Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™

Professional AngularJS

Valeri Karpov, Diego Netto

Contents

List of Illustrations

INTRODUCTION

It’s an exciting time to be a JavaScript developer. Between the meteoric rise of server-side JavaScript’s open source community (100,000 packages on the NodeJS package manager as of October 2014—twice as many as in December 2013), the popularity of next-generation client-side frameworks like AngularJS, and the growing number of companies that build web tools based on full-stack JavaScript, JavaScript language skills are in high demand. Modern tools allow you to build sophisticated browser-based clients, highly concurrent servers, and even hybrid native mobile applications using a single language. AngularJS is quickly becoming the leading next-generation client-side web framework, enabling individuals, small teams, and large corporations to build and test phenomenally sophisticated browser-based applications.

WHAT IS ANGULARJS?

Within the rapidly growing JavaScript community, AngularJS burst onto the scene when it released version 1.0 in June 2012. Although a relatively new framework, its powerful features and elegant tools for structuring applications have made it the front-end framework of choice for many developers. AngularJS was originally developed at Google by testing engineer Misko Hevery, who found that existing tools, like jQuery, made it difficult to structure browser user interfaces (UIs) that needed to display large amounts of sophisticated data. Google now has a dedicated team developing and maintaining AngularJS and related tools. AngularJS also powers some active Google applications, ranging from the DoubleClick

Digital Marketing Platform to the YouTube app on the PlayStation 3. AngularJS's popularity is growing rapidly: As of October 2014, it powers 143 of the Quantcast Top 10k websites and is rapidly outpacing its closest rivals, KnockoutJS, ReactJS, and EmberJS.

What makes AngularJS so special? One particularly pithy expression borrowed from the `https://angularjs.org/` website describes AngularJS as enabling you to "write less code, go have beer sooner." The heart of AngularJS is a concept called *two-way data binding*, which enables you to bind Hypertext Markup Language (HTML) and cascading style sheets (CSS) to the state of a JavaScript variable. Whenever the variable changes, AngularJS updates all HTML and CSS that references that JavaScript variable. For instance, in the following code:

```
<div ng-show="shouldShow">Hello</div>
```

If the `shouldShow` variable is changed to `false`, AngularJS automatically hides the `div` element for you. There is nothing special about the `shouldShow` variable: AngularJS doesn't require you to wrap your variables in special types; the `shouldShow` variable can be a plain old JavaScript Boolean value.

Although two-way data binding is the basis for what makes AngularJS so useful, it's only the tip of the iceberg. AngularJS provides an elegant framework for organizing your client-side JavaScript in a way to maximize reusability and testability. In addition, AngularJS has a rich set of testing tools, such as Karma, protractor, and ngScenario (see Chapter 9), which are optimized for use with AngularJS. AngularJS's focus on testable structures and rich testing tools makes it a natural choice for mission-critical client-side JavaScript. Not only does it enable you to write sophisticated applications fast, it supplies tools and structure that make testing your application easy. As a matter of fact, Google’s DoubleClick team cited AngularJS’s “full testing story” as one of its six biggest reasons for porting its digital marketing platform to AngularJS. Here is a brief overview of some of the concepts that make AngularJS special.

Two-Way Data Binding

In many older client-side JavaScript libraries, like jQuery and Backbone, you are expected to manipulate the Document Object Model (DOM) yourself. In other words, if you want to change the HTML contents of a `div` element, you need to write imperative JavaScript. For example:

```
$('div').html('Hello, world!');
```

AngularJS inverts this paradigm and makes your HTML the definitive source for how your data is displayed. The primary purpose of two-way data binding is to bind an HTML or CSS property (for instance, the HTML contents or background color of a `div` element) to the value of a JavaScript variable. When the value of the JavaScript variable changes, the HTML or CSS property is updated to match. The opposite is also true: If the user types in an `input` field, the value of the bound JavaScript variable is updated to match what the user typed. For instance, the following HTML greets whoever’s name is typed in the input field. You can find this example in this chapter’s sample code as `data _ binding.html`: Simply right-click on the file and open it in your browser—no web server or other dependencies required!

```
<input type="text" ng-model="user" placeholder="Your Name">
<h3>Hello, {{user}}!</h3>
```

No JavaScript is necessary! The `ngModel` directive and the `{{}}` shorthand syntax do all the work. There is limited benefit to using AngularJS in this simple example, but, as you’ll see when you build a real application in Chapter 1, data binding greatly simplifies your JavaScript. It’s not uncommon to see 800 lines of jQuery spaghetti code reduced to 40 lines of clean DOM-independent AngularJS code thanks to data binding.

Scopes in the DOM

DOM scopes are another powerful feature of AngularJS. As you might have guessed, there is no free lunch with data binding; code complexity has to go somewhere. However, AngularJS allows you to create scopes in the DOM that behave similarly to scopes in JavaScript and other programming languages. This permits you to break your HTML and JavaScript into independent and reusable pieces. For instance, here’s the same greeting example from earlier, but with two separate scopes: one for greeting in English, the other in Spanish:

```
    <div ng-controller="HelloController">
      <input type="text" ng-model="user" placeholder="Your
Name">
      <h3>Hello, {{user}}!</h3>
    </div>
    <hr>
    <div ng-controller="HelloController">
      <input type="text" ng-model="user" placeholder="Su
Nombre">
      <h3>Hola, {{user}}!</h3>
    </div>

    <script type="text/javascript"
            src="angular.js">
    </script>
    <script type="text/javascript">
      function HelloController($scope) {}
    </script>
```

The `ngController` directive is one way to create a new scope, enabling you to reuse the same code for two different purposes. Chapter 4 includes a thorough overview of two-way data binding and a discussion of internal implementation details.

Directives

Directives are a powerful tool for grouping HTML and JavaScript functionality into one easily reusable bundle. AngularJS has numerous built-in directives, like the `ngController` and `ngModel` directives you saw earlier, that enable you to access sophisticated JavaScript functionality from your HTML. You can write your own custom directives as well. In particular, AngularJS allows you to associate HTML with a directive, so you can use directives as a way of reusing HTML as well as a way of tying certain behavior into two-way data binding. Writing custom directives is beyond the scope of this introduction, but Chapter 5 includes a thorough discussion of the subject.

Templates

On top of two-way data binding, AngularJS lets you swap out entire portions of the page based on the state of a JavaScript variable. The `ngInclude` directive enables you to conditionally include *templates*, pieces of AngularJS-infused HTML, in the page based on the JavaScript state. The following example demonstrates a page with a `div` that contains different HTML based on the value of the `myTemplate` variable. You can find this example in `templates.html` in this chapter’s sample code:

```
<div ng-controller="TemplateController">
  <div ng-include="myTemplate">
  </div>
  <br>
  <a  ng-click="myTemplate = 'template1';"
      style="cursor: pointer"
      ng-class="{'selected': myTemplate === 'template1' }">
    Display Template 1
  </a>
```

```
      <a  ng-click="myTemplate = 'template2';"
          style="cursor: pointer"
          ng-class="{'selected': myTemplate === 'template2' }">
        Display Template 2
      </a>
    </div>

    <script type="text/javascript"
            src="angular.js">
    </script>
    <script type="text/javascript">
      function TemplateController($scope) {
        $scope.myTemplate = 'template1';
      }
    </script>
    <script type="text/ng-template" id="template1">
      <h1>This is Template 1</h1>
    </script>
    <script type="text/ng-template" id="template2">
      <h1>This is Template 2</h1>
    </script>
```

Chapter 6 includes a thorough discussion of AngularJS templates, including how to use them to structure single-page applications.

Testing and Workflow

Providing a framework for writing unit-testable code has been a core AngularJS goal from its first release. AngularJS includes an elegant and sophisticated dependency injector, and all AngularJS *components* (controllers, directives, services, and filters) are constructed using the dependency injector. This ensures that your code's dependencies are easy to stub out as necessary for your tests. Furthermore, the AngularJS team has developed numerous powerful testing tools, such as the Karma test runner and the protractor and ngScenario integration testing frameworks. These bring the sophisticated multibrowser testing infrastructure that was previously only feasible for large companies into the hands of the individual developer.

In addition, AngularJS's architecture and testing tools interface nicely with various open source JavaScript build and workflow tools, such as Gulp and Grunt. With these tools, you can execute your tests seamlessly, tie in tools like code coverage and linting into your test execution, and even scaffold entirely new applications from scratch. Core AngularJS is just a library, but the testing and workflow tools surrounding it make the AngularJS ecosystem as a whole an innovative new paradigm for building browser-based clients. Chapter 9 includes a more detailed discussion of the AngularJS testing ecosystem and the different types of testing strategies you can use for your AngularJS applications.

WHEN NOT TO USE ANGULARJS

Like any library, AngularJS is a perfect fit for some applications and a not-so-good fit for others. In the next section, you learn about several use cases in which AngularJS is a perfect fit. In this section, you learn about a few use cases in which AngularJS is not such a good fit and learn about some of AngularJS's limitations.

Applications Requiring Support for Old Versions of Internet Explorer

One limitation of AngularJS that may be significant for some users is that it doesn't support old versions of Internet Explorer. AngularJS 1.0.x supports Internet Explorer 6 and 7, but the version that you'll be learning about in this book, AngularJS 1.2.x, supports only Internet Explorer 8 and greater. Furthermore, the current experimental versions of AngularJS, 1.3.x, drop support for Internet Explorer 8 entirely. (They only support Internet Explorer 9 and greater.) If your application needs to support Internet Explorer 7, using AngularJS is probably not the right choice.

Applications That Don’t Require JavaScript Server I/O

AngularJS is an extremely rich and powerful library, and avid users are often tempted to use it for every application. However, there are many cases in which AngularJS is overkill and adds unnecessary complexity. For instance, if you need to add a button to a page that shows or hides a `div` element whenever a user clicks on it, using AngularJS cannot help you unless you need to persist the state of the `div` in the page’s URL or to the server. Similarly, choosing to write your blog in AngularJS is usually a poor decision. Blogs typically display simple data with limited interactivity, so AngularJS is often unnecessary. Also, blogs require good integration with search engines. If you were to write a blog in AngularJS, you would need to do some extra work (see Chapter 6) to make sure search engines could effectively crawl your blog, because search engine crawlers don’t execute JavaScript.

WHEN TO USE ANGULARJS

Now that you’ve learned about a couple of AngularJS’s limitations, you’ll learn about a few use cases in which AngularJS truly shines.

Internal Data-Intensive Applications

AngularJS is an extremely powerful tool for applications that need to display complex data in a browser UI, such as continuous integration frameworks or product dashboards. Much of the challenge in developing UIs for these applications lies in writing imperative JavaScript to render data correctly every time it changes. Two-way data binding frees you from needing to write this glue code, which results in much slimmer and easier-to-read JavaScript. As you’ll see when you write a stock market dashboard in Chapter 1, two-way data binding and directives make it easy to elegantly structure applications that need to display a lot of data.

Mobile Websites

AngularJS has extensive support for most common mobile browsers (Android, Chrome Mobile, iOS Safari). Furthermore, as you’ll see in Chapter 6, AngularJS has powerful animation support, and single-page apps enable you to leverage browser caching to minimize your bandwidth usage. This enables you to build mobile web applications that are fast and effectively mimic native applications. In addition, frameworks like Ionic (Chapter 10) enable you to build hybrid mobile applications, applications written in JavaScript but distributed through the Android and iPhone app stores, using AngularJS.

Building a Prototype

One theme that appears numerous times in this book is the idea of two-way data binding creating an effective separation between front-end JavaScript engineering and user interface/user experience (UI/UX) design. Two-way data binding enables the front-end JavaScript engineer to expose an application programming interface (API) that a UI/UX designer can then access in HTML, enabling both the front-end engineer and the designer to work in their preferred environments without stepping on each other’s toes. This is particularly useful for building out a prototype browser UI quickly, because you can then effectively parallelize tasks and enable your team to run more smoothly. In addition, AngularJS’s rich testing ecosystem enables you to ensure solid test coverage, and thus make sure your prototype doesn’t have any obvious bugs when you present it.

HOW TO USE THIS BOOK

Now that you’ve seen why AngularJS is such a popular library, next up is a brief overview of the contents of this book and how it can take you from writing beginner-level AngularJS to writing professional-level AngularJS.

You can think of this book as a “choose your own adventure” for learning AngularJS. If you are an AngularJS beginner, you will benefit a great deal from reading the book sequentially, as the chapters provide a logical sequence for learning AngularJS from scratch. However, the chapters and their examples are designed to be mostly independent of one another. If you are familiar with AngularJS and are looking to expand your knowledge in one particular area, such as using testing frameworks (Chapter 9), you can simply go to the appropriate chapter and skip the intermediate chapters. Some example code is shared between chapters, but each chapter explains each piece of example code under the assumption that you have never seen it before. Furthermore, some chapters reference information in other chapters, but they always provide a brief overview of the necessary concept. Whether you’re just getting started with AngularJS or you’re a more advanced user looking to learn about a specific topic, this book allows you to skip right to the most useful information. (However, if you are an AngularJS beginner, you should read Chapter 1 before skipping to other chapters.) Here are some brief highlights of what you can learn in each chapter.

Chapter 1: Building a Simple AngularJS Application

This chapter is geared toward readers who are new to AngularJS. You use AngularJS to build out a stock market dashboard application from scratch and get a high-level overview of the topics covered in subsequent chapters.

Chapter 2: Intelligent Workflow and Build Tools

In this chapter, you learn about the myriad open source tools for scaffolding new AngularJS applications, automating workflow, and including external dependencies. Special emphasis is placed on the popular scaffolding tool Yeoman, which enables you to quickly kick-start new AngularJS applications and provide powerful tools for managing your workflow.

Chapter 3: Architecture

This chapter offers an overview of best practices for structuring AngularJS components, including how to pass data between services, controllers, and directives. In addition, this chapter explores best practices for directory structures in applications of various sizes. Finally, this chapter covers two popular tools for managing file dependencies: RequireJS and Browserify.

Chapter 4: Data Binding

Although AngularJS data binding is elegant and intuitive, intermediate AngularJS developers often benefit from a deeper understanding of how data binding is actually implemented. This chapter explores how AngularJS scopes are structured and the implementation details of the `$digest` loop, so you can avoid common data binding pitfalls. This chapter also includes an overview of filters, including use cases and common mistakes.

Chapter 5: Directives

The first half of this chapter offers a basic working knowledge of how to write your own AngularJS directives and explores various use cases for directives. The second half focuses on designing more advanced directives using tools like transclusion.

Chapter 6: Templates, Location, and Routing

The primary purpose of this chapter is to supply an overview of how to write single-page applications in AngularJS, applications that allow a user to transition between multiple "views" without reloading the page. To build up to creating a single-page application, this chapter provides a detailed overview of AngularJS templates, the template cache, and the `$location` service. This chapter also provides an overview of using CSS3 animations with AngularJS and an example of how to make single-page applications search-engine-friendly using Prerender.

Chapter 7: Services, Factories, and Providers

This chapter provides a thorough description of the different methods of creating a service in AngularJS. You also learn how services work "under the hood" and how to take advantage of services' internal implementation.

Chapter 8: Server Communication

In this chapter, you use basic services and interceptors to create a login system. In addition, you learn how to bootstrap a simple back end using StrongLoop's Loopback API and integrate Facebook login with your client-side AngularJS application and your Loopback API.

Chapter 9: Testing and Debugging AngularJS Applications

This chapter includes a thorough overview of structuring unit tests and DOM integration tests (also known as *halfway tests*) for your AngularJS applications using the popular open source test runner Karma. This chapter also discusses the open source behavior-driven development (BDD) testing frameworks Mocha and Jasmine and explains how to run your tests in SauceLabs's browser cloud.

Chapter 10: Moving On

This chapter contains a brief overview of several popular open source modules that enable AngularJS to do some unexpected things. In particular, you learn how to integrate Twitter Bootstrap components using Angular-UI Bootstrap, how to build hybrid mobile applications with AngularJS and the Ionic framework, and how to integrate two popular open source JavaScript modules, Moment and Mongoose, with AngularJS. You also learn how to use ECMAScript 6 generators with AngularJS's `$http` service.

HOW TO WORK WITH THIS BOOK'S SAMPLE CODE

Each chapter in this book has its own sample code, available in the Code Downloads section at `http://www.wrox.com/go/proangularjs`. Each chapter starts with a reminder to visit this URL to download the sample code, so don't worry about bookmarking this exact page. Although each chapter includes code in the text as appropriate, it's best to download each chapter's sample code and try the examples for yourself.

This book's sample code has been designed to have a minimum of outside dependencies. The beginning of each chapter explains any special dependencies required for running its sample code. For many of this book's examples, you only require a modern browser. (The examples were primarily developed with Google Chrome 37 and Mozilla Firefox 32, but Internet Explorer 9 and Safari 6 should be sufficient.) These examples are in the form of `.html` files that you can open by right-clicking on the file and choosing to open the file in your browser using the `file://` protocol. For instance, to view the `data _ binding.html` example from this chapter's sample code, you may navigate to `file:///Users/user/Chapter%200/data _ binding.html` if this chapter's sample code is in the `/Users/user/Chapter 0` directory. You may safely assume that you can open any HTML file from this book's sample code in your browser without extra setup unless otherwise specified.

You don't require a special integrated development environment (IDE) for this book's sample code. Text editors like vim and SublimeText should be sufficient for experimenting with the sample code. You can use IDEs like WebStorm if you prefer, but there is limited benefit to using an IDE for this book's sample code.

Many of the concepts covered in this book require a web server to function properly. To make this process as lightweight as possible, this book utilizes NodeJS and the NodeJS package manager `npm` to start web servers. In addition, many of the tools you'll learn about in this book, like Grunt, Prerender, and Yeoman, are most easily installed through `npm`. To install NodeJS, you should go to http://nodejs.org/download and follow the instructions for your platform. NodeJS is easy to install and supports virtually every common desktop operating system (including Windows); furthermore, `npm` is automatically included with NodeJS. Most examples in this book that require NodeJS, however, assume that you are using a bash shell. Linux and OSX users can use their default terminals. On Windows, you should use git bash (http://msysgit.github.io), a bash terminal for Windows, if you want to run the command-line instructions as is. (Keep in mind, NodeJS does not officially support Cygwin, so using Cygwin is not recommended.) Each chapter explains how to install additional dependencies and reminds you to install NodeJS if necessary.

CONVENTIONS

To help you get the most from the text and keep track of what's happening, this book uses a number of conventions.

> **NOTE** *Notes, tips, hints, tricks, and asides to the current discussion are offset like this.*

As for styles in the text:

- URLs within the text are presented like so: https://angularjs.org/#!.
- Code shows up like this:

```
A monofont type is used for code examples.
Bold is used to emphasize code that is particularly important
in the present context or to show changes from a previous code
snippet.
```

ERRATA

Every effort has been made to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of the Wrox books, like a spelling mistake or faulty piece of code, please share your feedback. By sending in errata, you may save another reader hours of frustration; at the same time, you are helping to provide even higher-quality information.

To find the errata page for this book, go to http://www.wrox.com/WileyCDA/ and locate the title using the Search box or one of the title lists. Then, on the Book Search Results page, click the Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors.

NOTE *A complete book list including links to errata is also available at* http://www.wrox.com/WileyCDA/Section/id-105077.html.

If you don't spot "your" error on the Errata page, click the Errata Form link and complete the form to send the error you have found. The information will be checked and, if appropriate, a message will be posted to the book's errata page and the problem corrected in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at http://p2p.wrox.com. The forums are a web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At http://p2p.wrox.com, you will find a number of different forums that will help you not only as you read this book, but as you develop your own applications. To join the forums, just follow these steps:

1. Go to http://p2p.wrox.com and click the Register link.

2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you want to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

NOTE *You can read messages in the forums without joining P2P but in order to post your own messages, you must join.*

Once you join, you can post new messages and respond to messages that other users post. You can read messages at any time on the web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1
Building a Simple AngularJS Application

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Creating a new AngularJS application from scratch
- Creating custom controllers, directives, and services
- Communicating with an external API server
- Storing data client-side using HTML5 LocalStorage
- Creating a simple animation with ngAnimate
- Packaging your application for distribution and deployment using GitHub Pages

WROX.COM CODE DOWNLOADS FOR THIS CHAPTER

You can find the wrox.com code downloads for this chapter at `http://www.wrox.com/go/proangularjs` on the Download Code tab. For added clarity, the code downloads contain an individual directory for each step of the application building guide. The `README.md` file located in the root directory of the companion code contains additional information for properly utilizing the code for each step of the guide. Those who prefer to use GitHub can find the repository for this application, which includes Git tags for each step of the guide and detailed documentation, by visiting `http://github.com/diegonetto/stock-dog`.

WHAT YOU ARE BUILDING

The best way to learn AngularJS is to jump directly into a real-world, hands-on application that leverages nearly all key components of the framework. Over the course of this chapter, you will build StockDog, a real-time stock watchlist monitoring and management application. For the unfamiliar, a watchlist in this context is simply an arbitrary grouping of desired stocks that are to be tracked for analytical purposes. The Yahoo Finance API (application programming interface) will be utilized to fetch real-time stock quote information from within the client. The application will not include a dynamic back end, so all information will be fetched from the Yahoo Finance API directly or, in the case of company ticker symbols, be contained within a static JSON (JavaScript Object Notation) file. By the end of this chapter, users of your application will be able to do the following:

- Create custom-named watchlists with descriptions
- Add stocks from the NYSE, NASDAQ, and AMEX exchanges
- Monitor stock price changes in real time
- Visualize portfolio performance of watchlists using charts

StockDog will consist of two main views that can be accessed via the application's navigation bar. The dashboard view will serve as the landing page for StockDog, allowing users to create new watchlists and monitor portfolio performance in real time. The four key performance metrics displayed in this view will be Total Market Value, Total Day Change, Market Value by Watchlist (pie chart), and Day Change by Watchlist (bar graph). A sample dashboard view containing three watchlists is shown in Figure 1.1.

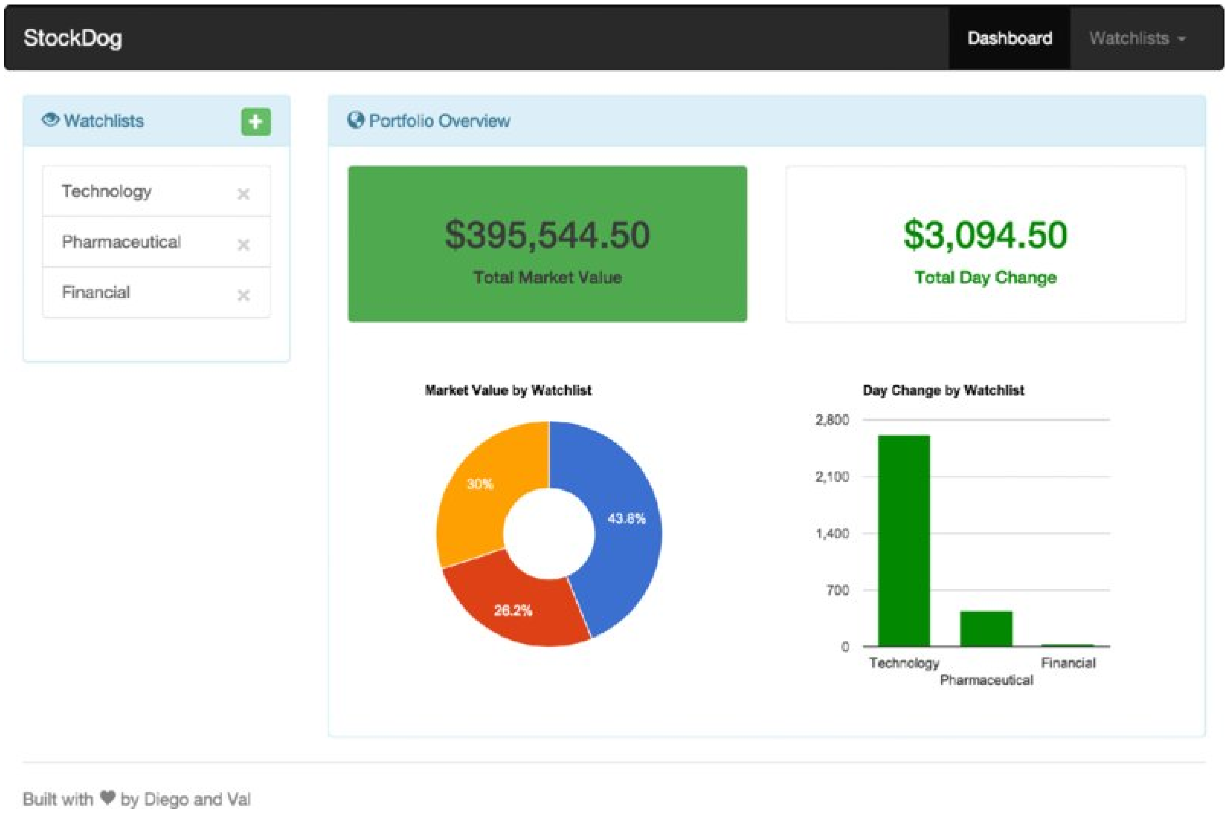
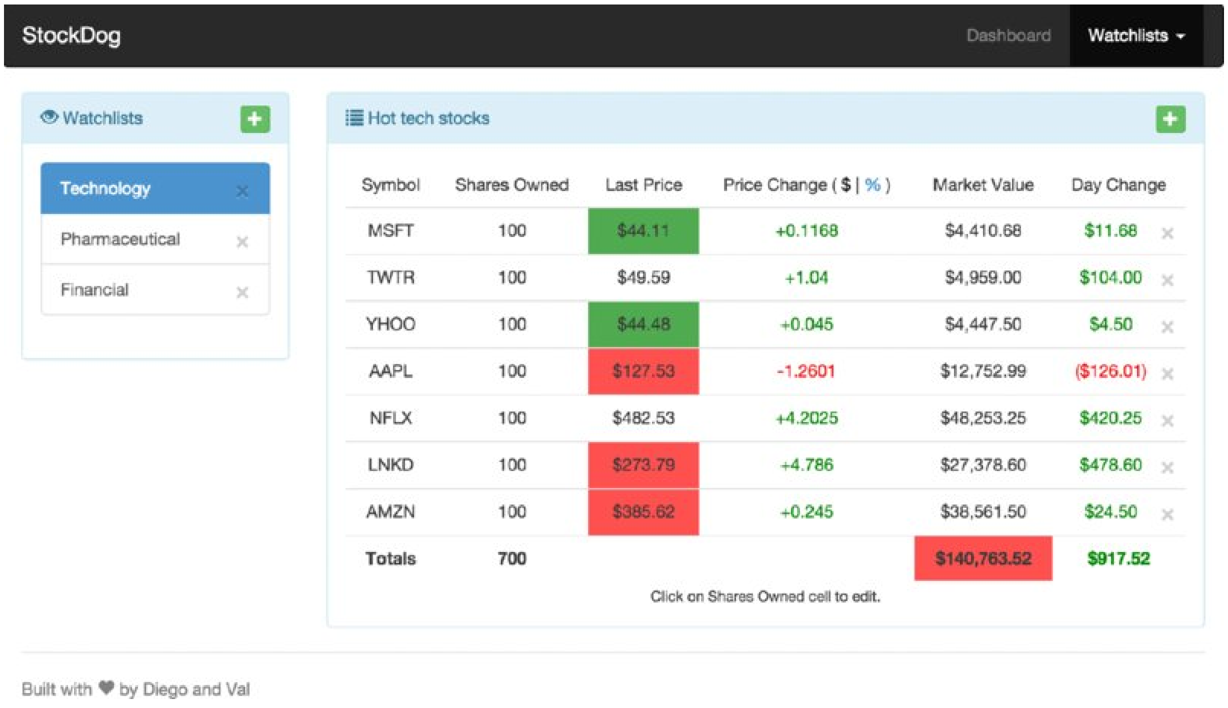


Figure 1.1

Each watchlist created in StockDog has its own watchlist view containing an interactive table of stock price information as well as a few basic calculations that assist in monitoring an equity position. Here, users of your application can add new stocks to the selected watchlist, monitor stock price changes in real time (during market hours), and perform in-line editing of the number of shares owned. A sample watchlist view tracking seven stocks is shown in Figure 1.2.



Symbol	Shares Owned	Last Price	Price Change ($ | %)	Market Value	Day Change
MSFT	100	$44.11	+0.1168	$4,410.68	$11.68
TWTR	100	$49.59	+1.04	$4,959.00	$104.00
YHOO	100	$44.48	+0.045	$4,447.50	$4.50
AAPL	100	$127.53	-1.2601	$12,752.99	($126.01)
NFLX	100	$482.53	+4.2025	$48,253.25	$420.25
LNKD	100	$273.79	+4.786	$27,378.60	$478.60
AMZN	100	$385.62	+0.245	$38,561.50	$24.50
Totals	**700**			**$140,763.52**	**$917.52**

Click on Shares Owned cell to edit.

Built with ♥ by Diego and Val

Figure 1.2

The process of building this application will be described over a series of 12 steps. Each step will focus on developing a key feature of StockDog, with AngularJS components introduced along the way, because they are needed to fulfill requirements defined by the application. Before beginning the construction of StockDog, it is important to establish a high-level overview of what you will be learning.

WHAT YOU WILL LEARN

The step-by-step guide included in this chapter will go beyond basic AngularJS usage. By implementing practical, real-world examples using the main building blocks of this framework, you will be exposed to most of the components provided by AngularJS, which will then be expanded upon in detail in subsequent chapters. It is important to keep this in mind because some of the features required by StockDog will utilize advanced concepts of the framework. In these cases, specific details on how the underlying AngularJS mechanism works will be omitted, but a high-level explanation will always be provided so that you can understand how the component is being utilized in the context of implementing the feature at hand. By the end of this chapter, you will have learned how to do the following:

- Structure a multiview single-page application
- Create directives, controllers, and services
- Configure `$routeProvider` to handle routing between views
- Install additional front-end modules
- Handle dynamic form validation
- Facilitate communication between AngularJS components
- Utilize HTML5 `LocalStorage` from within a service
- Communicate with external servers using `$http`
- Leverage the `$animate` service for cascading style sheet (CSS) animations
- Build application assets for production
- Deploy your built application to GitHub Pages

Now that the scope and high-level overview for StockDog have been discussed, you should have enough background and context to begin building the application. For those interested in viewing a working demonstration of StockDog immediately, you can find the completed application at `http://stockdog.io`.

STEP 1: SCAFFOLDING YOUR PROJECT WITH YEOMAN

Starting a brand new web application from scratch can be a hassle because it usually involves manually downloading and configuring several libraries and frameworks, creating an intelligent directory structure, and wiring your initial application architecture by hand. However, with major advancements in front-end tooling utilities, this no longer needs to be such a tedious process. Throughout this guide, you will utilize several tools to automate various aspects of your development workflow, but detailed explanations of how these tools work will be saved for discussion in Chapter 2, "Intelligent Workflow and Build Tools." Before getting started with scaffolding your project, you need to verify that you have the following prerequisites installed as part of your development environment:

- **Node.js**—`http://nodejs.org/`
- **Git**—`http://git-scm.com/downloads`

All the tools used in this chapter were built using Node.js and can be installed from the Node Packaged Modules (NPM) registry using the command-line tool npm that is included as part of your Node.js installation. Git is required for one of these tools, so please ensure that you have properly configured both it and Node.js on your system before continuing.

Installing Yeoman

Yeoman is an open source tool with an ecosystem of plug-ins called generators that can be used to scaffold new projects with best practices. It is composed of a robust and opinionated client-side stack that promotes efficient workflows which, coupled with two additional utilities, can help you stay productive and effective as a developer. Following are the tools Yeoman uses to accomplish this task:

- **Grunt**—A JavaScript task runner that helps automate repetitive tasks for building and testing your application
- **Bower**—A dependency management utility so you no longer have to manually download and manage your front-end scripts

You can find an in-depth discussion of Yeoman, its recommended workflow, and associated tooling in Chapter 2, "Intelligent Workflow and Build Tools." For now, all you need to do to get started is to install Grunt, Bower, and the AngularJS generator by running the following from your command line:

```
npm install –g grunt-cli
npm install –g bower
npm install –g generator-angular@0.9.8
```

NOTE *Specifying the* `-g` *flag when invoking* `npm install` *ensures that the desired package will be available globally on your machine. When you're installing* `generator-angular`*, the official AngularJS generator maintained by the Yeoman team, version 0.9.8 is specified. This should allow you to easily follow along with the rest of the guide, regardless of the current version. For any subsequent projects, it's highly recommended that you update to the latest version. You can do this by simply running* `npm install -g generator-angular` *once you have completed this chapter.*

Scaffolding Your Project

With all the prerequisite tools installed on your machine, you are ready to get started scaffolding your project. Thankfully, Yeoman makes this process quick and painless. Go ahead and create a new directory named `StockDog`, and