Introduction

This is just a small book, inspired by other free, short, and friendly programming texts such as Alex MacCaw's The Little Book on CoffeeScript (http://arcturo.github.io/library/coffeescript/index.html). I wrote this book to teach Angular the way I wish I had learned it, and for that reason it aims to be a good introduction to using the library in the context of typical, fast-paced web development work. Its approach is inspired by the Pareto principle (http://en.wikipedia.org/wiki/Pareto_principle): To give you access to a large part of Angular's power, while burdening you with only small part of its complexity. When you have finished it, although you may know just a fraction of all there is to learn about Angular, you should be able to use the framework to build powerful front-end applications with astonishing ease. Most importantly, this confidence will be built upon a solid foundation of actual experience. The book is itself a dynamic web site, in which a recent version of Angular is loaded and used to render every example. If you are inclined to active learning, the code examples are modifiable directly in-line, with changes to the output rendered on the fly. Tinkering and experimentation are strongly encouraged.

If you want to learn more about why I wrote this book, read on for a bit of background. The tale includes some of Angular's history, and is followed by a discussion of whether Angular is the right fit for your project. Otherwise, if you already know why you are here and are ready to get busy, you can skip ahead to the first chapter, Basics (/angular-basics/chapters/basics/).

Angular: hard or easy?

I have worked with implementations of Web MVC (http://en.wikipedia.org/wiki/Model-view-controller) for well over a decade, using everything from Struts (http://struts.apache.org/) to Spring MVC (http://projects.spring.io/spring-framework/) to Ruby on Rails (http://rubyonrails.org/) to Backbone (http://backbonejs.org/). I even wrote a book on Backbone and CoffeeScript (http://www.scriptybooks.com/books/backbone-coffeescript). Therefore, it seemed safe to assume that learning Angular would not be difficult. However, after digging into the API and its documentation, I found my progress blocked by an unfamiliar vocabulary that included terms like *transclusion*, *directive*, and *isolate scope*. As I read through the official documentation and tutorial, the prospect of easy mastery seemed to retreat into a fog. I remember in particular this passage (http://code.angularjs.org/1.0.7/docs/guide/directive#directivedefinitionobject):

> The advantage of transclusion is that the linking function receives a transclusion function which is pre-bound to the correct scope. In a typical setup the widget creates an isolate scope, but the transclusion is not a child, but a sibling of the isolate scope. This makes it possible for the widget to have private state, and the transclusion to be bound to the parent (pre-isolate) scope.

Today you will no longer find the passage above in the API guide, thanks to a serious effort by the Angular team to improve the accessibility of the project documentation. However, when I first read it, this passage left me feeling quite ignorant, an impression that gradually gave way to suspicion and concern. Was I entering new territory with this library? From nearly every side, I received the same message: Angular was a complex technology, and not one to be trifled with. In the commentary on Stack Overflow (http://stackoverflow.com/questions/tagged/angularjs), the AngularJS Google Group (https://groups.google.com/forum/#!forum/angular), and elsewhere, the minutiae of the framework was debated at length, with dire references to bugs, quirks, and pitfalls.

Like any self-respecting professional, I responded by rolling up my sleeves and battling forward. With time, I became comfortable with the vocabulary and concepts. I accepted that the learning curve was necessarily steep. That is, until the day I watched this video interview (http://www.youtube.com/watch?v=XoVsStcCCM8) with Miško Hevery, the creator of Angular. Listening to Hevery's account, I discovered a simple but important fact:

Angular was originally meant to be easy to use.

Hevery explained that he originally conceived Angular as a tool that non-programmers, working only in declarative markup, could use to build dynamic web applications. I realized that I had been exposed to many of Angular's details without gaining proportional benefit. It almost seemed as though the Angular project and its community had become engaged in a co-dependent relationship, in which challenge and complexity were supplied and consumed wantonly. A better approach would be to use the library more as Hevery had originally intended. To get behind a well-crafted set of controls, with the delicate internals safely hidden away, and *just drive*.

This approach turned out to be the right one for me. Although Angular is not the only tool I use for web development, there are situations in which it absolutely shines.

Is Angular right for you?

While client-side JavaScript tools like Angular make it easier than ever before to create single-page applications (SPAs), many web development projects are still best served by a more traditional approach that combines server-side rendering with "sprinkles" of jQuery (http://jquery.com/)-based interactivity. It's important to be honest about whether your project really needs to manage data and render HTML on the client, or whether you're stretching your requirements as an excuse to try something new and cool.

Also, Angular is a broad, opinionated solution. Although it is often referred to as a library, its built-in support for modules and dependency injection will dictate how you manage complexity in your project. You might prefer a different solution to Angular's, perhaps a more mature, *best-of-breed* solution, and will find it hard or impossible to replace Angular's. Also, unless your project is destined to reach the sort of scale that Google's engineers consider typical, you may never need the level of organizational support that Angular's core team takes for granted. Despite its value for unit testing, dependency injection has a complexity cost: See the Dependency Injection (/angular-basics/chapters/dependency-injection/) chapter for a concrete example.

Another problem arises when you want to bring in a non-Angular code and have it work with Angular's data binding or its rendering mechanism. This is when you must dive deeply into the arcane details, which can become a deep dive indeed. Contrast this with Backbone (http://backbonejs.org/), which sits lightly on top of jQuery and is relatively much smaller and easier to understand. If your project relies heavily on jQuery plugins, you may want to consider using Backbone to incrementally improve your application design.

Finally, Angular's fundamental approach of two-way binding between UI components and model objects appears to have limits with regard to application complexity. Developers at Facebook shared their frustration with two-way data binding (http://facebook.github.io/react/docs/flux-overview.html):

> We found that two-way data bindings led to cascading updates, where changing one object led to another object changing, which could also trigger more updates. As applications grew, these cascading updates made it very difficult to predict what would change as the result of one user interaction.

Although Facebook's React (http://facebook.github.io/react/) library is narrowly focused on rendering and therefore only a partial solution when compared with Angular, a client-side approach based on React is certainly worth evaluating.

Great reasons to use Angular

Angular is arguably the most popular JavaScript MV* solution in the world today, with almost 50,000 stars on GitHub currently, and probably many more as you read this. It has risen to this top spot from a very crowded field of solutions that is tracked and compared by the TodoMVC project (http://todomvc.com/). Let's look at some of the reasons for Angular's success.

Immediate productivity

If your project requires a reasonably complex client-side user interface for CRUD (http://en.wikipedia.org/wiki/Create,_read,_update_and_delete) operations on data, Angular handsomely delivers on its original promise of near-immediate productivity. With just a sprinkling of special attributes across ordinary HTML and a minor amount of JavaScript code, you can deliver web interactions that would take an order of magnitude more skill and effort to build using lower-level libraries.

Familiarity

Angular embraces the familiar world of plain old JavaScript and almost ordinary-looking HTML. I say *almost*, because Angular introduces new elements and attributes and some invasive-looking inline code. However, compared to many templating systems, Angular's templates stay very close to pure HTML. This makes it easy to understand for a majority of web developers.

Flexibility

Without much sacrifice in productivity, Angular fully embraces the current trend of UI-agnostic JavaScript frameworks. If you love using one of the popular UI frameworks such as Bootstrap (http://getbootstrap.com/), you can take advantage of the add-ons offered by third-party projects such as AngularUI (http://angular-ui.github.io/) for easy integration. So, whether you need to embellish a traditional, document-oriented web site with scattered interactivity, or to develop an ambitious, single-page web application, Angular is likely to work within your choices and constraints.

Future standards

While I am uncertain to what degree the design of Angular is prescient of the future, versus how much Angular's parent Google may be *creating* the future, it is clear that using Angular is a good way to familiarize oneself with proposed standards such as Web Components (http://www.w3.org/TR/components-intro/). In addition, because of Angular's alignment with upcoming JavaScript language features such as Object.observe (http://wiki.ecmascript.org/doku.php?id=harmony:observe), fundamental aspects of its approach are likely to be sustained rather than obsoleted.

Community

Of course, a valid reason to choose Angular over its competitors is simply that it is *really, really* popular. It is now used in countless high-traffic web sites; it is supported by Google; and it has spawned a broad ecosystem of add-ons and educational resources. However, the choice of the majority is not always right, and definitely not always right for everyone, all the time. It's important to think critically about what your project actually needs.

About this book

I got the idea for ScriptyBooks (http://www.scriptybooks.com/) several years ago, while reading an e-book about JavaScript programming on my MacBook Air. Wanting to run one of the examples, I downloaded an archive from the publisher's web site, puzzled over where to expand it, navigated through a hierarchy of directories, and finally opened the correct file in an editor. Then I realized I had to create an HTML document to hold the fragment. Even then it didn't run, and I went on a hunt for dependencies.

Why, if I was reading a book about a front-end technology on a computer, should I have to go through so much pain just to run the examples? A short time later I discovered the first edition of Marijn Haverbeke's book Eloquent JavaScript (http://eloquentjavascript.net/contents.html), and realized I could use his CodeMirror (http://codemirror.net) project to realize my own vision: An e-book that looked great, with beautiful type and a clean layout; but with editable, runnable code examples, right there in the text.

Therefore, all of the code in this book is presented in live, interactive editors. When it makes sense to display output for an example (or series of related examples), rendered HTML will appear just below the example in a sandboxed iframe.

expression.html

```
<strong>The lucky number {{3 + 4}}.</strong>
```

The lucky number 7.

I built this interactive experience using the fabulous CodeMirror (http://codemirror.net/) project--and Angular, of course! Go ahead, even if you don't know a thing about Angular, make some changes to the example above, and see what happens.

What's next

The first chapter of this book introduces you to working Angular code by gently exploring the basics of client-side templating and two-way bindings. Although the examples are so simple that most can be comprehended in a glance, even highly experienced Angular developers have confessed to taking away something of value from the chapter.

Next → (/angular-basics/chapters/basics/)

I hope you have been enjoying Angular Basics (/).
You can join the mailing list (/#signup) to hear about updates to the book.
Please also let me know (mailto:quartzmo@scriptybooks.com) what you liked and what I can improve.
And please share the word using the social buttons below!