

Join the discussion @ p2p.wrox.com



Wrox Programmer to Programmer™

A close-up photograph of a running track's starting line. The line is white and textured, set into a dark grey asphalt surface. In the foreground, there is a red rectangular area with a repeating circular pattern, likely a book cover or advertisement. The title text is overlaid on this red area.

Professional AngularJS

Valeri Karpov, Diego Netto

PROFESSIONAL ANGULARJS

INTRODUCTION	xxv	
CHAPTER 1	Building a Simple AngularJS Application	1
CHAPTER 2	Intelligent Workflow and Build Tools	57
CHAPTER 3	Architecture	95
CHAPTER 4	Data Binding.	131
CHAPTER 5	Directives	157
CHAPTER 6	Templates, Location, and Routing.....	185
CHAPTER 7	Services, Factories, and Providers	217
CHAPTER 8	Server Communication	243
CHAPTER 9	Testing and Debugging AngularJS Applications	277
CHAPTER 10	Moving On	315
APPENDIX	Resources	345
INDEX	347	

PROFESSIONAL

AngularJS

PROFESSIONAL

AngularJS

Valeri Karpov
Diego Netto



Professional AngularJS

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2015 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-118-83207-3

ISBN: 978-1-118-83209-7 (ebk)

ISBN: 978-1-118-83208-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2014951014

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

*For my father, the elder Valeri Karpov,
who taught me to never settle for “good enough.”*

—VALERI KARPOV

*For my mother, Liliana, who showed me how to find
happiness by living each day like it might be your last.*

—DIEGO NETTO

ABOUT THE AUTHORS

VALERI KARPOV is a NodeJS Engineer at MongoDB, where he focuses on maintaining the popular Mongoose ODM and numerous other MongoDB-related NodeJS modules. In addition, he's a Hacker in Residence at BookaLokal, a blogger for StrongLoop, and the person who gave the MEAN stack its name. He has been running production AngularJS apps since AngularJS v0.9.4 in 2010. Most recently, he used AngularJS to build out BookaLokal's mobile website and a web client for MongoDB's internal continuous integration framework.

DIEGO NETTO is a software consultant and open source evangelist. He wears the many hats of a full stack engineer and entrepreneur. Owner of a development shop operating out of Los Angeles and Dallas, Diego creates web and mobile applications for both startups and enterprise companies. Maintainer of the IonicFramework Yeoman generator, he has most recently used AngularJS and the IonicFramework to build the Prop mobile app for www.aboatapp.com, and is using Famo.us/Angular to build the mobile app for www.modelrevolt.com.

ABOUT THE TECHNICAL EDITOR

STÉPHANE BÉGAUDEAU graduated from the Faculty of Sciences and Technology of Nantes and is currently working as a web technology specialist and Eclipse modeling consultant at Obeo in France. He has contributed to several open source projects in the Eclipse Foundation, and he is the leader of Acceleo. He also worked on Dart Designer, an open source tooling for the Dart programming language.

CREDITS

PROJECT EDITOR
Kelly Talbot

TECHNICAL EDITOR
Stéphane Bégaudeau

PRODUCTION EDITOR
Christine O'Connor

COPY EDITOR
Karen Gill

**MANAGER OF CONTENT DEVELOPMENT
& ASSEMBLY**
Mary Beth Wakefield

MARKETING DIRECTOR
David Mayhew

MARKETING MANAGER
Carrie Sherrill

**PROFESSIONAL TECHNOLOGY &
STRATEGY DIRECTOR**
Barry Pruett

BUSINESS MANAGER
Amy Knies

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Brent Savage

PROOFREADER
Nancy Carrasco

INDEXER
Johnna VanHoose Dinse

COVER DESIGNER
Wiley

COVER IMAGE
iStock.com/Manuel Faba Ortega

ACKNOWLEDGMENTS

SOCRATES WROTE THAT “Education is the kindling of a flame, not the filling of a vessel.” In that vein, I’m thankful that I have had teachers and mentors who made programming my lifelong passion rather than just a career. In particular, I’d like to thank Professors Kernighan and Tarjan from Princeton University; and Dr. Nevard, Dr. Sankaran, Mr. Scarpone, and Mr. Nodarse from the Bergen County Academies. In addition, I’d like to thank Misko Hevery, my mentor when I interned at Google and original author of AngularJS, who taught me more about software engineering in 12 weeks than I had learned in my life leading up to that summer.

—VALERI KARPOV

TO QUOTE THE SUCCESSFUL ENTREPRENEUR FELIX DENNIS, “Anyone not busy learning is busy dying.” This is a prudent reminder, especially in the constantly evolving field of software engineering, that in order to remain relevant and sustain success we must commit to a lifelong pursuit of knowledge. I would like to thank Professors Tatar and Ribbens from Virginia Tech for enlightening me to this realization. I also want to thank Addy Osmani for helping me discover the importance of intelligent tooling and for inspiring me to contribute to the open source community. Special thanks to my friend and co-author Valeri Karpov for getting me involved with AngularJS during such an early stage.

—DIEGO NETTO

CONTENTS

INTRODUCTION	xxv
<hr/>	
CHAPTER 1: BUILDING A SIMPLE ANGULARJS APPLICATION	1
What You Are Building	1
What You Will Learn	3
Step 1: Scaffolding Your Project with Yeoman	4
Installing Yeoman	4
Scaffolding Your Project	5
Exploring the Application	6
Cleaning Up	8
Step 2: Creating Watchlists	9
The Application Module	9
Installing Module Dependencies	10
Bootstrapping the Application	11
The Watchlist Service	11
The Watchlist-Panel Directive	13
Basic Form Validation	16
Using the Directive	18
Step 3: Configuring Client-Side Routing	19
The Angular ngRoute Module	19
Adding New Routes	20
Using the Routes	21
Template Views	22
Step 4: Creating a Navigation Bar	23
Updating the HTML	23
Creating MainCtrl	25
Step 5: Adding Stocks	26
Creating the CompanyService	27
Creating the AddStock Modal	27
Updating the WatchlistService	29
Implementing WatchlistCtrl	30
Modifying the Watchlist View	31
Step 6: Integrating with Yahoo Finance	32
Creating the QuoteService	33
Invoking Services from the Console	35

Step 7: Creating the Stock Table	36
Creating the StkStockTable Directive	36
Creating the StkStockRow Directive	37
Creating the Stock Table Template	39
Updating the Watchlist View	40
Step 8: Inline Form Editing	40
Creating the Contenteditable Directive	41
Updating the StkStockTable Template	43
Step 9: Formatting Currency	44
Creating the StkSignColor Directive	44
Updating the StockTable Template	44
Step 10: Animating Price Changes	46
Creating the StkSignFade Directive	46
Updating the StockTable Template	48
Step 11: Creating the Dashboard	49
Updating the Dashboard Controller	49
Updating the Dashboard View	52
Production Deployment	53
Conclusion	55
<hr/> CHAPTER 2: INTELLIGENT WORKFLOW AND BUILD TOOLS	57
What Can Tooling Do for Me?	57
What Is Bower?	58
Getting Started with Bower	58
Searching for Packages	58
Installing Packages	58
Versioning Dependencies	59
What Is Grunt?	60
Getting Started with Grunt	60
Installing Plug-Ins	62
Directory Structure	62
The Gruntfile	63
Configuring Tasks and Targets	64
The Connect Task	64
The Less Task	65
The JSHint Task	66
The Watch Task	68
The Default Task	69
Creating a Custom Task	69

What Is Gulp?	73
Getting Started with Gulp	73
Installing Plug-Ins	73
The Gulpfile	73
Creating Tasks	74
The Connect Task	74
The Less Task	75
The JSHint Task	77
The Watch Task	77
The Default Task	78
Arguments and Asynchronous Behavior	79
Gulp, Grunt, and Make	82
Automation Using Make	82
When to Use Make	84
When to Use Grunt	84
When to Use Gulp	84
What Is Yeoman?	84
Getting Started with Yeoman	85
Scaffolding a New Project	85
Exploring Plug-Ins and Tasks	85
load-grunt-tasks	86
time-grunt	86
grunt-newer	86
grunt-contrib-watch	86
grunt-contrib-connect	87
grunt-contrib-jshint	87
grunt-contrib-clean	87
grunt-autoprefixer	87
grunt-wiredep	88
grunt-contrib-compass	88
grunt-filerev	88
grunt-usemin	88
grunt-contrib-imagemin	89
grunt-svgmin	89
grunt-contrib-htmlmin	89
grunt-ng-annotate	90
grunt-google-cdn	90
grunt-contrib-copy	90
grunt-concurrent	90
grunt-karma	91

Alias Tasks and Workflow	91
serve	91
test	91
build	92
default	92
Modifications	92
Subgenerators	92
Popular Generators	93
angular-fullstack	93
jhipster	93
ionic	94
Conclusion	94
 CHAPTER 3: ARCHITECTURE	 95
Why Is Architecture Important?	95
Controllers, Services, and Directives	96
Controllers	96
Scope Inheritance	98
Event Transmission	99
The ModelService Paradigm	102
Services	104
Services Depending on Other Services	104
The event-emitter Module	105
Directives	107
Exposing API Using Controllers	108
Conclusion	109
Organizing Your Code with Modules	109
Directory Structure	113
Small Projects	114
Medium Projects	115
Large Projects	117
Module Loaders	119
RequireJS	119
Browserify	122
Best Practices for Structuring User Authentication	127
Services: Loading from and Storing Data to the Server	127
Controllers: Exposing an API to HTML	128
Directives: Interfacing with the DOM	128
Conclusion	129

CHAPTER 4: DATA BINDING	131
What Is Data Binding?	131
What Data Binding Can Do for You	134
Scoping Out AngularJS Scopes	136
Scope Inheritance	137
\$watch	140
\$apply	141
\$digest	141
Performance Considerations	142
An ngRepeat Gone Wrong	143
Filters and Data-Binding Gotchas	145
Use Case 1: Rules for Converting Objects to Strings	146
Use Case 2: Wrappers for Global Functions	150
Use Case 3: Manipulating Arrays	152
Conclusion	155
CHAPTER 5: DIRECTIVES	157
What Is a Directive?	157
Understanding Directives	158
An 80/20 Understanding of Directives	159
Writing Your Own Render-Only Directive	160
Writing Your Own Event Handler Directive	162
Writing Your Own Two-Way Directive	165
Beyond the Simple Design Patterns	167
A Deeper Understanding of Directives	167
Directive Composition Using Templates	167
Creating Separate Scopes for Directives	169
The First Way of Using the scope Setting	170
The Second Way of Using the scope Setting	171
The restrict and replace Settings	176
Moving On	179
Changing Directive Templates at Runtime	179
Transclusion	179
Using the transclude: true Setting	179
Using the transclude: 'element' Setting	182
The compile Setting, or compile Versus link	183
Conclusion	184

CHAPTER 6: TEMPLATES, LOCATION, AND ROUTING	185
Part I: Templates	187
Templating with ngInclude	188
ngInclude and Performance	191
Including Templates with script Tags	191
The \$templateCache Service	193
Next Steps: Templates and Data Binding	194
Part II: The \$location Service	196
What's in a URL?	196
Introducing \$location	197
Tracking Page State with \$location	198
Next Steps: Routing and SPAs	200
Part III: Routing	200
Using the ngRoute Module	202
The \$routeProvider Provider	203
The \$routeParams Service	205
Navigation in Your SPA	205
Search Engines and SPAs	207
Setting Up Prerender on the Server	207
The Google AJAX Crawling Spec	209
Configuring AngularJS for Search Engines	210
Search Engine Integration in Action	210
Introduction to Animations	211
The ngAnimate Module in Action	213
Conclusion	215
CHAPTER 7: SERVICES, FACTORIES, AND PROVIDERS	217
A Brief Overview of Dependency Injection	218
The \$injector Service	219
Function Annotations	220
Building Your Own Services	221
The factory() Function	222
The service() Function	224
The provider() Function	228
Common Use Cases for Services	232
Building a \$user Service	233
Building the \$stockPrices Service	234
Utilizing Built-In Providers	236
Custom Interpolation Delimiters	236
Whitelisting Links with \$compileProvider	237

Global Expression Properties with \$rootScopeProvider	240
Conclusion	241
CHAPTER 8: SERVER COMMUNICATION	243
Why Will I Learn?	243
Introduction to Promises	244
Services for HTTP Requests	246
\$http	247
Setting the HTTP Request Body	248
JSONP and Cross Site Scripting (XSS)	249
HTTP Configuration Objects	249
Setting Default HTTP Headers	250
Using HTTP Interceptors	251
The \$resource Service	259
Consuming the Twitter REST API	262
Scaffolding a REST API with StrongLoop LoopBack	264
Building a Simple API Using LoopBack	265
Creating a New Application	265
Creating a LoopBack Model	266
The API Explorer	266
Generating Resources with Loopback AngularJS SDK	267
Using Web Sockets with AngularJS	270
Using Firebase with AngularJS	273
Conclusion	275
CHAPTER 9: TESTING AND DEBUGGING ANGULARJS APPLICATIONS	277
AngularJS Testing Philosophy	277
The Testing Pyramid	279
Unit Testing in AngularJS	281
The Mocha Testing Framework	281
Unit Testing in the Browser with Karma	285
Browser Testing in the Cloud with Sauce	288
Evaluating the Unit Testing Options	292
DOM Integration Tests	292
A Guide to \$httpBackend	293
The Page You'll Be Testing	297
DOM Integration Tests with ng-scenario	298
DOM Integration Testing with Protractor	304
Evaluating ng-scenario and Protractor	309

Debugging AngularJS Apps	309
The debug Module	309
Debugging Using Chrome DevTools	311
Launching Developer Tools	312
Inspecting the State of the DOM	312
Using the Console Tab	312
Setting Breakpoints in the Sources Tab	313
Debugging Network Performance	314
Conclusion	314
CHAPTER 10: MOVING ON	315
Using Angular-UI Bootstrap	316
Modals	316
Datepicker	320
Timepicker	321
Custom Templates	321
Hybrid Mobile Apps with the Ionic Framework	325
Setting Up Ionic, Cordova, and the Android SDK	326
Using AngularJS in Your Ionic App	327
Yeoman Workflow and Building for Production	329
Icons, Splash Screens, and Cordova Hooks	330
Integrating Open Source JavaScript with AngularJS	331
Dates and Time Zones with Moment	331
Schema Validation and Deep Objects with Mongoose	335
AngularJS and ECMAScript 6	341
Using <code>yield</code> for Asynchronous Calls	342
Conclusion	343
APPENDIX: RESOURCES	345
INDEX	347

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.