jQuery vs. AngularJS: A Comparison and Migration Walkthrough

https://www.airpair.com/angularjs/posts/jquery-angularjs-comparison-migration-walkthrough

1 TL;DR Cheatsheet

For those of you in a hurry, the following chart is a "too long; didn't read" summary.

	jQuery	AngularJS
Abstracts the DOM	✔	✔
Unit Test Runner	✔	✔
Deferred Promises	✔	✔
Cross-Module Communication	✔	✔
Animation Support	✔	✔
AJAX / JSONP	✔	✔
RESTful API	✘	✔
Integration Test Runner	✘	✔
MVC Pattern Support	✘	✔
Templating	✘	✔
Two-way Data Binding	✘	✔
Dependency Management	✘	✔
Deep-Link Routing	✘	✔
Form Validation	✘	✔
Localization	✘	✔
File Size	32KB	38KB

1 Framework Comparison

DISCLAIMER: The following comparisons are subjective. What would be counted as a strength in one case could be considered a drawback in another. For example, using complex CSS3 selectors in jQuery lets you do a lot with a single line of code, but that does not mean it is the most efficient way to interact with the Document Object Model.

1.1 The jQuery Library

The jQuery library is a modular set of cross-browser methods for making AJAX requests, manipulating elements, triggering and listening for events, selecting elements from the DOM, running animations and effects, getting and setting form input values, traversing the DOM, using deferred promises to manage future events, and more. It works as a facade to standardize and ease the task of programmatically interacting with elements on a web page.

1.1.1 Intuitive API

Even as modern browsers are becoming more standardized, there is no doubt that cross-browser DOM manipulation is still painful. One of the greatest strengths of the jQuery library is its clean and intuitive API. New developers have no trouble picking it up quickly, yet it is powerful enough for JavaScript experts to leverage as well.

Suppose we have the following HTML markup, and we want to add the text 'World' to the message 'Hello' when the button is clicked the first time.

```
<span id="msg">Hello</span>

<button id="btn">Click Me</button>
```

Like learning from posts like this? **Subscribe for more!**

The following Vanilla JavaScript code demonstrates one way to do this (live example).

```
// Vanilla JavaScript

var btn = document.getElementById('btn'),

    msg = document.getElementById('msg');


function setMessage() {


  // update the message

  msg.innerHTML += ' World';


  // remove the event listener

  if (btn.removeEventListener) {
```

```
    btn.removeEventListener('click', setMessage, false);
  } else if (btn.detachEvent) {
    btn.detachEvent('click', setMessage);
  }
}

// check for supported event subscription method
if (btn.addEventListener) {
  btn.addEventListener('click', setMessage, false);
} else if (btn.attachEvent) {
  btn.attachEvent('click', setMessage);
}
```

Like learning from posts like this? **Subscribe for more!**

In contrast, it is trivial to do the exact same thing with jQuery (live example).

```
// jQuery
$('#btn').one('click', function() {
  $('#msg').append(' World');
});
```

Like learning from posts like this? **Subscribe for more!**

Pretty sweet, right?! You can see why jQuery's slogan is, "Write less. do more". This is the primary reason jQuery is so popular, with nearly 50 million websites using it .

However, just because you *can* do something with a single line of code does not mean you *should*. As an example, I came across the following jQuery selector in production code, which is trying to select the child checkboxes for a given category but is *horribly* inefficient. It was taking over **6 seconds** to run.

```
// jQuery
$('#' + subcatID + ' .childLevelCheckbox[catid=' + catid + ']:not(:checked)')
```

Like learning from posts like this? **Subscribe for more!**

After refactoring the code to use simple selectors and manually looping on the elements, the time dropped to **13 milliseconds**. *Take away:* do not do your filtering directly on the DOM. It helps to know that the jQuery selector engine works from right to left. Take the following selector, for example:

```
// jQuery
$('#' + subcatID + ' input[catid=' + catid + ']);
```

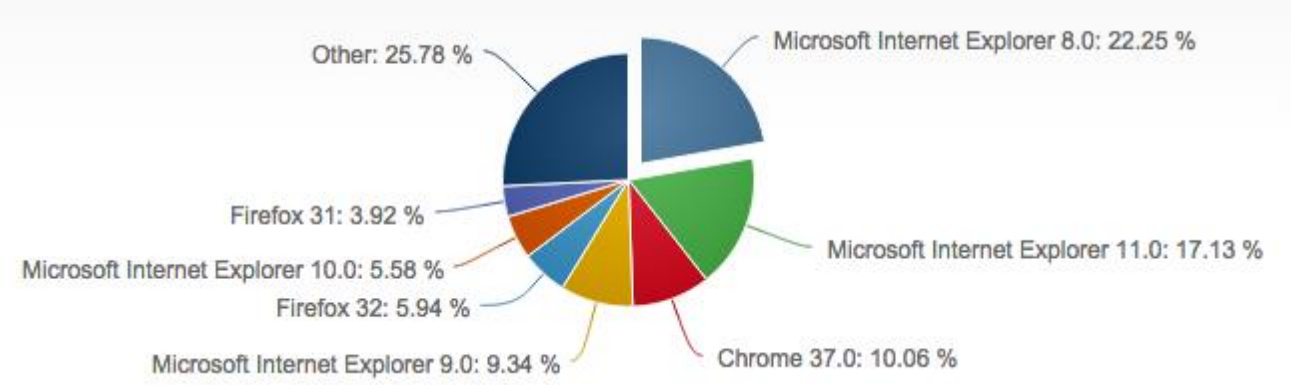
Like learning from posts like this? **Subscribe for more!**

For the above selector jQuery does the following:

- First it finds all the input elements on the page;
- Then, it checks if it has a lowercase attribute called catid with the value we are looking for;
- Next, it walks up the DOM tree to check that it is a child of the required subcatID;
- Finally, it can return a list of elements that satisfy that criteria.

1.1.2 Cross-Browser Support

Web browsers have come a long way since the "browser wars" of the late 1990s and Netscape's `document.layers`. But even today, for something as simple as working with the text inside an element, some browsers use `textContent` while others (read Internet Explorer*) use `innerText`. With jQuery you can get or set the text of an element simply by using `$(elm).text()` and avoid those shenanigans. Thankfully, Internet Explorer did get `textContent` support in version 9, but version 8 still accounts for over 22% of all desktop web traffic.



You *could* create your own utility methods for adding event listeners or for making AJAX requests, but there is no reason to reinvent the wheel when there is a battle-tested solution available for free. You can simply use `$.ajax` to pull in your dynamic data and be confident that it will work the same in the Chrome browser as it does in Internet Explorer. Doing so allows you to spend more time working on what makes your application unique instead of tracking down bugs that have already been solved.

1.1.3 Community Size

When choosing an open source technology it is important to take into account the size and level of active participation within the development community. Is the solution you are considering gaining in popularity and support, or is it waning? Will you be able to get help by searching for the issues you are running up against? jQuery is certainly a winner, with nearly 60% of all websites using it, hundreds of thousands of plugins ready to augment the library, tutorials of every kind, as well as extensive and well-maintained documentation about every feature the library provides. I did not find any definitive source on the number of jQuery plugins

available, so this is an informal estimate, but GitHub alone has 175,000 results when searching for jQuery plugin.

1.1.4 Smaller Footprint

While file size certainly is not the most important consideration when choosing a JavaScript framework, it is worth mentioning that jQuery currently weighs in at 32KB minified and gzipped. By comparison the most recent stable version of AngularJS is 38KB compressed and minified. So what does this mean? It means that if jQuery is all you need, then you should use it! But if you need the more advanced functionality available in AngularJS, then it is the clear winner. You might ask why:

"Can't I just mix and match the jQuery plugins that my team prefers to use?" - You

Yes you can, and there are some benefits to doing that. But I maintain they are out-weighed by the drawbacks of managing the many 3rd party dependencies required to build a large-scale jQuery application. Once you take into account adding in 10 or 15 jQuery plugins, you may find that your footprint is not smaller after all.

1.1.5 Open Source Additions

If you find you need to stick with jQuery but still want to use more advanced features when building your application such as templating, data-binding, and dependency management, I suggest you look into the following open source projects.

- **Templating:** There are many open source templating solutions to choose from including:
    - - Mustache available in Ruby, JavaScript, Python, Erlang, node.js, PHP, Perl, Perl6, Objective-C, Java, .NET, Android, C++, Go, Lua, ooc, ActionScript, ColdFusion, Scala, Clojure, Fantom, CoffeeScript, D, Haskell, XQuery, ASP, Io, Dart, Haxe, Delphi, Racket, Rust, and C#. I like the logic-less approach of a mustache template as it encourages a separation of concerns. Also, having the template language in both client-side and the server-side language of your choice is a huge win for template reuse, consistency, and maintainability.
    - - Handlebars is basically a superset of Mustache (above) with the goal of putting a small amount of logic back into the logic-less templates. For example, you can register custom methods to help process the JSON data. As well as the ability to change the current context by using paths such as `../` to change to the parent context. As for using it server-side, you are restricted to Node.js, but it is possible - by pre-compiling the templates - to use the same templating language on both the client and server.
    - - Dust.js is currently my favorite client-side templating language. Dust takes the best of Mustache and Handlebars. Like Handlebars, it extends the Mustache syntax with path support and helper functions, while adding in a few features of

its own, such as asynchronous rendering, built in filters, support for streaming, and inline parameters while still remaining highly performant.

- **Data-Binding:** It can be very helpful to use one or two-way data-binding to ease syncing updates to and from DOM elements, given the dynamic nature of many modern web applications.
    - - Knockout is an MVVM framework allows you to easily associate DOM elements with model data using a concise, readable syntax. When your data model's state changes, your UI updates automatically. Knockout enables you to implicitly set up chains of relationships between model data, to transform and combine it.
    - - Rivets.js is agnostic about your model/controller layer and works well with existing libraries that employ an event-driven model such as Backbone.js and Stapes.js. It ships with a built-in adapter for subscribing to plain JS objects using ES5 natives, however this can be replaced with a Watch.JS adapter or an Object.observe adapter.
    - - Ractive.js is a template-driven UI library for building reactive user interfaces. It supports declarative two-way data-binding, event handling through a publish/subscribe mechanism, and performant animations and transitions with intro and outro directives.
    - - Ember.js provides a convention-over-configuration opinionated framework for building large web applications, and provides tools for url routing, data-binding templates, creating components, loading data from servers, and more.
- **Dependency Management:**
    - - RequireJS is a very popular module file loader that is capable of running in most web browsers, Node.js, and Rhino and supports the Asynchronous Module Definition (AMD) format.
    - - Inject is a dependency management solution that supports both the CommonJS and AMD module formats, as well as cross-origin fetching, localStorage caching, and more.
    - - StealJS is designed to simplify dependency management while maintaining flexibility and power. It supports ES6, CommonJS, AMD, CSS, LESS and more.

The fact that a diverse group of open source solutions will all be of varying levels of code quality and developed on independent timelines is important to consider when choosing your front-end architecture stack.

"Improve maintainability by minimizing dependencies." - Daniel Lamb

Like trying to herd cats, the more diversified technologies you have in your code base, the more divergent they can become over time. This is not to say that you should avoid open source innovations; it is amazing to be able to leverage an entire community of developers to accomplish a common goal. However, it is a good idea to keep an eye on the number of dependencies on which your application relies.

1.2 The AngularJS Toolset

AngularJS is a toolset for building the framework most suited to your application development. - angularjs.org

In the spectrum of structure they provide for building web applications, I would put AngularJS in between Backbone.js and Ember.js. With Backbone (as the name implies), you are given just the "bare bones," as it were, with the freedom to implement your application however you see fit. On the other hand, Ember.js is highly opinionated about how things should be done. (That is not necessarily a bad thing; having a framework that promotes adherence to solid design patterns can often be more helpful than restrictive.)

AngularJS strikes a nice balance between these two frameworks by giving you a structure that encourages a separation of concerns, with constructs to organize your code into controllers, services, constants, directives and filters, while maintaining implementation flexibility and extensibility.

1.2.1 Unit Testing

I am passionate about code quality, and unit testing is an important component for maintaining quality over time and aid in refactoring. AngularJS was built from the very beginning to be testable. That means the AngularJS team set out to remove every possible excuse for *not* testing an AngularJS application. As an example, AngularJS uses the dependency injection design pattern so you can easily swap out your production code for mocks.

The team even went so far as to create a unit test runner called Karma (originally Testacular) that supports the popular test frameworks Jasmine, Mocha and QUnit to make sure you can run unit tests.

1.2.2 End-to-End Testing

One area often overlooked by client-side solutions is the use of integration or end-to-end tests to validate page behavior. Once again, the AngularJS team has gone above and beyond by open sourcing an end-to-end test framework called Protractor.

Protractor is a powerful tool built on top of WebDriverJS. It executes your tests in real browsers, ensuring your application is exercised just like a user would. By using both Karma and Protractor, *all* of your tests can be written in JavaScript with a single testing framework (such as Jasmine).

This means you can use the same assertion patterns and mocking strategies whether you are writing unit tests for your features or end-to-end tests for web interactions, thereby improving test writing efficiency and reducing cognitive load, a considerable benefit over other available solutions.

Yet even with all the testability and tools AngularJS has to offer, there are still gaps that need addressing, e.g.: performance testing and load testing (although some progress has been made in defining AngularJS test patterns for common scenarios such as A/B/n Testing).

1.2.3 Integration

As with many powerful tools, I have found that configuration can be cumbersome and inflexible when integrating into your software development lifecycle. To streamline this process, I created an open source project called AQUA or "Automated QUality Analysis". AQUA raises the visibility of code quality and increases awareness within teams by giving instant feedback about code smells before they become technical debt.

You are free to structure your projects any way you like, and still take full advantage of AQUA. Simply create an `aqua.project.json` file that documents where things are. The goal of AQUA is to improve code quality without dictating a strict project structure. Ramp-up gradually by only configuring the parts of AQUA you want to use.

1.2.4 Template Data-Binding

jQuery is great for adding interactivity to a website, but it does not supply us with tools for data-binding a model to dynamic template views.

AngularJS, however, provides us with declarative (vs. imperative) two-way data-binding. How do these two methods compare?

Imperative Data-Binding with jQuery (live example)

```
<!-- jQuery Markup -->
<div>
    <input id="name" type="text" />
    <span id="greeting">Hello</span>
</div>
```

Like learning from posts like this? **Subscribe for more!**

```
// jQuery
//look up the input element
var name = $('#name');

//look up the output element
var greeting = $('#greeting');

//listen for keyboard events
name.keyup(function() {

  //update the output element with the new input
  greeting.text('Hello ' + name.val());
```

```
});
```

Like learning from posts like this? **Subscribe for more!**

Even though the clean API of jQuery abstracts away a lot of cross-browser grunt work for us, it is even less code with AngularJS. Less Code === Less Bugs!

Declarative Data-Binding with AngularJS (live example)

```
<div ng-app>
    <input type="text" ng-model="name" />
    <span>Hello {{name}}</span>
</div>
```

Like learning from posts like this? **Subscribe for more!**

```
// *cricket cricket* That's it, no JavaScript needed in AngularJS!
```

Like learning from posts like this? **Subscribe for more!**

1.2.5 Dependency Management

The difficulty in managing dependencies grows with the size of an application, especially if you are following good design patterns such as the single responsibility principle and are writing your code in a modular fashion.

While other libraries like the Google Closure Library configure dependencies with explicit calls to `goog.require`, in its simplest form, AngularJs uses an elegant and intuitive annotation style by simply using the arguments of the function declaration.

```
// AngularJS
myModule.controller('MyController', function($scope, dep1, dep2) {
  // ...
  $scope.myMethod = function(arg1) {
    return dep1(arg1);
  };
  // ...
});
```

Like learning from posts like this? **Subscribe for more!**

When `MyController` is created by AngularJS, the `$scope` and instances of `dep1` and `dep2` are automatically passed in. This eliminates the need for hard-coding dependencies with global variables or creating instances manually within the component, which greatly complicates and limits testability.

1.2.6 Learning Curve

It takes time to master all that AngularJS has to offer. There is a longer learning curve involved with AngularJS than with jQuery. Fortunately, AirPair can help you, with access to many top-shelf AngularJS experts.

2 Migration Walkthrough

Given the strengths and drawbacks of the two frameworks just reviewed, you may have a better idea of which one will best fit your specific needs. The remainder of this article assumes it would benefit you to migrate from jQuery to AngularJS and discusses possible approaches you can use.

2.1 Greenfield Projects

When testing out new technology, a common approach to mitigate the risks is to try it on a completely new feature of your website. To use the "Greenfield Project" method, it is important to choose a section or page that has little to no interactions with your legacy systems so it is safer to experiment. If you don't have the luxury of starting from scratch, next I will cover a few ideas on migrating existing code to AngularJS.

2.2 Case Study

Let's set the stage for our discussion around the premise that our team needs to build a robust, non-trivial, single page application using JavaScript.

I like concrete examples, so I am going to create a fictional social network called airstream to use for our case study. The jQuery version of airstream will use Sammy.js for basic routing support, and Mustache.js for templating. The AngularJS version of the site will provide the same functionality as the jQuery version, but will only rely on the core AngularJS libraries.

2.3 From jQuery Plugin to Angular Filter

Our fictional social network will facilitate communication between its users through posts. Like any good social network, we want to automatically link the text of the post when it contains mentions or hashtags. Let's say that our jQuery application does this using regular expressions that are encapsulated in a plugin (live example):

```
// jQuery
(function ($) {
    var link = '<a href="http://airstream.com/$1">$1</a>',
        list = [/\B(@[\w]*)/g, /\B(#[\w]*)/g];
    $.fn.autolink = function () {
        return this.each(function () {
            var elm = $(this);
```

```
            $.each(list, function (i, regex) {
                elm.html(elm.html().replace(regex, link));
            });
        });
    };
})(jQuery);
```

Like learning from posts like this? **Subscribe for more!**

Before you begin migrating your code, I strongly suggest you take the time to verify its current behavior. This way, you can be sure regressions will not be introduced as it is moved over to the new framework.

```
// jQuery
test("should link @ mentions", function() {
  // arrange
  expect(1);
  var text = 'this is a @daniel test.',
    expected = 'this is a <a href="http://airstream.com/@daniel">@daniel</a> test.',
    $fixture = $("#qunit-fixture").text(text);
  // act
  $("#qunit-fixture").autolink();
  // assert
  equal($fixture.html(), expected, "mentions should be linked");
});
test("should link hashtags", function() {
  // arrange
  expect(1);
  var text = 'this is a #test.',
    expected = 'this is a <a href="http://airstream.com/#test">#test</a>.',
    $fixture = $("#qunit-fixture").text(text);
  // act
  $("#qunit-fixture").autolink();
  // assert
  equal($fixture.html(), expected, "hashtags should be linked");
});
```

2.4 Full Migration

Taking the time to completely rewrite your jQuery plugins into "native" AngularJS directives will take longer than the hybrid approach mentioned below, but the long term benefits of modularity and testability are well worth it. Since we are manipulating string data, an AngularJS filter will be a good fit for migrating the auto-link feature to AngularJS (live example):

If you're following a TDD process you can start by moving over your unit tests first:

```
// AngularJS
describe('when evaluating an expression', function() {
  it('should link @ mentions', function() {
    var text = 'this is a @daniel test.',
      expected = 'this is a <a href="http://airstream.com/@daniel">@daniel</a> test.';
    expect(autolink(text)).toBe(expected);
  });
  it('should link hashtags', function() {
    var text = 'this is a #test.',
      expected = 'this is a <a href="http://airstream.com/#test">#test</a>.';
    expect(autolink(text)).toBe(expected);
  });
});
```


Then write the code that fulfills them:

```
// AngularJS
angular.module('airstream', ['ngSanitize']).filter('autolink', [function () {
    var link = '<a href="http://airstream.com/$1">$1</a>',
        list = [/\B(@[\w]*)/g, /\B(#[\w]*)/g];
    return function (text) {
        angular.forEach(list, function (regex) {
            text = text.replace(regex, link);
        });
        return text;
    };
}]);
```

2.5 Hybrid Migration

However, what if you have complicated jQuery plugins, and don't want to take the time to port them over before you start using AngularJS? It is possible to take advantage of the fact that AngularJS can augment jQuery. In short, you create an empty shell of the component you want to migrate, write your unit tests for it, but call the original jQuery plugin under the covers. Then as you have time you can move the code over. But I do not recommend that approach. Take a look at the following (live example):

```
// AngularJS
angular.module('airstream', ['ngSanitize']).filter('autolink', [function () {
    return function (text) {
        // This works because angular.element can extend jQuery
        return angular.element('<span>').text(text).autolink().html();
    };
}]);
```

Like learning from posts like this? **Subscribe for more!**

3 Conclusion

As I have outlined in this article, AngularJS can do everything that jQuery does and much more, yet is roughly equivalent in download size. It is easy to both write and run unit tests and end-to-end tests for AngularJS applications. Dependency management is effortless and intuitive. Binding dynamic data to your views is straightforward and powerful. Other topics I didn't cover - such as directives, routing, services, validation, resources, animation and localization - are equally thought out and useful tools. **AngularJS is a solid foundation for building testable web applications that scale.**

4 Errata

Typos happen. Please notify me if you find any technical or information inaccuracies so I can update this article to address them, at dlamb.open.source@gmail.com.