AngularJS IN ACTION

Brian Ford
Lukas Ruebbelke

MEAP

MANNING

MEAP Edition
Manning Early Access Program
AngularJS in Action
Version 9

For more information on this and other Manning titles go to
www.manning.com

brief contents

1 Introduction - AngularJS

5 key points:

- AngularJS is a client-side JavaScript framework
- It's used to build web applications
- It is prescriptive
- Makes creating a user interface (UI) easier through data-binding
- It helps organize and architect an application

If you are reading this book, you probably know something about AngularJS. You might've heard that it has custom elements and attributes, or it uses dependency injection. At very least you're curious. In short, AngularJS is a framework that helps you build front-ends for web-based applications. In this chapter, we're first going to describe AngularJS by defining some key terms and explaining AngularJS's relation to them. Then we'll look at the benefits and dig into a hello world example. By the end of the chapter you should have a pretty good idea of what AngularJS does and what role it can play in your software projects. All of what you learn in chapter 1 will set the stage for when we introduce our running example in chapter 2.

1.1 What is AngularJS?

In a sentence, AngularJS is a prescriptive client-side JavaScript framework used to make single-page web apps. That's a mouthful! Let's talk break it down piece by piece. AngularJS is prescriptive in the sense that it has a recommended way of building web apps. It has its own spin on the ubiquitous Model View Controller (MVC) pattern that is especially well suited for JavaScript. In this way, Angular is prescribing a way for you to divide your application up into smaller parts. AngularJS is a framework that runs in the web browser. A framework (as opposed to a library) has some overarching idea of how an app should work, and typically handles things like starting the app up and navigating to different parts of the app. AngularJS is a JavaScript framework because (unremarkably) it is used to write apps in JavaScript. If you are comfortable writing JavaScript, you should be right at home using AngularJS. Beyond that, we said AngularJS is for single page apps, meaning the browser only loads the page once, but then makes asynchronous calls to the server to fetch new information. Here asynchronous means that the application can continue to function without having to stop and wait for the server to respond.

In this section we'll look at the features of AngularJS and talk about the benefits to using Angular over other approaches in building web apps. We're going to expand on these ideas in 1.1.1 as we talk about the core features of AngularJS and why they are important. We'll also walk through a super basic AngularJS app to get your feet wet.

1.1.1 Core Features

As we kick off the AngularJS tour, we are going to visit some of the most important features so that you will have a familiarity of all the pieces we are going to use to make the awesome sample application in the rest of the book.

Table 1.1 Core Features of AngularJS

Two Way Data-binding	Model View Whatever
HTML Templates	Deep Linking
Dependency Injection	Directives

TWO WAY DATA-BINDING

Let's say you have a form to store user contact info and you want to update the addresses input box when data is returned from the server. You could do the following:

Listing 1.1

```
<span id="yourName"></span>
```

Listing 1.2

```
document.getElementById('yourName')[0].text = 'bob';
```

You manually seek to the element you want to change, calling a long, awkward series of DOM methods. This is brittle and can require some heavy lifting depending on how complex your DOM is. You also have to make this call every time something causes the address to change. Rather than spend your time worrying about these details, you could instead use AngularJS's two-way data binding. If we take the same example above and use data-binding we accomplish the same thing but faster. Here's our example with data-binding:

Listing 1.3

```
<span>{{yourName}}</span>
```

Listing 1.4

```
var yourName = 'bob';
```

In this approach you avoid repeating yourself, or having to worry about when some state changes and the form needs to be updated. It all happens automatically, thanks to AngularJS. This means that there is only one place in your application where that piece of information is stored. When that data changes, everything that is displaying that information just needs to update to reflect the new value. This is where two way data-binding comes in! It makes it easy to change a value and effortlessly have it update the DOM.

MODEL VIEW WHATEVER

If you're an experienced web developer, you're likely familiar with the Model View Controller (MVC) design pattern. If you've never heard of MVC, don't worry about it, the idea is quite simple. MVC is a pattern for dividing an application into different parts (called Model, View, and Controller), each with distinct responsibilities. There are many different variants on MVC, depending on how responsibilities are divided between parts. Two examples of popular variants are Model View Presenter (MVP) and Model View ViewModel (MVVM).

Although AngularJS's architecture closely resembles MVC and its variants, the way in which AngularJS maps responsibilities to parts doesn't perfectly fit any of them. Rather than become embroiled in a debate about the philosophical nuances of design pattern architecture, the AngularJS team humorously refers to AngularJS as a "Model View Whatever" framework.

We have personally found that the Model View ViewModel (MVVM) pattern is a close enough approximation for a general illustration of how AngularJS works. Take a look at Figure 1.1 to get a better picture of how MVVM works. In the diagram you can see that app.html serves as the View and is bound to ViewModel which is app.js. If someone were to change the name property on `$scope.user` in app.js, it would automatically update the property in the name field in app.html. The converse is true in that if someone were to change the name field it in app.html, it would automatically update the `$scope.user` object. App.html can also issue commands to app.js such as calling `$scope.save` and letting app.js perform some unit of logic such as saving the `$scope.user` object to users.js so that it can be shared elsewhere in the application.

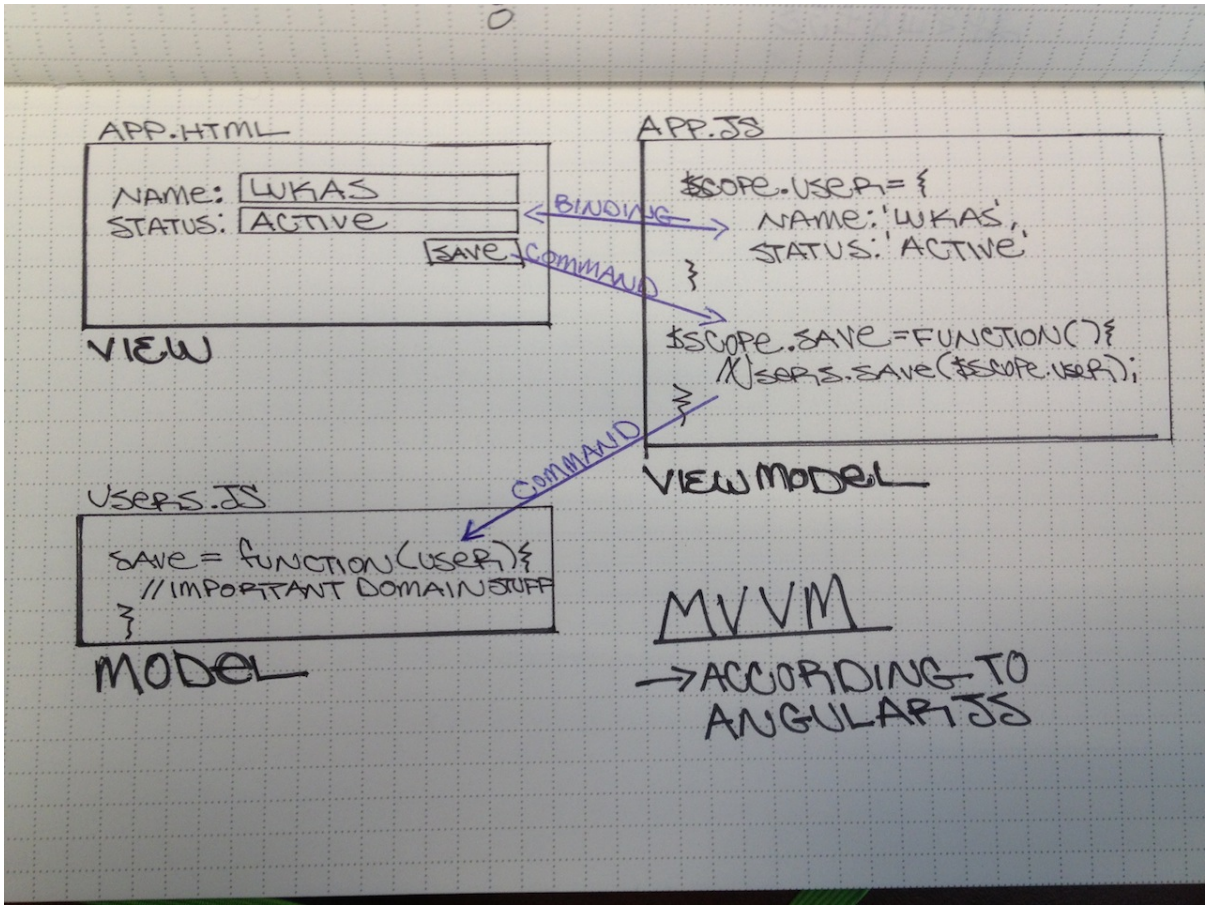

Figure 1.1 MVVM According to AngularJS

The MVVM pattern works really well for applications involving a rich user interfaces because the View is bound to the ViewModel and when the state of the ViewModel changes the View will update itself automatically. This is possible because we have two way databinding built into the framework.

In AngularJS, the View is simply your HTML with the AngularJS syntax compiled into it. Once the initial compilation cycle completes, the View can then bind to the `$scope` object which is essentially the ViewModel. We will get into this later but `$scope` is just a JavaScript object with some eventing attached to it to enable two way data-binding. You can also issue commands to the ViewModel to perform some operation as well.

We will elaborate more on each part later in the book but this should give you the general idea how AngularJS is constructed. This pattern creates a clean separation between the View and the logic driving the View. One of the most profound side-effects of the ViewModel pattern is that it makes your code very testable.

HTML TEMPLATES

Another core feature of AngularJS is that it uses HTML for templates. HTML templates are useful when you want to predefine a layout with dynamic sections to be filled in it is connected with an appropriate data structure. An example of this would be if you wanted to repeat the same DOM element over and over in a page such as a list or a table. You would define what you want a single row to look like and then attach a data structure, such as a JavaScript array, to it. The template would repeat each row for as many items in the array filling in each instance with the values of that current array item.

AngularJS has some really nice features that are going to make designer types feel right at home. It has a templating system that is not built on top of HTML but is HTML. It allows web designers to bring their current skill set to the table and start developing immediately. No new syntax. No pre-processing. Just HTML.

There are plenty of templating libraries out there but most of them require you learn a new syntax. This additional complexity can really slow down new developers. Furthermore, these other templates often need to be run through a pre-processor.

AngularJS templates are HTML and are validated by the browser just like the rest of the HTML in the page. There is some clever AngularJS functionality built in that allows you to control how your data structure is rendered within the HTML template. We will cover this in depth in a later chapter.

DEEP LINKING

AngularJS is a library for single-page apps, but there's a good chance your users won't even notice (besides being pleasantly surprised at the speed). While many modern web apps "break the back button," AngularJS makes it easy for you to update the URL of your application based on what the user is doing. Angular will use either the HTML5 history API, or fallback to hasbang URLs when the HTML5 history API is not available.

What this means for users is that they can bookmark and share application state. While this seems like a small convenience, it's actually immensely powerful in the age of social media. It also makes changing application state easy for you, the developer -- you can use good 'ol hyperlinks to navigate the user around your application.

DEPENDENCY INJECTION

Dependency injection (DI) is a really fancy phrase that describes a technique you have been using from when you first started programming. If you have ever defined a function that accepts a parameter you have leveraged dependency injection. You are injecting something that your function depends on to complete its work. It is that simple.

DI encourages you to write code that clearly describes its dependencies. This is handy because it makes your code to test and debug. It's easier to test because you can pass in a replacement for any dependency, as long as it provides the same interface. This could be a real object that you have constructed or a mock object that simulates the object. It is easier to debug because it encourages you to write small units of code which is easier to step through because of its granular nature.

In AngularJS, in most cases, gaining access to a particular component in the application is simply a matter of declaring it is a variable of the method you are using to define the element that needs that component.

Let's look at a simple case where this behavior can help us reduce ordering constraints, or the requirement that we explicitly define when parts of the application get loaded when. One simple case is that we conditionally want to swap out one function's implementation for an implementation in a different file. Consider this code:

Listing 1.5 Defining functions without DI

```
function a () {
    return 5;
}
function b () {
     return a() + 1;
}
console.log(b());
```

Is there some way to load a file either before or after we load the above to swap out the implementation of function a? Maybe, but it takes some tricks. You've also introduced another level of coupling: ordering constraints. Functions cannot easily be moved around. This will make it harder and harder to add new parts to an app as you write it. Now consider this code using DI:

Listing 1.6 Defining functions with DI

```
service('a', function () {
    return 5;
});
service('b', function (a) {
    return a() + 5;
});
service('main', function (b) {
    console.log(b());
});
```

There are several practical benefits to this change. The first is that functions can be moved around in any order. This may seem small, but for larger applications it can save valuable time. Next, you can easily override any of the functions, like this:

Listing 1.7 Overriding with DI

```
// swap out 'b' easily by redefining it in another file:
service('b', function (a) {
    return 1; // this is a test value
});
```

This is important in unit testing, or any instance where you want to have a drop-in replacement for some part of your code.

DIRECTIVES

Directives are our favorite part of AngularJS because it allows you to extend HTML to do some really powerful things. By "some really powerful things" we mean "pretty much anything you can imagine."

You can create custom DOM elements, attributes, or classes that attach functionality that you define in JavaScript. If you've worked with Flex before, you might recognize that this feature is similar to how you would define your behavior in an ActionScript class, then instantiate the class in MXML. HTML is excellent for declaring layout but other than that it is inherently dumb. Directives provide the perfect way to merge the declarative nature of HTML and the functional nature of JavaScript into one place.

Let's say you're writing an application that has a dropdown. Normally, you have some weird, difficult to remember class hierarchy like this:

Listing 1.8

```
<div class="container">
  <div class="inner">
    <ul>
      <li>Item
        <div class="subsection">item 2</div>
      </li>
    </ul>
  </div>
</div>
```

You can use a directive to make a shortcut so you only have to type:

Listing 1.9

```
<dropdown>
  <item>Item 1
    <subitem>Item 2</subitem>
  </item>
</dropdown>
```

But more than that, you can attach animations, behaviors, and more to this dropdown, and have all of these features included in your app just by writing `<dropdown></dropdown>`.

With our core intact, we want to point out what the benefits are of AngularJS!

1.1.2 Benefits

With our core in tact, we want to point out what the benefits are of AngularJS. We couldn't introduce you to Angular without showing you how it can help you in your day-to-day work. By looking at how Angular can help you, you'll be able to see where could you start using it, and how to leverage these strengths in your projects.

TESTABLE

How do you REALLY know your app works? The answer is "by writing tests," but it's often more complicated than that.

Remember when we said that AngularJS has distinct parts? One of the main driving forces between dividing the responsibilities between parts like this is so

that they would be easy to test in isolation. Being able to test parts of your app in isolation means that if something does break, you can quickly pinpoint exactly where.

Tests mean you can refactor, add new features, or make optimizations with confidence that everything still works. Web standards and browsers are moving fast these days. Features pop in and out, specs and APIs change. Tests mean you can ensure that your app works with the latest versions of your user's web browsers, even if you don't religiously follow every change to every browser.

If testing is easy, you're more likely to do it. In turn, this makes your application easier to maintain. AngularJS makes it easy to write unit tests. Mocking out parts of your application, you can test each feature in isolation. Angular also makes it easy to write integration tests that ensure your whole stack is functioning as expected.

As we walk through example applications, we'll show you how to write tests for each part.

EMBEDDABLE

AngularJS is not an "all or nothing" proposition. Imagine you have an existing website that is built entirely on jQuery and you want to use AngularJS on just single feature in the page, this is not a problem at all. It is easy to embed AngularJS into an existing website with as much surface area as you wish.

You can simply attach the AngularJS application a specific DOM element and not have to worry about nasty side effects outside of that element and into the rest of the page. This makes it very easy to get started with AngularJS and sets up a nice path to refactor legacy applications into something more manageable.

JUST JAVASCRIPT

AngularJS embraces JavaScript. While this may seem like an obvious thing for a JavaScript framework to do, other frameworks sometimes try to abstract the JavaScript away. In practice, this means you don't have to memorize pages and pages of APs or extend some deep class hierarchy to reap the benefits of AngularJS. Just plain old JavaScript here. This makes your code easy to test, maintain, reuse, and again free from boilerplate.

Because AngularJS is just JavaScript, you can easily use it with any language that compiles to JavaScript if you prefer. This includes CoffeeScript, TypeScript, ClojureScript, and any of the other compile-to-JavaScript languages yet to come.

A quick recap: AnguarJS has many benefits and features. If you are were to

pick three things to remember about it, they are data-binding, testability, and "just JavaScript." Or consider the old way of writing web-apps with event handlers and DOM mutations, and think of how much easier it would make your development process to use Angular instead.

1.2 Hello Angular

The traditional "Hello World" program is a bit too simple to show off AngularJS features. Instead, we're going to make a "Hello ___" program, where we have an input box that fills the blank in with whatever we type into the box.

This is what our "hello world" will look like:

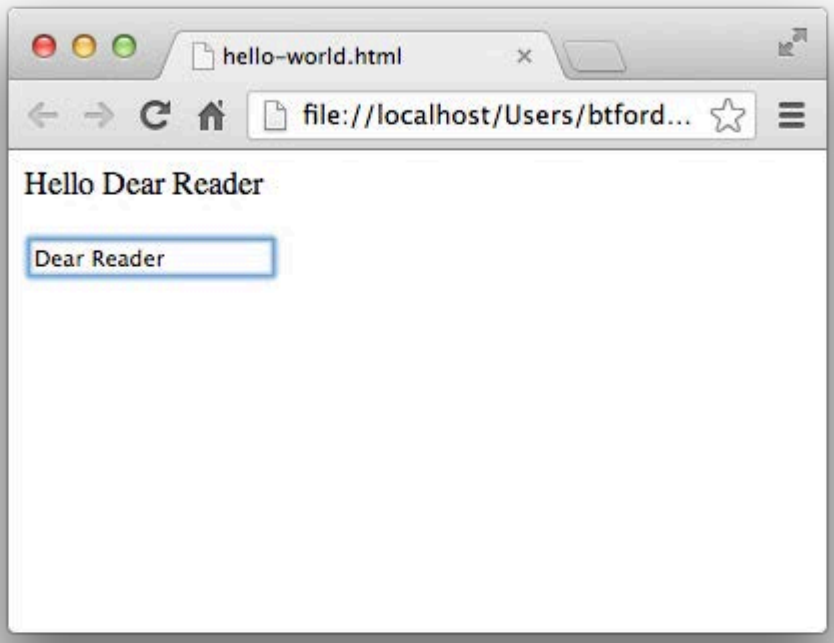

Figure 1.2 A screenshot of "Hello ___"

As mentioned before, as you type into the input box, The line "Hello ___" gets filled in with the contents of the input box. One thing that might surprise you coming from jQuery or another framework is that it changes letter-by-letter. This behavior is a direct result of AngularJS's data-binding.

1.10 shows what the code looks like for this "hello world."

Listing 1.10

```
<body ng-app="helloApp">
  <div ng-controller="HelloCtrl">
    <p>{{greeting}} {{person}}</p>
    <input ng-model="person">
  </div>
  <script>
    angular.module('helloApp')
      .controller('HelloCtrl', function ($scope) {
        $scope.greeting = 'Hello';
        $scope.person = 'World';
      });
  </script>
</body>
```

(1) These are organizational parts of Angular
(2) ng-model sets up data-binding to {{greeting}} and {{person}}

Note the special attributes in the HTML: "ng-app," "ng-controller," and "ng-model." These are all things that AngularJS uses to do its "magical" data-binding. The first line of JavaScript and the first two lines of HTML are organizational parts of Angular. We'll talk about how what controllers and modules do in the next chapter, but the short version is that they help organize and namespace your code.

The main thing we want to draw your attention to is `ng-model` and the "double curly brace" variables, like `{{greeting}}` and `{{person}}`. As you probably noticed, there's a correspondence between the second two lines of JavaScript, and the text between double curly braces. That is, the value of $scope.greeting gets put into `{{greeting}}`. In the case of `{{person}}`, the value is initially set to "world," but then constantly updated to reflect the value of the input box. This is thanks to the `ng-model` attribute of the input box.

Try running this example yourself. What happens when you create other `ng-model` input boxes?

1.3 Summary

Let's quickly recap. AngularJS is a framework used to build dynamic, single page web applications. The biggest feature AngularJS brings to the table is data-binding, which makes synchronizing application data and UI state fast and easy. The most important general benefits of using AngularJS are that it improves the organization and robustness of your app.

Now that we have covered the major features and benefits of AngularJS, we are going to spend the next chapter going over a very simple app. We are also going to

cover a few high level best practice consideration and take our first pass at the code.

See you in 3 - 2 - 1.