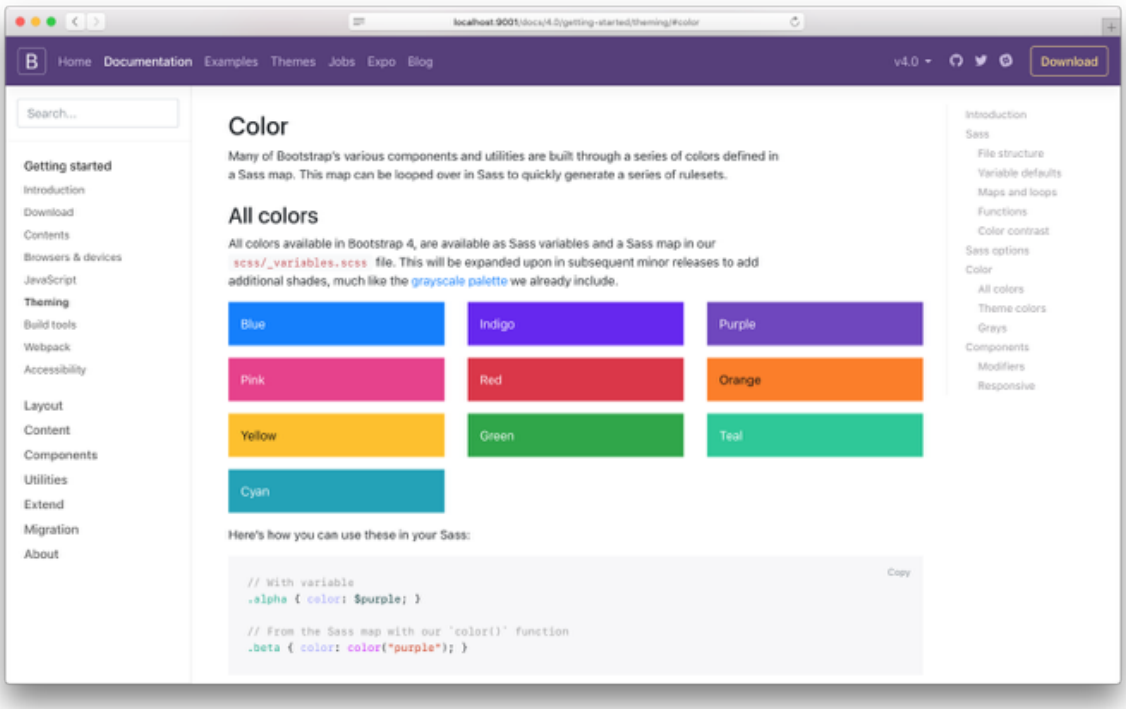

Blueprint CSS Framework UI Elements

When mobile phones manufacturers began shipping devices with web browsers, there wasn't much effort to redesign websites for mobile. The mobile browsers themselves were capable of reformatting web pages to fit into the small screens.

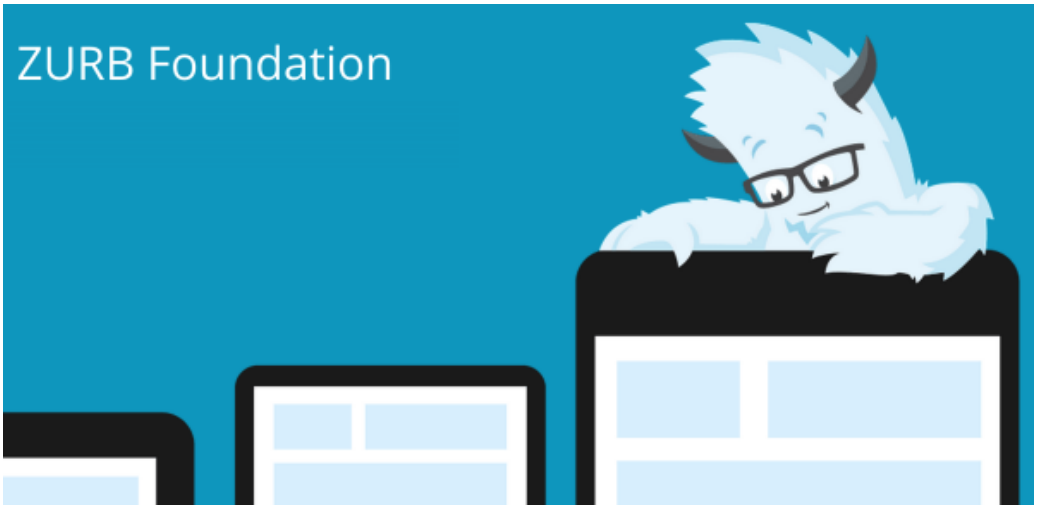
As time passed on, the hardware placed in phones improved immensely. In 2006, Opera Mobile became the first mobile browser to pass the ACID2 test. Safari was the first desktop browser to pass the test.


In 2007, the first iPhone was announced by Steve Jobs. It had the largest screen at the time. Soon Android devices were launched with similar or larger screen sizes. The popularity of smartphones and mobile browsing led to a huge uptake of Internet usage. Soon, companies began to realize the importance of having to design their websites for mobile screens.

In 2011, Twitter released the Bootstrap CSS framework as an open-source project. In 2012, Bootstrap 2 was released with a twelve-column responsive grid layout system alongside many other new features. In 2013, Bootstrap 3 was released which had redesigned components and had a mobile-first design philosophy.



Bootstrap version 4, released on August 2017, added support for Sass and Flexbox. Bootstrap is currently the most popular CSS framework. It's more popular with developers who want to quickly prototype a great looking web interface.




In October 2011, ZURB open-sourced their CSS framework and released it as Foundation 2.0. It featured a responsive grid and supported Sass. It also had an extensive list of pre-designed UI elements. Foundation is more popular with designers who want complete control over the customization of their site. Foundation also comes with templates for emails.

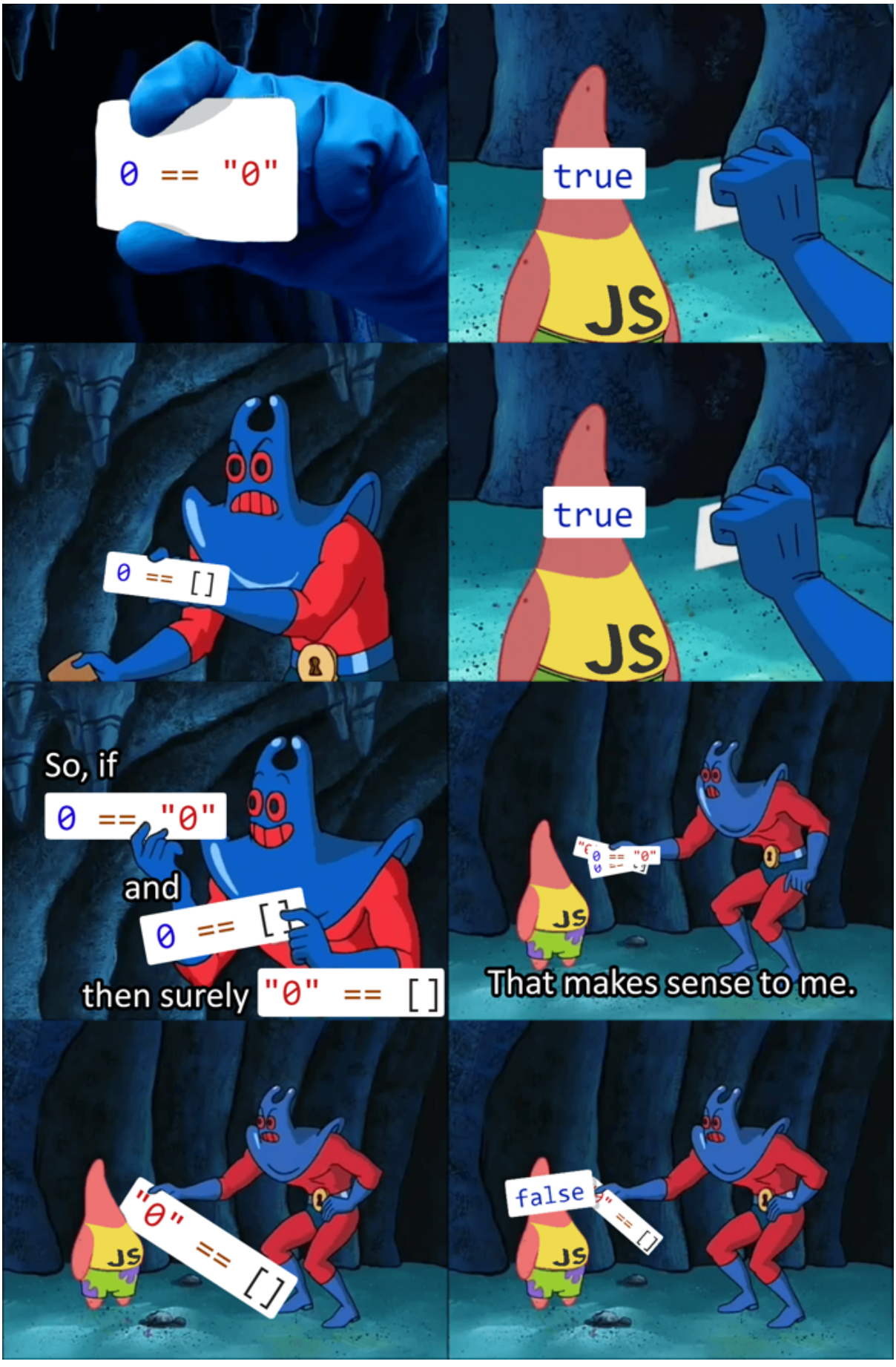
There are many other awesome CSS frameworks. You can learn more about each framework from the links below:

- Semantic UI
- Bulma
- Materialize
- UIKit
- Pure CSS
- Picnic CSS
- Skeleton

## JavaScript frameworks

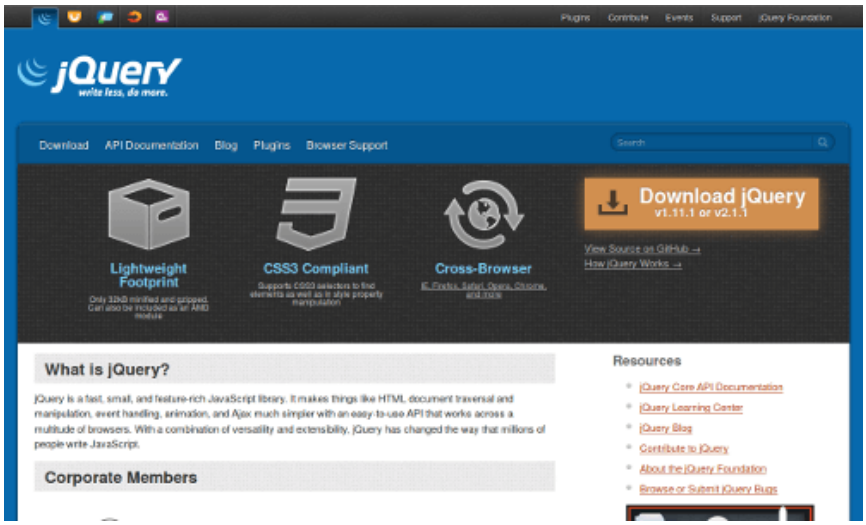
Just like CSS, JavaScript went through a bit of a rough patch where browser compatibility was an issue. You would have to put a bit of extra effort to ensure your JavaScript code ran correctly on all of the browsers that you supported.

You also needed to be aware of several quirks it had especially if you were coming from a strongly-typed language.

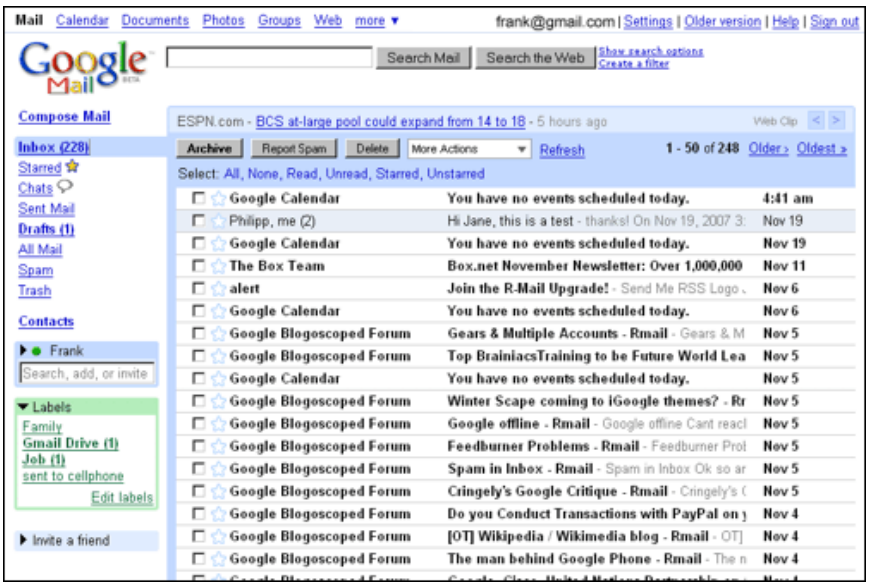



One of the earliest libraries that made working with JavaScript fun was jQuery.

It was released in August 2006. It helped developers write JavaScript code without the worry of browser compatibility. It contained tons of useful functions that made it easy to make any website interactive.




Prior to this was AJAX technology which existed as early as 1996. It had a different name back then and was initially implemented by Internet Explorer and Mozilla. However, the technology was underutilized. In 2004, Google implemented a standardized version of AJAX technology on their Gmail and Google Maps products.

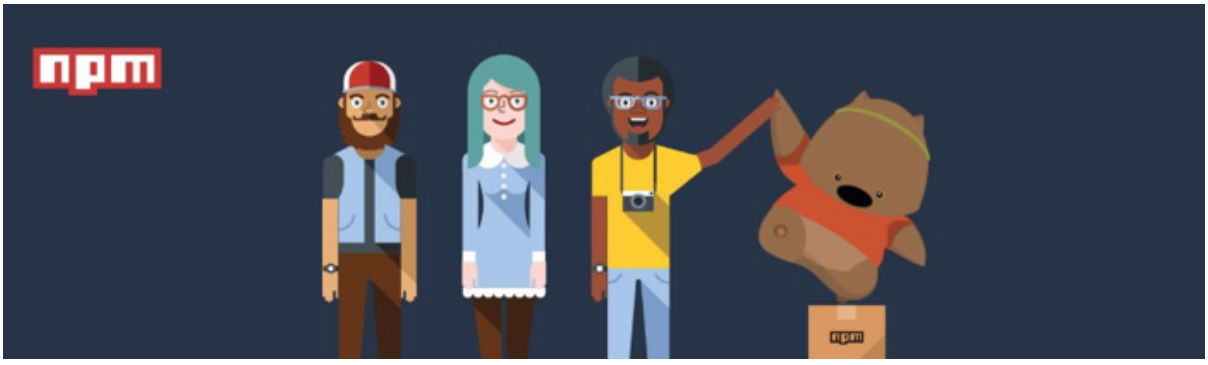



Soon after, they started implementing it on their web applications. Frameworks and libraries such as jQuery began adding support for AJAX.

As the number of frontend frameworks and libraries increased, the need for managing dependencies was desired. In 2012, Twitter launched Bower, a package manager for frontend dependencies.


Bower was used to fetch and download dependencies that were located all over the web.

In 2014, a central repository for frontend packages was created, called the npm registry.




It currently hosts over 350,000 packages.

While jQuery was the most popular JavaScript library for web interfaces, it lacked facilities for handling data consistently across shared views. Several frameworks were built to tackle this problem. Backbone, Knockout and Ember are some of the earliest such JavaScript frameworks that quickly gained popularity.

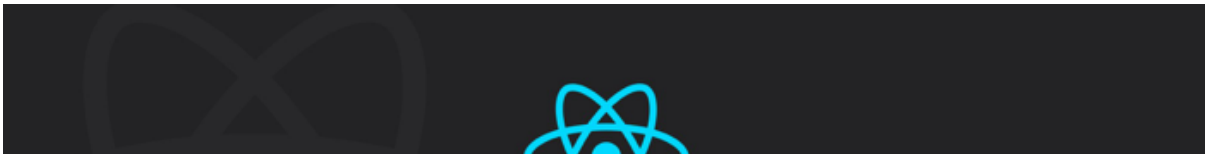
AngularJS came into the market in October 2010. It quickly became the most popular JavaScript MVC framework. It offered two-way data binding, dependency injection, routing package and much more.


AngularJS helped developers solve a lot of the problems they were facing building web projects. However, as the complexity of an AngularJS project increased, web developers began experiencing frustrations with the framework.

The AngularJS team decided to redesign the entire framework, and call it Angular 2.


Unfortunately, this new version was completely incompatible with AngularJS. There was no migration path that was provided. This infuriated a lot of AngularJS developers causing them to abandon the framework completely. Since then, Angular hasn't been able to reach the numbers it once had.

In May 2013 at a JavaScript Conference in the US, a new game-changer library called React was introduced. It was created by Jordan Walke, a software engineer working for Facebook. The audience was amazed by many of its innovative features such as the Virtual DOM, the one-way data flow and the Flux pattern.




The React team explained how this architecture helped them solved some of their biggest recurring challenges they faced when fixing bugs for Facebook. Soon after the conference, React quickly grew in popularity.

In 2015, the Redux library was created by Dan Abramov and Andrew Clarke. It became a revolutionary data-flow architecture that was inspired by Facebook's Flux architecture.


As you probably know, a rich ecosystem of libraries and tools were built for React. Even existing CSS frameworks such as Bootstrap built a version for React. Today, React is the most popular JavaScript technology. It's hard not to find a library that meets your project requirements if you are using React.

In the fall of AngularJS, a new framework that has similarities to both Angular and React has been on the rise. Vue.js, a progressive JavaScript framework, was initially released in February 2014 by Evan You.

He previously worked at Google and Meteor. Unlike React, which is very flexible, or Angular which is very opinionated, Vue tries to pick the middle ground. Vue.js provides a lightweight framework surrounded with an ecosystem that is officially maintained by the Vue team.

## Timeline recap

The process of standardization is like the glue that sticks all the layers together. Without it, frontend technology would be more fragmented and unusable. Today, this technology stack is being compressed.

To see the big picture, let's do a quick recap:

**1990 - 1995** : HTML, CSS and JavaScript are invented

**1996 - 1999** : Standardization efforts begin. Browser compliance is terrible. Browser wars ignite.

**2000 - 2004** : CSS frameworks begin to emerge. jQuery is born. Frontend package management.

**2005 - 2009 :** W3C specification compliance is met. Chrome browser takes the lead. Responsive designs and frameworks are introduced.

**2010 - 2015:** JavaScript Frameworks are born i.e. Backbone, Ember, AngularJS, React, Angular, Vue. HTML5 is announced.

**2016 - 2018:** GraphQL emerges. Native HTML, CSS & JavaScript become more powerful. New platforms built on-top existing JavaScript frameworks emerge: StoryBook, Motion UI, Gatsby, Next.js.

## The future

I'll end the article by looking at what the future of frontend frameworks looks like. The future does indeed look bright for upcoming developers as they don't have to deal with the chaos of the last decade.

Most frontend frameworks are stable now and are currently being optimized even further to make them perform better and become more developer

friendly.

Static site generators are becoming a trend. Static websites are more SEO friendly and offer a far better performance than server rendered sites. The generated websites are easier to cache and distribute over a CDN network. We currently have Next.js and Gatsby that use React. Other popular static site generators include Nuxt, Jekyll and Hugo.

The current version of HTML, CSS and Javascript are far more advanced than what it was a decade ago. Most browsers are now competing to meet compliance, hence the latest specifications get rolled out faster. It's highly likely that most of the heavy lifting tasks provided by modern libraries will soon be implemented natively in the future.

As a result, a new generation of high-performance lightweight frameworks are in the works. The Sapper framework that uses Svelte is already available to the public. Angular Ivy and React Prepack are still under development.

The Node.js platform may soon be replaced by a new one called Deno. It's being worked on by the original Node.js creator, Ryan Dahl. He addressed the unfixable issues with Node.js at a JS Conference. Those issues are now being addressed in his new project.

I really hope Deno becomes ready soon and that the whole community will shift to it quickly. It will make development a whole lot easier, leaner and faster. This may mean a rewrite of existing frameworks and libraries, which may not be backward compatible. Despite the drawback, I think the savings in overall resources used during development will be immense.

Finally, I would like to conclude by predicting that Drag n Drop builders such as Bootstrap Studio will soon become mainstream if they are open-sourced. This may happen within the next 5 years. They generate far much cleaner code than the garbage generated by WYSIWYG editors a decade ago. More people will soon start creating websites with very little technical background.

https://logrocket.com/signup/