

Dennis Sheppard
Christopher Miller
A. J. Liptak

Sams **Teach Yourself**

AngularJS for .NET Developers

in **24**
Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Dennis Sheppard
Christopher Miller
AJ Liptak

Sams **Teach Yourself**
AngularJS for
.NET
Developers **in 24**
Hours

SAMS

800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself AngularJS for .NET Developers in 24 Hours

Copyright © 2016 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33757-4

ISBN-10: 0-672-33757-6

Library of Congress Control Number: 2015910923

Printed in the United States of America

First Printing October 2015

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author(s) and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Editor-in-Chief

Greg Wiegand

Acquisitions Editor

Joan Murray

Development Editor

Keith Cline

Managing Editor

Sandra Schroeder

Project Editor

Seth Kerney

Copy Editor

Keith Cline

Indexer

Ken Johnson

Proofreader

Chuck Hutchinson

Technical Editor

Jesse Smith

Publishing Coordinator

Cindy Teeters

Book Designer

Mark Shirar

Compositor

codeMantra

Contents at a Glance

Introduction	1
HOUR 1 Introducing Modern Front-End Development	5
2 Presenting JavaScript Patterns	13
3 Tinkering with Tools for Modern Front-End Development	29
4 Mastering Modules	37
5 Covering Controllers	41
6 Understanding Views, Data Binding, and Event Handling	49
7 Discovering Services: Part I	63
8 Discovering Services: Part II	71
9 Using Built-In Directives	79
10 Conquering Custom Directives	89
11 Depending on Dependency Injection	103
12 Rationalizing Routing	111
13 Actualizing Application Organization	123
14 Figuring Out Filters	129
15 Approaching Angular Patterns	149
16 Making Components Communicate	155
17 Demonstrating Deployment	175
18 Unraveling Unit Tests	185
19 Destroying Debugging	199
20 Applying Angular to ASP.NET Web Forms	225
21 Applying Angular to ASP.NET MVC	247
22 Using Angular with ASP.NET Web API	265
23 Using Angular with ASP.NET SignalR	293
24 Focus on the Future	307
Index	319

Table of Contents

Introduction	1
HOURL 1: Introducing Modern Front-End Development	5
Why Does Everything Look So Different?	5
How Is the Code Different?	6
Options for Front-End Development	8
Why Angular?	9
Why Is Angular 1 Still Relevant?	10
Summary	10
Q&A	11
Workshop	11
Exercise	11
HOURL 2: Presenting JavaScript Patterns	13
Five JavaScript Tips and Tricks	14
JavaScript Patterns	17
Summary	26
Q&A	26
Workshop	27
Exercise	27
HOURL 3: Tinkering with Tools for Modern Front-End Development	29
The Tool Landscape	29
Node	30
Package Management	30
Summary	35
Q&A	35
Workshop	35
Exercise	36

HOOR 4: Mastering Modules	37
Angular Modules	37
Creating Modules	38
Summary	40
Q&A	40
Workshop	40
Exercise	40
HOOR 5: Covering Controllers	41
Angular Controllers	41
\$scope	42
Controller Inheritance	44
What Should and Shouldn't Go into a Controller	45
Best Practices	45
Summary	47
Q&A	47
Workshop	47
Exercise	47
HOOR 6: Understanding Views, Data Binding, and Event Handling	49
Angular Views	49
Data Binding	50
Two-Way Data Binding with ng-model	53
Data Binding Performance	55
Multiple Controllers in a View	58
Multiple Templates	58
Angular Event Binding	59
View Best Practices	60
Summary	61
Q&A	61
Workshop	62
Exercise	62
HOOR 7: Discovering Services: Part I	63
Angular Services	63
Service Versus Factory	64

Using Services	67
Summary	69
Q&A	69
Workshop	70
Exercise	70
HOOR 8: Discovering Services: Part II	71
Using Services for a DAL	71
Promises and \$q	76
Summary	78
Q&A	78
Workshop	78
Exercise	78
HOOR 9: Using Built-In Directives	79
Angular Directives	79
Built-In Angular Directives	81
Summary	86
Q&A	86
Workshop	87
Exercise	87
HOOR 10: Conquering Custom Directives	89
Why Custom Directives?	90
Link Function	92
Directive Scope	94
Element Versus Attribute	101
Summary	101
Q&A	101
Workshop	102
Exercise	102
HOOR 11: Depending on Dependency Injection	103
Inversion of Control and Dependency Injection	103
Dependency Injection in .NET	104
Using Angular's DI	106

Summary	108
Q&A	109
Workshop	109
Exercise	109
HOURL 12: Rationalizing Routing	111
Routing in a Single-Page App	111
Setting Up Routing with Angular	112
Cleaning Up your URLs with HTML5 Mode	117
Executing Code Before a Route Change with a Resolve	118
Summary	120
Q&A	120
Workshop	120
Exercise	121
HOURL 13: Actualizing Application Organization	123
Angular in a Single File	123
Breaking Your Application into Separate Files	124
Organizing Your Application by File Type	126
Organizing Your Application by Feature	126
Application Pro Tips	127
Summary	128
Q&A	128
Workshop	128
Exercise	128
HOURL 14: Figuring Out Filters	129
Examples of Formatting with Built-In Angular Filters	129
Create Your Own Angular Filter	132
Using a Filter to Search on ng-repeat	143
Summary	145
Q&A	146
Workshop	147
Exercise	147

HOOR 15: Approaching Angular Patterns	149
Design Patterns	149
Controller Patterns and Principles	150
Service Patterns and Principles	151
Angular Architecture Patterns	152
Summary	153
Q&A	153
Workshop	153
Exercise	154
HOOR 16: Making Components Communicate	155
Communication Between Components	155
Calling Directive Functions from a Controller	157
Using \$watch to Communicate	161
Using Events to Communicate	165
Nested Controller Communication	169
Summary	171
Q&A	171
Workshop	172
Exercise	173
HOOR 17: Demonstrating Deployment	175
Automating Deployment Tasks	175
Writing Production-Ready Angular Code	177
Error-Checking Your Angular Code	178
Minifying and Concatenating Your Angular Code	181
Deploying Your Angular Code to Microsoft Azure	182
Summary	183
Q&A	183
Workshop	184
Exercise	184
HOOR 18: Unraveling Unit Tests	185
Unit Testing in Angular	185
Karma and Jasmine	186

Getting Everything Set Up	186
Testing Controllers	189
Testing Services	191
Testing Directives	196
Summary	197
Q&A	197
Workshop	198
Exercise	198
HOURL 19: Destroying Debugging	199
Debugging Strategies and Tips	199
Tools	201
Summary	222
Q&A	222
Workshop	223
Exercise	223
HOURL 20: Applying Angular to ASP.NET Web Forms	225
Strategies for Modernizing Web Applications	225
Translating Web Forms into Angular	226
Augment Web Forms with Angular	239
Summary	245
Q&A	246
Workshop	246
Exercise	246
HOURL 21: Applying Angular to ASP.NET MVC	247
Building a Sample ASP.NET MVC App	247
Adding Angular to an ASP.NET MVC App	256
Summary	262
Q&A	263
Workshop	263
Exercise	264
HOURL 22: Using Angular with ASP.NET Web API	265
Single-Page Apps with Angular and Web API	265
Creating a Simple Web API	266

Consuming the Web API in Angular	273
Consuming Other APIs with Angular	285
Summary	291
Q&A	291
Workshop	292
Exercise	292
HOUR 23: Using Angular with ASP.NET SignalR	293
When Should You Use SignalR?	293
Configuring SignalR	294
Adding SignalR to the OWIN Pipeline.	294
Sending SignalR Messages in Your API	297
Receiving SignalR Messages in Angular	299
Summary	305
Q&A	305
Workshop	306
Exercise	306
HOUR 24: Focus on the Future	307
What's the Current Status of Angular?	307
What's Changing in Angular 2?	308
How Can You Prepare for Angular 2?	309
ES6	311
Web Components	312
Why the First 23 Hours of This Book Are More Important Than This Last Hour	315
Summary	316
Q&A	316
Workshop	316
Exercise	317
Index	319

About the Authors

Dennis Sheppard began his development career more than 20 years ago on an Apple IIe writing BASIC programs that printed “Hello!” an infinite number of times. It wasn’t quite love at first sight, but it was close enough. Several years later, after graduating from Louisiana Tech University with a computer science degree, Dennis got all professional with front-end development using ExtJS and .NET. Since then, he’s worked with a plethora of technologies, including a handful of JavaScript frameworks. He’s quite smitten with AngularJS and the roller coaster ride that is being a front-end developer. Dennis is a Microsoft Certified Solutions Developer and has delivered enterprise solutions for the private equity, insurance, healthcare, education, and distribution industries. Dennis is now the Front-End Architect at NextTier Education in Chicago, and lives in the suburbs with his wife, two kiddos, and a golden retriever.

Christopher Miller is an Architect in West Monroe Partners’ Technology practice. He received a B.S. with Highest Distinction in computer and information technology from Purdue University and started full-time at West Monroe Partners shortly thereafter. Beginning his career in the private equity space, he helped transform his client’s aging applications into modern web applications with the help of newer UI technologies such as HTML5 and jQuery.

He has moved on from investment management applications and is currently working on a Software-as-a-Service solution in the renewable energy space at West Monroe Partners. A Microsoft Certified Solutions Developer in Web Applications, his interests include multi-tenancy, RESTful API development, message-based architecture, Microsoft Azure, and of course, AngularJS and other front-end web technologies. Chris lives with his new wife, Hatlyn, in Chicago’s West Loop neighborhood.

AJ Liptak is a Senior Consultant at West Monroe Partners in the Technology practice, focusing on modern web applications. After earning his degree in computer information systems from Bradley University, he started at West Monroe Partners working in the banking, private equity, and distribution industries. He is a Microsoft Certified Solutions Developer and has recently provided transformative solutions for the telecom and healthcare industries. AJ lives in Chicago’s West Loop neighborhood, where he spends most of his free time riding his bike, trying new restaurants, and exploring cutting-edge technology.

Dedications

For my family, who inspires me to do crazy things like writing a book. —DS

To my new wife, Hatlyn, for putting up with me writing this book while planning our wedding. —CM

To my mom and dad for teaching me that hard work and perseverance are the keys to success. —AL

Acknowledgments

I'd like to thank my co-authors for helping me do something I never could have done alone in such a short amount of time. Thank you to Joan Murray at Pearson. Thank you to my former co-workers at WMP for making me a better developer or just supporting me (and in a lot of cases both!), particularly Kailin Johnson, Emily Bichler, Sarah Granger, Letteer Lewis, Ted Nubel, James Kinney, Rick Williams, Adam Kerr, Drew Leffelman, Andy Atteberry, Pamela Macario, Cory Chaplin, Ted Mazza, and Verite Pitts. Thank you Ryan Jones, Nate Tovo, Chris Jones, Jim Frantzen, and Karl Jackson for getting me started with front-end development in our ComScore days. Thank you for your support and/or teaching me things: Jim Lyman, Tom Dunlop, Alla Radunsky, Laura Foster, Fred Westrom, Clayton Chandler, William Bridges, John Smith, Mike O'Neal, and Chris Cunningham. Thank you Rundblades, Camerons, Hopkinseys, and Velasquezes for your encouragement. Thank you to my mom and dad for instilling in me a love of learning very early in my life. And finally, I want to thank Betsy, Violet, and Cameron Sheppard—for everything. —DS

Thanks to my co-authors and to Pearson for giving me this opportunity. Also, thanks to my friend and former co-worker, Brett Davis, for introducing me to front-end development while we were at Purdue. Also, thanks to other friends and former co-workers who helped me develop front-end training materials and who attended front-end conferences with me: Jared Jenkins, Kailin Johnson, Kevin Kinnebrew, and Letteer Lewis. Finally, thanks to Hatlyn, Mom, Dad, Kyle, and Kayla, who always inspire me, support me, and motivate me. —CM

A huge thanks goes out to my co-authors, without whom this book wouldn't have been finished. I'd like to thank all those who have made me the developer I am today, especially Kevin Kinnebrew, Letteer Lewis, and Umair Rashid for not letting me drown on my first web project. Finally, a special thanks goes out to Dan and Megan for being the best roommates anyone could ever have. —AL

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: consumer@sampublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

Congratulations! You're about to embark on an epic journey in front-end web development! Since its original release in 2009, AngularJS has become the most widely adopted JavaScript framework for building dynamic web applications. Angular empowers developers to organize, reuse, and test JavaScript code in ways few other JS frameworks have. The creators of Angular say that it is what HTML would have been, had it been designed for applications. You'll find out exactly what that means, and what the best ways are to use Angular to organize, reuse, and test JavaScript code.

Throughout this book, you will use your .NET background as a baseline to understanding new Angular concepts. There are a lot of parallels between Angular and .NET, and drawing on those comparisons should expedite your understanding of the framework. From models to modules and factories to filters, there are all sorts of new terms in the Angular world; this book will help you understand it all.

Audience and Organization

The title of this book says it's specifically for .NET developers. That's not entirely a lie, and if you've gotten past the title and are reading this, you're probably at least a little familiar with .NET. If you promise to tell all your non-.NET friends, here's a secret: You don't have to know .NET to get a lot out of this book. The goal of this book is to leverage a .NET developer's existing knowledge to more easily explain AngularJS concepts. If you don't have any existing .NET knowledge, however, just ignore the .NET parts. Other than Hours 20 through 23, which cover specific .NET topics, you'll be just fine.

It'd be great if you had some basic JavaScript knowledge. Hour 2, "Presenting JavaScript Patterns," is a refresher on some topics, but if you're completely new to JavaScript, you might want to check out a beginner's tutorial on it and then come back and try again.

How This Book Is Organized

This book is organized in a way that facilitates the gradual introduction of topics. AngularJS is known for having a steep learning curve, but the way this book builds on itself helps flatten that

curve. Once in a while, you might come across a concept that isn't fully explained until a later hour. In these cases, a callout (more on conventions in the next section) will point you to the hour where that topic is discussed in depth. Due to the gradual introduction of topics, you should absolutely read this book front to back, and do the exercises at the end of every hour. By the time you've finished with this book, you'll have a fully working application and a solid grasp on AngularJS. Enjoy the experience!

Conventions Used in This Book

This book uses several design elements and conventions to help you prioritize and reference the information it contains:

- ▶ **By the Way** boxes provide useful sidebar information that you can read immediately or circle back to without losing the flow of the topic at hand.
- ▶ **Did You Know?** boxes highlight information that can make your programming more effective.
- ▶ **Watch Out!** boxes focus your attention on problems or side effects that can occur in specific situations.
- ▶ **Go To** boxes call out other places in the book where we refer to the topic you're currently learning about.
- ▶ New terms appear in an italic typeface for emphasis.
- ▶ In addition, this book uses various typefaces to help you distinguish code from regular English. Code is presented in a monospace font.

Exercises

Each hour in this book ends with an exercise that tests your mastery of the skills learned in the hour. Most of these hours build upon one another toward developing a full front-end application. If you complete all these exercises by the end of the book, you will have built a reading list management application that enables users to save books to a reading list, remove books from their reading list, mark books as read, leave personal notes on books, and view book reviews for books on their reading list. Books saved to the application have these properties: author, title, description, genre, publication date, publisher, number of pages, average rating, price, and ISBN. To check your work from hour to hour, we have published the source code for the application at each point during development. Please see the book website to download this source code.

Onward and Upward!

Whether or not you're a .NET developer, you're going to have a lot of fun working through this book. By the time you complete it, you'll be able to develop, debug, and deploy Angular apps. You will have gained foundational knowledge on the motivations for "thick-client" front-end web development, and you will have learned countless Angular best practices. We love using Angular on our consulting projects, products, and side projects, and we hope you will as well.

This page intentionally left blank

This page intentionally left blank

HOUR 3

Tinkering with Tools for Modern Front-End Development

What You'll Learn in This Hour:

- ▶ The current tool landscape
- ▶ Node
- ▶ NPM
- ▶ Bower
- ▶ Grunt
- ▶ JSHint
- ▶ Yeoman and Angular scaffolding

Why tools? Well, in short, tools are awesome! They simplify our lives and make the overall development experience easier, quicker, and less prone to errors. The tools discussed this hour represent a set of great tools that on their own provide a huge amount of value, but when combined create a robust development experience to rival any other.

If you're coming from the .NET world, many of these tools will seem familiar, and if you're not, don't worry. All of these tools are fairly simple to use, and once you get the hang of them, you won't imagine web development without them.

The Tool Landscape

Before we dive into the tools, let's briefly discuss the current tool landscape. The front-end community is thriving and growing, which has brought about tremendous evolution in the past few years: but this has created a bit of an issue. Because the community is innovating so quickly, keeping up with the latest tools is becoming difficult. Also, these tools and libraries seem to burst on to the scene with tremendous force, bringing with them new and exciting ideas that quickly catch on, only to have a new tool innovate on top of the innovation. You might hear talk of the package manager wars, or the task manager wars. These "wars" are a result of the rapid innovation that the community is currently undergoing. Don't let this scare you, though; the results of these innovations are well worth the constant change.

Node

Node is a huge topic that can, and has, filled books in its own right. So, it's not covered much here, but it's important to know about. From the Node team: "As an asynchronous event-driven framework, Node.js is designed to build scalable network applications." Well, that sounds all fine and dandy, but what are you going to use Node for? The answer to this is, you won't really.

To clarify, almost all the tools covered this hour are built on top of Node or require Node in some way or another. Node is a highly flexible module-based platform built on top of the Google Chrome V8 engine and ECMAScript, which means that almost all of JavaScript's native functions, objects, and methods are supported. This has enabled developers to quickly build amazing tools using the language they are already comfortable with.

It is important to note that Node.js is a "server-side" framework that you can use for all sorts of applications, and the use of it in this book is to enable front-end development tools. For the uses covered in this book, install Node on your development machine and not on your web server.

To install Node, visit the Node website at <https://nodejs.org>.

Package Management

For those who have a .NET background, package management should already be familiar. If you are comfortable with the concept of package management, feel free to skip the next paragraph, although you might learn a thing or two even if you are already comfortable.

There is a saying that if you don't manage your packages, your packages will manage you. (In my experience, this is certainly true.) In the extremely innovative front-end community, libraries are constantly being updated, and many of these libraries rely on other libraries, which in turn rely on other libraries. This can cause a mess of dependencies that either forces you to spend hours resolving these dependencies or ultimately forces you to give up. This is where package managers come in. They maintain a list of libraries and their dependencies, and will help you automatically resolve these dependencies.

The first package manager discussed here is NPM.

NPM

The first thing you should know about NPM (Node Package Manager) is that's automatically installed when you install Node. So, take the next 10 seconds to celebrate not having to install anything for this topic.

NPM is by far one of the most common tools for web developers. Where Node allows developers to build great development tools, NPM allows the developers to share their libraries, while simultaneously making it easy for people to install and update. NPM is by far the largest

package manager for JavaScript libraries, boasting more than 125,000 packages at the time of this writing, with over a billion installs (that's right, a billion, with a B) per month.

NPM can be accessed via command line. So, if you are on a Windows PC, open the command prompt. If you're running OS X or Linux, open Terminal and type **npm**.

If you are experiencing errors, see the information about Node earlier (what it is and how to install it.).

Now that you have NPM up and running, let's install a package.

If you closed the command prompt or Terminal, open it again and run the following command:

```
npm install -g bower
```

Let's break this command down so that you better understand what it's doing. By typing `npm install`, you are invoking NPM and telling it to install a package called Bower. The `-g` flag is telling Node to install this package globally. The reason you install Bower globally is that it is not project dependent.

If you receive an error message when trying to install Bower, you may need to run the command with elevated privileges. On a Windows computer, try running the command prompt as an administrator. If you are on OS X or Linux, try running the command with the `sudo` modifier:

```
sudo npm install -g bower
```

That should prompt you for your password, and even though it doesn't look like you are typing anything, you are.

Follow these tips if you run into any problems installing packages throughout the remainder of this hour.

Bower

Bower is a package manager. Wait, didn't you just learn that was what NPM is? That's right! At this point, you might be thinking that the web development community just likes to complicate things (or that somebody is pulling your leg), and you would be partially right.

Bower is indeed another package manager, but one that focuses on installing front-end frameworks, libraries, and assets. While NPM can indeed install all of these things, it does so in a much different way. The following sidebar covers the differences between NPM and Bower.

BY THE WAY

Bower Versus NPM

As mentioned earlier, NPM and Bower are both package managers, so what's the point in having two? Doesn't that just get confusing? It certainly has caused and will continue to cause confusion

and arguments. This discussion does not take sides, and instead explains why there are two and what each one brings to the table.

The biggest difference between the two is the way they manage dependencies. NPM uses a nested dependency tree structure, which allows the developer of the package you are installing to request specific versions of libraries. This ensures that the package you are installing will work no matter what other versions of its dependencies you already have installed. This is really great for those writing server-side applications because size and download speed are not really issues. However, if you are developing front-end applications, making your end user download three separate versions of jQuery is simply unacceptable.

This is where Bower comes in. Bower uses a flat dependency structure, ensuring that your end user is downloading only one version of each library. This greatly reduces the number of files your users have to download, which in turn decreases the load time of your application. However, Bower can't ensure that when you install a new package that relies on one you already have it will work. This puts the burden of resolving dependencies on you, the developer.

Don't feel that you need to pick only one of these. In fact, you should use both. NPM for development dependencies like Bower, Grunt, Gulp, and so on, and use Bower for application dependencies such as jQuery and Underscore.

To get started with Bower, create a new folder somewhere easily accessible, and navigate to it using your operating system's terminal. Bower works just like NPM, where all you have to do is type **bower install**, but it can also grab files from other places. Here are some examples of ways to install libraries, assets, or even git repositories.

```
#### registered package
$ bower install jquery
#### GitHub shorthand
$ bower install desandro/masonry
#### Git endpoint
$ bower install git://github.com/user/package.git
#### URL
$ bower install http://example.com/script.js
```

Bower packages are installed in a subfolder called `bower_components` inside whatever folder you are currently in when you run the command.

Grunt

Grunt, at its core, is a task runner. Add a few of the hundreds of community-made plug-ins and you can automate your entire build process with just a few lines of code. Grunt uses Node's `require()` system, along with adding some things to the `package.json` file. So, make sure to install Grunt as follows:

```
sudo npm install -g grunt-cli
```

This installs the Grunt command-line interface (CLI) globally so that it's available to use in any of your projects. After you have it installed, you can start using it in your projects. To set it up,

first make sure that you have a `package.json` file in your project directory. If you don't, you can easily create one by answering a few questions after running the following command:

```
npm init
```

After answering questions from Node, you need to generate a Gruntfile, which is where you can set up tasks. To do that, run the following command:

```
npm install grunt --save-dev
```

The `--save-dev` flag tells the Grunt CLI that you want to make Grunt a development dependency.

WATCH OUT!

Installation Trouble!

If for some reason this command fails to generate a Gruntfile, visit <http://gruntjs.com/getting-started> to view the latest example Gruntfile.

This allows your fellow teammates, with just a single command, to make sure that they have all the necessary Node packages installed:

```
npm install
```

To add other premade Grunt tasks to your project, simply install them the same way you did Grunt. Here is the command to install JSHint, which is a code-validation task that ensures you write both valid and good JavaScript code:

```
npm install grunt-contrib-jshint --save-dev
```

Now that you have JSHint installed, let's take a look at your Gruntfile. It should look something like this:

```
module.exports = function(grunt) {

  grunt.initConfig({
  });

};
```

To configure JSHint and set up a task to run it, modify your Gruntfile like so:

```
module.exports = function(grunt) {

  grunt.initConfig({
    jshint: {
      files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
```

```
    options: {
      globals: {
        jQuery: true
      }
    }
  }
});

grunt.loadNpmTasks('grunt-contrib-jshint');

grunt.registerTask('default', ['jshint']);

};
```

This sets some options for JSHint, like what files it should examine and what globals you will have defined. Grunt then needs to make sure it loads the tasks, and finally, you register a task called default that will run JSHint. To actually run JSHint, just run the following:

```
grunt
```

This executes the default task that you just set up to run JSHint! While your IDE of choice may come with some built-in tools for front-end development, thousands of plug-ins can be used in any combination, providing you with the flexibility to automate your entire build. Grunt is an amazingly powerful task runner that can save you loads of time with your build and deployment process.

Yeoman

After reading through this hour, you might be thinking that it takes a lot of work to get all these tools set up for every new project you work on, and you wouldn't be wrong. This is where Yeoman comes in. Yeoman is a scaffolding tool that allows anyone to write plug-ins for it. It's simple to use and infinitely extensible. If you find yourself creating the same config files, installing the same packages, and scaffolding out the same folder structure every time you start a new project, you're going to love Yeoman. If this is your first foray into JavaScript programming, you could probably use a little jumpstart. Let's use Yeoman to scaffold out a brand new Angular application. To do so, let's first install Yeoman, as follows:

```
npm install -g yo
```

Just like when you use `-g` in the Grunt `install` command, this ensures that Yeoman is installed globally so that you can use it wherever you are.

Let's also run the following command to install the Yeoman Angular generator globally:

```
npm install -g generator-angular
```

At this point, make a new project folder and `cd` into it from your command prompt. Then run the following command:

```
yo angular MySweetNewAppName
```

Yeoman is going to ask you a few questions to make sure that it installs all the features you want. Once you complete all the prompts, Yeoman installs all the dependencies it needs and scaffolds a simple Angular application for you. To view what's created, run the following Grunt command:

```
grunt serve
```

This command starts the Angular application within a locally running Node server where you can play with the app and debug it.

Summary

In this hour, you learned about the ever-shifting landscape of JavaScript and the open source community, dug into Node and its super-useful NPM (which was then compared to its rising alternative, Bower). You also set up Grunt to help you run a code-hinting tool called JSHint, and installed Yeoman to help scaffold out an entire Angular application.

Q&A

- Q.** NPM and Bower do similar things; is there a competing tool to Grunt?
- A.** Yes! It's called Gulp, and is very similar to Grunt, but has different syntax and some extra features. It's really up to you which you choose to use, as the installation processes and setup are almost identical.

Workshop

Quiz

1. What command creates a new `package.json` file?
2. How do you save a dependency when using Bower?
3. What's the command for running the default Grunt task?

Answers

1. `npm init`
2. `bower install --save-dev`
3. `grunt`

Exercise

For the book management system mentioned in the Introduction, set up your project by running `npm init`, `bower init`, `bower install angular`, and `bower install bootstrap`.

This page intentionally left blank

Index

Symbols

- @ symbol in .NET MVC data binding, 50
- [WebMethods]s, modernizing ASP.NET Web Forms, 235-236
- { } (curly braces) in data binding, 50, 53, 56
- \$http
 - ASP.NET Web API and AngularJS, 279-280
 - requests, 76
 - services for DAL, 70-74
- \$inject property annotation and dependency injection, 108
- \$q and promises, 76-77
- \$resource
 - custom methods, creating, 76
 - services for DAL, 74-75
- \$routeProvider and routing, 113
- \$sce and Spotify API, 288
- \$scope and controllers, 42-46
- \$watch, communicating between controllers, 161-164
- :: (double colon) syntax in data binding, 56
- == (double equals) in JavaScript, 16

A

- Activate pattern (controller patterns), 150-151
- aliases and controllers, 51

Angular data binding, 50-51, 61

- ControllerAs syntax, 51-53
- controllers
 - aliases, 51
 - declaring, 51
- “dirty checking” 56
- double colon (::) syntax, 56
- one-time binding, 56
- performance, 54-56
- three-way data binding, 55
- two-way data binding, 53-54
- ViewModels, 51

Angular event binding, 58-61

Angular Mocks, 186

AngularJS, 9

- Angular 1, 10, 306-308, 314-316
- Angular 1.4, 306-308
- Angular 1.5, 308
- Angular 2, 10, 306, 308-309, 315-316
 - changes to, 308-309
 - component router, 309-311
- ControllerAs syntax, 311
- syntax, 315
- TypeScript, 311, 313-314
- web components, 312-315
- Angular in a single file, 121-124
- ASP.NET Web API, 273
 - CORS, 273-275
 - writing services, 275-285

- current status of, 306-308
- debugging strategies and tips, 198-200
- dependency injection system, 10
- DOM interaction, 10
- ES6, 308, 311-312
- folder structures, 38
- Front-End Development
 - AngularJS versus jQuery, 7-8
 - role in, 9
- Google and, 10
- jQuery versus, 6-8
- organizing folders, 38
- reasons for using, 9-10
- routing, 260
- SignalR messages, receiving, 297-301
- Spotify API, 285-291
- support for, 9-10
- testability, 10

annotation and dependency injection

- \$inject property annotation, 108
- implicit annotation, 106-107
- inline array annotation, 107-108

anti-forgery tokens (MVC), 251

API (Application Programming Interface)

- ASP.NET Web API, 264
 - AngularJS and, 273-285
 - single-page apps, 264-273

- back-end API, importance of, 6
 - SignalR, sending messages in API, 296-297
 - Spotify API and AngularJS, 285-291
 - third-party API authentication, 287
 - Application modules**
 - creating, 38-40
 - naming, 39
 - apps**
 - organizing
 - Angular in a single file, 121-124
 - by feature, 126-127
 - by file type, 126
 - LIFT principal, 127
 - into separate files, 124-125
 - single-page apps
 - ASP.NET Web API, 264-273
 - routing, 109-112
 - transaction ledger
 - application, building UI for, 137-144
 - architecture patterns, 152**
 - arrays (dependency), 40-42**
 - ASP.NET**
 - routing, 115
 - SignalR, 292
 - adding to OWIN pipeline, 294-296
 - configuring, 294
 - uses for, 292-294
 - ASP.NET MVC**
 - AngularJS in, 255-262
 - Razor View Engine and, 246, 256-260
 - sample app, building, 247-255
 - autogenerated front-end files, 249
 - MVC anti-forgery tokens, 251
 - ASP.NET Web API, 264**
 - AngularJS in, 273
 - CORS, 273-275
 - writing services, 275-285
 - single-page apps, 264-266
 - building a simple API, 266-271
 - deploying, 273
 - in-memory caches versus persistence, 266-267
 - testing API, 271-273
 - ASP.NET Web Forms, modernizing, 223**
 - augmenting via AngularJS, 239-246
 - Local Storage, 242-243
 - read-only requests, 237
 - strategies and tips, 223-226
 - translating to AngularJS, 226-239
 - [WebMethods]s, 235-236**
 - ASPX designer files, 227**
 - Attribute directives, 80, 100-101**
 - authentication (third-party API), 287**
 - autogenerated front-end files in ASP.NET MVC sample app, 249**
 - autogenerated SignalR proxies, 301-302**
 - automating deployment tasks, 173-176**
 - error-checking production-ready code, 178-180
 - Grunt, 176
 - Gulp, installing, 176
 - production-ready code
 - concatenating, 180-182
 - linting, 178-180
 - minifying, 180-182
 - writing, 176-178
 - Azure (MS) deployments**
 - AngularJS code deployments to, 182-183
 - ASP.NET Web API, 273
- ## B
- Backbone.js, 8-9**
 - back-end API (Application Programming Interface), 6**
 - Batarang, debugging TodoMVC, 215-221**
 - best practices**
 - controllers, 45-47
 - patterns (AngularJS)
 - architecture patterns, 152
 - controller patterns, 150-151
 - design patterns, 147-150
 - service patterns, 151-152
 - views, 60-61
 - bindable properties in controller patterns, 150**
 - blink tag and directives, 92-93**
 - block scoping versus function scoping in JavaScript, 15-16**
 - Bower, 31-32**
 - built-in directives, 81**
 - ng-class, 81-83
 - ng-click, 85-86
 - ng-hide, 81
 - ng-if, 81
 - ng-model, 86
 - ng-repeat, 83-85
 - ng-show, 81

C**C#**

- closures, 18
- functions, overloading, 16-17
- as “strongly typed” language, 14

capitalization filters, 130, 132-133

chaining deferred calls, 77

Chrome (Google) and DevTools, 201-210

Class directives, 80

cleaning up URL (routing), 116-117

closures

- C# closures, 18
- JavaScript closures, 17-18

code

- inline server tags, 227
- production-ready code, 227
 - concatenating, 180-182
 - deploying to Azure (MS), 182-183
 - error-checking, 178-180
 - linting, 178-180
 - minifying, 180-182
 - writing via Gulp, 177-178

colon (:), double colon (::) syntax in data binding, 56

Comment directives, 80

component communication, 154

- controllers
 - \$watch, 161-164
 - calling directive functions, 157-161
 - multiple controllers in views, 154-157
 - nested controller communication, 169-171

events, 164-169

scope chain, 167

component routers (Angular 2), 309-311

concatenating

- AngularJS code, 180-182
- ASP.NET MVC, 181

Constant (services), 66

controller patterns, 150

- Activate pattern, 150-151
- bindable properties in controller patterns, 150
- ControllerAs syntax, 150

ControllerAs syntax, 51-53, 150, 311

controllers, 40

- \$scope and, 42-46
- aliases, 51
- best practices, 45-47
- communication between controllers
 - calling directive functions, 157-161
 - multiple controllers in views, 154-157
 - nested controller communication, 169-171
- ControllerAs syntax, 51-53, 150, 311
- declaring, 51
- defining, 40-42
- dependencies, 105
- dependency arrays, 40-42
- DOM connections, 42
- functions, adding to, 43
- inheritance, 44-45
- multiple controllers in views, 56-58
- MV* pattern (JavaScript), 26

MVC

- anti-forgery tokens, 251
- .NET MVC views versus Angular views, 47-50
- routing, 260
- nested controller communication, 169-171
- responsibilities of, 45
- scope chain, 167
- services in, 67
- single responsibility principal, 45
- unit tests, 189-191
- ViewModels, 51

CORS (Cross-Origin Resource Sharing)

- ASP.NET Web API and AngularJS, 273-275
- OWIN and, 295

Crockford, Douglas, 19

CSS files and template file organization, 60

curly braces ({ }) in data binding, 50, 53, 56

currency filters, 131-132, 135-137

custom directives, 89-90

- examples of, 90-91
- naming, 91

custom filters, creating

- capitalization filters, 132-133
- currency filters, 135-137
- date filters, 133-134
- string casing filters, 132-133

D

DAL (Data Access Layer) and services, 70-76

data binding, 61

- Angular data binding, 50-51
- ControllerAs syntax, 51-53
- controllers
 - aliases, 51
 - declaring, 51
- “dirty checking” 56
- double colon (::) syntax, 56
- inline server tags, 227
- .NET MVC binding, 50-51
- one-time binding, 56
- performance, 54-56
- three-way data binding, 55
- two-way data binding, 53-54
- ViewModels, 51

date filters, 131, 133-134**debugging**

- AngularJS, strategies and tips, 198-200
- Batarang, debugging
 - TodoMVC, 215-221
- DevTools, debugging TodoMVC, 201-210, 212-213
- Firebug, 201
- Firefox Toolbox, 201, 210-211
- JavaScript, strategies and tips, 198-200
- TodoMVC, 201
 - Batarang, 215-221
 - DevTools, 201-210, 212-213
 - Firefox Toolbox, 210-211
- Visual Studio, 200-201

deferred calls, chaining, 77**delegates (.NET development), 15****dependencies**

- controllers and, 105
- dependency arrays, 39-42
- dependency injection, 10, 104, 106

\$inject property

- annotation, 108

Front-End Development, 105-106

- implicit annotation, 106-107
- inline array annotation, 107-108

.NET development, 104-105

- routing and, 112-113
- unit tests, 184

deployment

- ASP.NET Web API, 273
- automating tasks, 173-176
 - concatenating production-ready code, 180-182
 - error-checking production-ready code, 178-180
- Grunt, 176
- installing Gulp, 176
- linting production-ready code, 178-180
- minifying production-ready code, 180-182
- writing production-ready code, 176-178
- Azure (MS) deployments, 182-183

design patterns (AngularJS), 147

- modules, 150
- separation of concerns design principal, 147

design patterns (JavaScript), 11, 17

- closures, 17-18
- IIFE, 20-21
- MV* pattern, 24-25
 - controllers, 26
 - models, 25
 - presenters, 26-27

ViewModels, 26

- views, 25-26
 - whatevers (*), 26
- promises, 21-23
- Pub/Sub pattern, 23-24
- revealing module patterns, 19-20

DevTools and debugging**TodoMVC, 201-210, 212-213****directives, 78, 80**

- Attribute directives, 80, 100-101
- blink tag, 92-93
- built-in directives list, 81
- Class directives, 80
- Comment directives, 80
- custom directives, 89-90
 - examples of, 90-91
 - naming, 91
- DOM manipulation directives, 81, 92
- Element directives, 80, 100-101
- examples of, 78-80
- functions, calling from controllers, 157-161
- libraries, 90
- link function, 91-92
- naming, 91
- ng-class, 81-83
- ng-click, 85-86
- ng-hide, 81
- ng-if, 81
- ng-model, 86
- ng-repeat, 83-85
- ng-show, 81
- ngView and routing, 114
- scope
 - isolated scope, 95-100
 - scope chain, 167

- scope.\$apply, 92
- sharing scope, 93-95
- ui-router, 114
- unit tests, 196-197

“dirty checking” 56

DOM (Document Object Model)

- AngularJS and DOM interaction, 10
- controllers, connecting to DOM, 42
- directives and, 92
- manipulation
 - directives, 81
 - link function, 91-92
- routing and single-page apps, 109-112

dot notation, naming Application modules, 39

double colon (::) syntax in data binding, 56

double equals (==) in JavaScript, 16

E

Element directives, 80, 100-101

elements

- attributes of, 60
- hiding from views, 81
- views
 - best practices in, 60
 - hiding from, 81

Ember.js, 9

equality in JavaScript, 16

equals (=) in JavaScript

- double equals (==) in JavaScript, 16
- triple equals (===) in JavaScript, 16

error-checking production-ready code, 178

- JSHint, 178-180
- JSLint, 178

ES6, 308, 311-312

events

- binding, 58-61
- communication between components, 164-169

extensibility, 7

F

Factory (services), 64-66

feature, organizing apps by, 126-127

files

- file type, organizing apps by, 126
- front-end files, autogenerated in ASP.NET MVC sample app, 249

filters, 128

- capitalization filters, 130, 132-133
- creating, 132
 - capitalization filters, 132-133
 - currency filters, 135-137
 - date filters, 133-134
 - string casing filters, 132-133
- currency filters, 131-132, 135-137
- date filters, 131, 133-134
- executing, 133
- overwriting, 135
- searching via, 144-146
- string casing filters, 130, 132-133

Firebase, 54

Firebug, debugging via, 201

Firefox Toolbox, debugging via, 201, 210-211

folder structures

- AngularJS, 38
- .NET development, 36-38

Font Awesome, 83

formatting filters, 128

- capitalization filters, 130, 132-133
- currency filters, 131-132, 135-137
- date filters, 131, 133-134
- overwriting, 135
- searching via, 144-146
- string casing filters, 130, 132-133

Front-End Development, 3, 6, 8

AngularJS

- jQuery versus, 7-8
- reasons for using, 9-10
- role of, 9
- autogenerated front-end files in ASP.NET MVC sample app, 249
- Backbone.js, 8-9
- back-end API, importance of, 6
- dependency injection, 105-106
- Ember.js, 9
- extensibility, 7
- jQuery, diminished role of, 6-7
- mobility, importance of, 6
- SPA, 3-6
- tools, 27-30
 - Bower, 31-32
 - Grunt, 32-34
 - Node, 30
 - NPM, 30-32
 - Yeoman, 34-35

functions

- C#, 16-17
- controllers, adding functions to, 43
- JavaScript functions, 16-17
 - function scoping versus block scoping, 15-16
 - objects as functions, 15
- overloading
 - C#, 16-17
 - JavaScript, 16-17

G**Google**

- AngularJS and, 10
- Chrome and DevTool debugging, 201-210

Grunt, 32-34, 176**Gulp**

- concatenating production-ready code, 180-182
- Grunt versus, 176
- installing, 176
- JSHint and, 179-180
- production-ready code
 - linting, 179-180
 - minifying, 180-182
 - writing, 176-177-178

H**hiding**

- elements from views, 81
- raw content in templates, 60-61

HTML5 history API, 117**HTML5 mode, cleaning up URL (routing), 116-117****\$http**

- ASP.NET Web API and AngularJS, 279-280

- requests, 76
- services for DAL, 70-74

I**IIFE, 20-21**

- production-ready code, writing via Gulp, 177

implicit annotation and dependency injection, 106-107**inheritance**

- controllers, 44-45
- prototypal inheritance, 53

\$inject property annotation and dependency injection, 108**inline array annotation and dependency injection, 107-108****in-memory caches versus persistence, 266-267****installing**

- Angular Mocks, 186
- Grunt, 32-33
- Gulp, 176
- JSHint, 33
- Karma for unit tests, 186
- ngRoute, 112-113
- Node, 30
- Yeoman, 34

Internet Explorer and DevTools debugging, 201-210, 212-213**IoC (Inversion of Control) design pattern, 102-104****isolated scope, 95-100****J****Jasmine unit tests, 186-187, 189-191****JavaScript, 11**

- closures, 17-18

- debugging strategies and tips, 198-200

design patterns, 11, 17

- closures, 17-18
- IIFE, 20-21
- MV* pattern, 24-27
- promises, 21-23
- Pub/Sub pattern, 23-24
- revealing module patterns, 19-20

double equals (==), 16**equality in JavaScript, 16****flexibility of, 14-15****functions**

- function scoping versus block scoping, 15-16
- objects as functions, 15
- overloading, 16-17

as “loosely typed” language, 14-15**new keyword, 19-20****objects**

- creating with new keyword, 19-20
- as functions, 15
- prototypal inheritance, 53

tips and tricks, 11-17**triple equals (===), 16****jQuery**

- AngularJS versus, 6-8
- Front-End Development, diminished role in, 6-7

JSHint

- Grunt and, 33-34
- Gulp and, 179-180
- installing, 33
- production-ready code, linting, 178-180

JSLint, linting production-ready code, 178

JSON (JavaScript Object Notation)

- ASP.NET MVC, 258-260
- ASP.NET Web API, 264-266

K**Karma unit tests, 186-188****Katana (MS), adding SignalR to OWIN pipeline, 294-296****Kovalyov, Anton, 178****L****legacy ASP.NET Web Forms, modernizing, 223**

- AngularJS
 - augmenting Web Forms via, 239-246
 - translating Web Forms to, 226-239
- Local Storage, 223-226
- read-only requests, 237
- strategies and tips, 223-226

[WebMethods]s, 235-236**LIFT principal, 127****link function and directives, 91-92****linting production-ready code, 178**

- JSHint, 178-180
- JSLint, 178

Local Storage, modernizing legacy ASP.NET Web Forms, 223-226**logger services, 66-68****M****Malcom, Dr. Ian, 6****mapping and routing**

- redirecting traffic, 114
- URL, 113-114

methods, creating via**\$resource, 76****Microsoft Azure**

- ASP.NET Web API
 - deployments, 273
- deploying AngularJS code to, 182-183

Microsoft Katana, adding SignalR to OWIN pipeline, 294-296**minifying**

- AngularJS code, 180-182
- ASP.NET MVC, 181

models

- MV* pattern (JavaScript), 25
- ViewModels, 26, 51

modernizing ASP.NET Web Forms, 223

- AngularJS
 - augmenting Web Forms via, 239-246
 - translating Web Forms to, 226-239
- Local Storage, 223-226
- read-only requests, 237
- strategies and tips, 223-226

[WebMethods]s, 235-236**modules, 36, 40**

- Application modules
 - creating, 38-40
 - naming, 39
- configuring, 39-40
- creating, 38-40
- defining, 36
- dependency arrays, 39
- design patterns, 19-20, 150
- responsibilities of, 36-38
- single responsibility principal, 36-38

multiple controllers in views, 56-58**multiple templates, 58****MV* pattern (JavaScript), 24-25**

- controllers, 26
- models, 25
- presenters, 26-27
- ViewModels, 26
- views, 25-26
- whatevers (*), 26

MVC (Model-View-Controller)

- anti-forgery tokens, 251
- .NET MVC views versus Angular views, 47-50
- routing, 260

N**naming**

- Application modules, 39
- directives, 91

nested controller communication, 169-171**.NET development**

- data binding, 50-51
- delegates, 15
- dependency injection, 104-105
- event binding, 59
- folder structures, 36-38
- .NET MVC views versus Angular views, 47-50
- organizing folders, 36-38
- routing, 116
- routing URL, 113-114

.NET user controls, 161**new keyword (JavaScript), 19-20****ng-class, 81-83****ng-click, 85-86****ng-hide, 81****ng-if, 81****ng-model, 53-54, 86**

ng-repeat, 83-85, 144-146

ngRoute

installing, 112-113

parameters, 116

ng-show, 81

ngView, 114. *See also* ui-router

Node, 30

NPM (Node Package Manager), 30-32

O

objects (JavaScript)

creating with new keyword, 19-20

as functions, 15

one-time binding, 56

organizing

apps

Angular in a single file, 121-124

by feature, 126-127

by file type, 126

LIFT principal, 127

into separate files, 124-125

folders

AngularJS, 38

.NET development, 36-38

template file

organization, 60

overloading

C# functions, 16-17

JavaScript functions, 16-17

overwriting filters, 135

OWIN pipeline, adding SignalR to, 294-296

P

package management, 30

Bower, 31-32

Grunt, 32-34

NPM, 30-32

Yeoman, 34-35

Papa, John, 127, 151

patterns (AngularJS)

architecture patterns, 152

controller patterns, 150

Activate pattern, 150-151

bindable properties in controller patterns, 150

ControllerAs syntax, 150

design patterns, 147

modules, 150

separation of concerns design principal, 147

service patterns, 151-152

patterns (JavaScript), 11, 17

closures, 17-18

IIFE, 20-21

MV* pattern, 24-25

controllers, 26

models, 25

presenters, 26-27

ViewModels, 26

views, 25-26

whatevers (*), 26

promises, 21-23

Pub/Sub pattern, 23-24

revealing module patterns, 19-20

performance and data binding, 54-56

persistence versus in-memory caches, 266-267

Plunker, 275-285

postbacks, 229

Postman, testing ASP.NET Web API, 271-273

presenters, MV* pattern (JavaScript), 26-27

production-ready code

concatenating, 180-182

deploying to Azure (MS), 182-183

error-checking, 178

JSHint, 178-180

JSLint, 178

linting, 178

JSHint, 178-180

JSLint, 178

minifying, 180-182

writing via Gulp, 176-177

IIFE, 177

“use strict” statements, 177-178

promises (JavaScript), 21-23

\$q, 76-77

deferred calls, 77

prototypal inheritance, 53

Provider (services), 66-68

Pub/Sub pattern (JavaScript), 23-24

Q

\$q and promises, 76-77

R

Razor View Engine and ASP.NET MVC, 246, 256-260

redirecting traffic (mapping URL), 114

Require.js unit tests, 187
resolves and routing, 117-118
\$resource
 custom methods, creating, 76
 services for DAL, 74-75
Revealing Module pattern, 151-152, 279
\$routeProvider and routing, 113
routing
 \$routeProvider, 113
 AngularJS routing, 260
 ASP.NET routing, 115
 cleaning up URL, 116-117
 dependency injection, 112-113
 executing code before route changes, 117-118
 HTML5 history API, 117
 HTML5 mode, 116-117
 mapping URL, 113-114
 MVC routing, 260
 .NET routing, 116
 ngRoute
 installing, 112-113
 parameters, 116
 ngView, 114
 redirecting traffic, 114
 resolves, 117-118
 single-page apps, 109-112
 ui-router, 114

S

\$sce and Spotify API, 288
\$scope and controllers, 42-46
scope
 block scoping, 15-16

directives
 isolated scope, 95-100
 sharing scope, 93-95
 function scoping, 15-16
 inheritance in controllers, 44-45
 scope chain, 167
 scope.\$apply and directives, 92
searching via filters, 144-146
separation of concerns design principal (AngularJS design patterns), 147
Service (services), 67-68
services, 62
 \$http, 70-74, 76
 \$q, 76-77
 \$resource, 74-76
 Constant, 66
 controllers and, 67
 DAL, 70-76
 examples of, 62, 66-68
 Factory, 64-66
 logger services, 66-68
 Provider, 66
 Service, 64-68
 service patterns, 151-152
 singletons, services as, 64
 terminology, 64
 types of, 64
 unit tests, 191-196
 Value, 66
sharing scope in directives, 93-95
Shell pattern (architecture patterns), 152
SignalR, 292
 autogenerated SignalR proxies, 301-302
 configuring, 294

jquery.signalr, 299-300
 messages
 receiving in AngularJS, 297-301
 sending in API, 296-297
 OWIN pipeline, adding SignalR to, 294-296
 uses for, 292-294
single responsibility principal (design)
 controllers, 45
 modules, 36-38
single-page apps
 ASP.NET Web API, 264-266
 building a simple API, 266-271
 deploying, 273
 in-memory caches versus persistence, 266-267
 testing API, 271-273
 routing, 109-112
SPA (Single-Page Applications), 3-6
Spotify API and AngularJS, 285-291
startsWith() method, 145
Storage (Local), modernizing legacy ASP.NET Web Forms, 223-226
string casing filters, 130, 132-133
synchronous calls and promises (JavaScript), 21-23

T

templates
 file organization, 60
 hiding raw content, 60-61
 multiple templates, 58

testing

- AngularJS, 10
- ASP.NET Web API, 271-273
- unit tests, 184
 - Angular Mocks, 186
 - controllers, 189-191
 - directives, 196-197
 - Jasmine unit tests, 186-187, 189-191
 - Karma unit tests, 186-188
 - Require.js, 187
 - services, 191-196
 - setting up, 186-189

three-way data binding, 55**TodoMVC**

- debugging, 201
- Batarang, 215-221
- Developer Tools (Internet Explorer), 212-213
- DevTools, 201-210, 212-213
- Firefox Toolbox, 210-211

tools (Front-End Development), 27-30

- Bower, 31-32
- Grunt, 32-34
- Node, 30
- NPM, 30-32
- Yeoman, 34-35

transaction ledger application, building UI for, 137-144**triple equals (===) in JavaScript, 16****two-way data binding, 53-54****TypeScript and Angular 2, 311, 313-314****U****UI (User Input), building for transaction ledger application, 137-144****ui-router, 114. See also ngView****unit tests, 184**

- Angular Mocks, 186
- controllers, 189-191
- dependency injection, 184
- directives, 196-197
- Jasmine unit tests, 186-187, 189-191
- Karma unit tests, 186-188
- Require.js, 187
- services, 191-196
- setting up, 186-189

URL (Uniform Resource Locators)

- cleaning up (routing), 116-117
- mapping for routing, 113-114

“use strict” statements, writing production-ready code via Gulp, 177-178**V****Value (services), 66****ViewModels, 26, 51****views, 47, 61**

- Angular views versus .NET MVC views, 47-50
- best practices, 60-61
- communication between controllers
 - calling directive functions, 157-161
 - multiple controllers in views, 154-157
- elements, hiding, 81
- multiple controllers in views, 56-58
- MV* pattern (JavaScript), 25-26
- .NET MVC views versus Angular views, 47-50

Visual Studio

- ASP.NET MVC, building a sample app, 246-255
- debugging, 200-201
- legacy ASP.NET Web Forms, modernizing, 226-227

W**\$watch, communicating between controllers, 161-164****web components and Angular 2, 312-315****Web Forms (ASP.NET), modernizing, 223****AngularJS**

- augmenting Web Forms via, 239-246
- translating Web Forms to, 226-239

Local Storage, 223-226**read-only requests, 237****strategies and tips, 223-226****[WebMethods]s, 235-236****whatevers (*), MV* pattern (JavaScript), 26****writing services**

- ASP.NET Web API and AngularJS, 275-285
- factories versus services, 279

X - Y - Z**Yeoman, 34-35**