**[The Top 10 Javascript MVC Frameworks Reviewed](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/)**

JANUARY 13, 2012

**UPDATE 1/14/2012:** *Added Batman.js and Angular.js due to popular demand and because they looked impressive.*

Over the last several months I have been in a constant search for the perfect javascript MVC framework. Driven by a dire need for the right level of abstraction and features, I have tried out - some more cursorily than others - every framework I could get my hands on. Here lies a brief synopsis of each framework. Lastly, I share the framework which I ultimately decided on.

Specifically, the following four features are very important to me:

* **UI Bindings** - I'm not just talking about templates, I'm talking about a declarative approach to automatically updating the view layer when the underlying model changes. Once you have used a framework (such as Flex) that supports UI bindings, you can never go back.
* **Composed Views** - Like all software developers, I enjoy creating modular reusable code. For this reason, when programming UI, I would like to be able to compose views (preferably at the template layer). This should also entail the potential for a rich view component hierarchy. An example of this would be a reusable pagination widget.
* **Web Presentation Layer** - We are programming for the web here people, the last thing I want are native-style widgets. There is also no reason for a web framework to create it's own layout manager. HTML and CSS are already the richest way to do style and layout in existence, and should be used as such. The framework should be centered around this concept.
* **Plays Nicely With Others** - Let's face it, jQuery is pretty amazing. I don't want a framework which comes bundled with a sub-par jQuery clone, I want a framework which recommends using jQuery itself.

**The Contenders**

Here is a table showing all of the frameworks support for the above features. Click through the title for more detail.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Framework** | **UI Bindings** | **Composed Views** | **Web Presentation Layer** | **Plays Nicely With Others** |
| [Backbone.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#backbone) | **✗** | **✗** | **✓** | **✓** |
| [SproutCore 1.x](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#sproutcore) | **✓** | **✓** | **✗** | **✗** |
| [Sammy.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#sammy) | **✗** | **✗** | **✓** | **✓** |
| [Spine.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#spine) | **✗** | **✗** | **✓** | **✓** |
| [Cappuccino](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#cappuccino) | **✓** | **✓** | **✗** | **✗** |
| [Knockout.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#knockout) | **✓** | **✗** | **✓** | **✓** |
| [Javascript MVC](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#javascript-mvc) | **✗** | **✓** | **✓** | **✓** |
| [Google Web Toolkit](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#gwt) | **✗** | **✓** | **✗** | **✗** |
| [Google Closure](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#closure) | **✗** | **✓** | **✓** | **✗** |
| [Ember.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#ember) | **✓** | **✓** | **✓** | **✓** |
| [Angular.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#angular) | **✓** | **✗** | **✓** | **✓** |
| [Batman.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#batman) | **✓** | **✗** | **✓** | **✓** |

**1. Backbone.js**

[Backbone.js](http://documentcloud.github.com/backbone/) is the web's darling framework. You can't go anywhere without hearing about it and they have an impressive list of brands using it. This was naturally one of the first frameworks I tried. I used it to build some of our internal administrative features at[GroupTalent](http://grouptalent.com/).

**Pros:** Strong community and lots of momentum. Underscore.js (which it uses heavily) is also a great framework.

**Cons:** Lacks strong abstractions and leaves something to be desired. The entire framework is surprisingly lightweight and results in lots of boilerplate. The larger an application becomes, the more this becomes apparent.

**2. SproutCore 1.x**

[SproutCore](http://sproutcore.com/) is what Apple used on its iCloud initiative. Despite having a horrible name, it is actually an extremely well thought out framework. It is also one of the largest frameworks.

**Pros:** Bindings support. Solid community. Tons of features.

**Cons:** Overly prescriptive. Hard to decouple from unneeded features. Forces a native-like paradigm. I have a serious problem with any framework which discourages using html for layout.

**3. Sammy.js**

[Sammy.js](http://sammyjs.org/) was a smaller framework that I stumbled upon. Due to its simplicity, it almost didn't make this list. It's core feature is a routing system to swap out areas of an application with AJAX.

**Pros:** Simple learning curve. Easier to integrate with an existing server side app.

**Cons:** Too simple. Not sufficient for larger applications.

**4. Spine.js**

Based on the name, [Spine.js](http://spinejs.com/) is obviously heavily influenced by backbone. Like backbone, it is very lightweight and follows a similar model.

**Pros:** Lightweight with good documentation.

**Cons:** Fundamentally flawed. A core concept of spine is "is asynchronous UIs. In a nutshell, this means that UIs should ideally never block". Having built a serious non-blocking realtime application in the past, I can say this is entirely unrealistic unless the backend has something like [operational transformation](http://en.wikipedia.org/wiki/Operational_transformation).

**5. Cappuccino**

[Cappuccino](http://cappuccino.org/) is one of the more unique frameworks, coming with its own language Objective-J. Cappuccino tries to emulate Cocoa in the browser.

**Pros:** Large thought-out framework. Good community. Great inheritance model.

**Cons:** Out of all the languages you could emulate in javascript, Objective-C would be my last choice. This is coming from an iOS developer. I simply can't get past the idea of programming Objective-J in the browser.

**6. Knockout.js**

[Knockout.js](http://knockoutjs.com/) is an MVVM framework that receives lots of praise from its supporters. It stresses declarative UI bindings and automatic UI refresh.

**Pros:** Binding support. Great documentation and amazing tutorial system.

**Cons:** Awkward binding syntax and lacks a solid view component hierarchy. I want to be able to reuse components easily. I also feel like identifying as an MVVM framework is deleterious. Hardly any of these frameworks are MVC, but are of the MV\* variety (MVP, MVVM, etc).

**7. Javascript MVC**

[Javascript MVC](http://javascriptmvc.com/), in the interest of full disclosure, is a framework that I didn't spend very much time evaluating.

**Pros:** Solid community and legacy.

**Cons:** Awkward inheritance model based on strings. Controllers are too intimate with views and lack bindings. The name is way too generic - the equivalent would be if RoR was called "Ruby Web Framework".

**8. Google Web Toolkit**

[GWT](http://code.google.com/webtoolkit/) is a serious client-side toolkit that includes more than just a framework. It compiles Java to Javascript, supporting a subset of the standard java library. Google used it internally for Wave.

**Pros:** Comprehensive framework with great community. Solid Java-based component inheritance model. Great for behemoth client-side applications.

**Cons:** Despite what Google says, GWT is not going to stand the test of time. With initiatives like [DART](http://www.dartlang.org/) its clear that Java is not the future of the web. Furthermore, the abstraction of Java on the client is slightly awkward.

**9. Google Closure**

[Google Closure](http://code.google.com/closure/) is more of a toolkit than simply a javascript framework. It comes bundled with a compiler and optimizer.

**Pros:** Use by Google for many of their major apps. Nice component-based ui composition system.

**Cons:** Lack of UI-binding support.

**10. Ember.js**

[Ember.js](http://emberjs.com/) (formerly ~~Amber.js~~ SproutCore 2.0) is one of the newest contenders. It is an attempt to extricate the core features from SproutCore 2.0 into a more compact modular framework suited for the web.

**Pros:** Extremely rich templating system with composed views and UI bindings.

**Cons:** Relatively new. Documentation leaves lots to be desired.

**11. Angular.js**

[Angular.js](http://docs.angularjs.org/) is a very nice framework I discovered after I originally posted this review. Developed by Googler's, it has some very interesting design choices.

**Pros:** Very well thought out with respect to template scoping and controller design. Has a dependency injection system (I am a big fan of IOC). Supports a rich UI-Binding syntax to make things like filtering and transforming values a breeze.

**Cons:** Codebase appears to be fairly sprawling and not very modular. Views are not modular enough (will address this in more detail in the cons of [Batman.js](http://codebrief.com/2012/01/the-top-10-javascript-mvc-frameworks-reviewed/#batman)).

**12. Batman.js**

[Batman.js](http://batmanjs.org/), created by Shopify, is another framework in a similar vein to Knockout and Angular. Has a nice UI binding system based on html attributes. The only framework written in idiomatic coffeescript, it is also tightly integrated with Node.js and even goes to the extent of having its own (optional) Node.js server.

**Pros:** Very clean codebase. Has a nice simple approach to binding, persistence, and routing.

**Cons:** I very much dislike singletons, let alone the idea of enforcing singleton controllers. Suffers from the same ailments as Knockout and Angular with regards to nested components. I want to be able to declaratively reuse more than just templates. What Ember has over these frameworks is a way to declaratively re-use entire components that are backed by their own (possibly controller-level) logic.

**The Winner**

At the end of the day, Ember.js is the *only* framework which has everything I desire. I recently ported a relatively small Backbone.js application over to Ember.js and, despite some small performance issues, I am much happier with the resulting code base. Being championed by [Yehuda Katz](http://yehudakatz.com/), the community around Ember.js is also amazing. This is definitely the framework to watch out for.

Of course this list is far from comprehensive. Almost all of these frameworks here were discovered by sheer notoriety, word of mouth, or by being mentioned on [Hacker News](http://news.ycombinator.com/). I am also not reviewing proprietary frameworks (or frameworks with disagreeable licenses - ExtJS anyone?).

What MVC framework do you use?