Community Experience Distilled

AngularJS UI Development

Design, build, and test production-ready applications in AngularJS

Amit Gharat Matthias Nehlsen

www.allitebooks.com

AngularJS UI Development

Design, build, and test production-ready applications in AngularJS

Amit Gharat

Matthias Nehlsen

BIRMINGHAM - MUMBAI

www.allitebooks.com

AngularJS UI Development

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: October 2014

Production reference: 1171014

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-847-2

www.packtpub.com

Cover image by Aniket Sawant (aniket_sawant_photography@hotmail.com)

www.allitebooks.com

Credits

Authors
Amit Gharat
Matthias Nehlsen

Reviewers
Ashutosh Das
Abhishek Dey
Anuj Gakhar
Steve Perkins

Commissioning Editor
Akram Hussain

Acquisition Editor
Sam Wood

Content Development Editor
Madhuja Chaudhari

Technical Editor
Indrajit A. Das

Copy Editors
Dipti Kapadia
Deepa Nambiar

Project Coordinator
Akash Poojary

Proofreaders
Simran Bhogal
Maria Gould
Ameesha Green
Paul Hindle

Indexers
Hemangini Bari
Rekha Nair
Tejal Soni

Production Coordinator
Shantanu N. Zagade

Cover Work
Shantanu N. Zagade

www.allitebooks.com

About the Authors

Amit Gharat is a full-stack engineer and open source contributor. He has built and made some of his personal projects open source, such as Directives, SPAs, and Chrome extensions written in AngularJS. He has an excessive urge to share his programming experiences in an easy-to-understand language through his personal blog in order to inspire and help others. When not programming, he enjoys reading, watching videos on YouTube, and watching comedy shows with his family. He has also written an article for *Appliness and Sdjournal Magazine*, Poland.

I would like to thank my family who have encouraged me to take up the challenge of writing this book. Huge thanks to my colleagues Saikumar Padamati and Akshay Ravindranath, for working hard on my behalf so that I could focus on my writing.

Matthias Nehlsen is a freelance software engineer and passionate open source contributor with around 15 years of experience in Information Technology. His current focus is on web applications, and he frequently works with AngularJS. He also founded the Hamburg AngularJS Meetup. You can find his open source projects on `https://github.com/matthiasn` and his blog at `http://matthiasnehlsen.com`. You can also follow him on Twitter at `@matthiasnehlsen`.

I would like to thank my friends and family for the support. Love you all. I would also like to thank my co-author and the team at Packt for making this book a reality after all.

www.allitebooks.com

About the Reviewers

Ashutosh Das, who hails from Bangladesh, works mainly as a backend developer, and his experience includes working with Django, Node.js, Laravel, and so on. He also likes to work with AngularJS. He spends his spare time writing for GitHub. He also works as a freelancer and has a part-time job. He is currently in the process of reviewing the following books:

- *AngularJS Web Application Development Blueprints*, *Packt Publishing*
- *AngularJS Testing Cookbook*, *Packt Publishing*
- *AngularJS By Example*, *Packt Publishing*

Abhishek Dey is a graduate student at the University of Florida, Gainesville, conducting research in the fields of Computer Security, Data Science, Big Data Analytics, Analysis of Algorithms, and Concurrency and Parallelism. He is a passionate programmer who started programming in C and Java at the age of 10 and developed a strong interest in web technologies when he was 15. He possesses profound expertise in developing high-volume software using C++, Java, C#, JavaScript, AngularJS, HTML5, Bootstrap, Hadoop MapReduce, Pig, Hive, and many more. He is a Microsoft Certified Professional, an Oracle Certified Java Programmer, an Oracle Certified Web Component Developer, and an Oracle Certified Business Component Developer.

Abhishek has contributed in bringing new innovations in the field of Highway Capacity Software Development at McTrans Center at the University of Florida (`http://mctrans.ce.ufl.edu/mct/`) in collaboration with the Engineering School of Sustainable Infrastructure and Environment (`http://www.essie.ufl.edu/`). In his leisure time, he loves to travel to different interesting places, or paint on canvas and give color to his imagination. He has also reviewed *Kali Linux CTF Blueprints*, *Packt Publishing*. To find out more about him, visit `www.abhishekdey.com`.

> I'd like to take this opportunity to thank Saptaparna Das, who has always been the biggest support in my life and has always shown me the right path.

www.allitebooks.com

Anuj Gakhar is a software consultant based in the UK and has over 14 years of experience in the industry. He currently works with technologies such as HTML5, CSS3, and JavaScript with a heavy focus on BDD and TDD, although he started off his career with server-side technologies such as ColdFusion and PHP. He has also worked on several enterprise projects using Flex and ActionScript 3.

You can find his blogs at `http://www.anujgakhar.com` and follow him on Twitter at `@anujgakhar`.

Steve Perkins is the author of *Hibernate Search by Example*, *Packt Publishing*, and has over 15 years of experience working with Enterprise Java. He lives in Atlanta, GA, USA, with his wife, Amanda, and their son, Andrew. Steve currently works as an architect at BetterCloud, where he writes software for the Google Cloud Platform.

When he is not writing code, Steve plays the fiddle and guitar and enjoys working with music production software. You can visit his technical blog at `http://steveperkins.net` and follow him on Twitter at `@stevedperkins`.

www.allitebooks.com

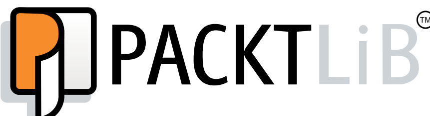
www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



http://PacktLib.PacktPub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

www.allitebooks.com

www.allitebooks.com

Table of Contents

Preface **1**

Chapter 1: Setting Up the Environment **7**

Hello World **8**
Using objects instead of primitives 11
Building our first directive 13
Installing Node.js and NPM **15**
OS X 15
Windows 15
Linux (Ubuntu) 16
Managing client-side dependencies with Bower **16**
Testing the Hello World application **19**
Unit tests 19
Installing Karma and Jasmine 20
Integration / end-to-end tests with Protractor 24
Building the application **28**
Running Protractor from Grunt 31
Managing the source code with Git **33**
Summary **35**

Chapter 2: AngularUI – Introduction and Utils **37**

Downloading AngularUI **38**
Building AngularUI-Utils **38**
Integrating AngularUI-Utils into our project **40**
uiMask directive **42**
Event Binder **46**
Keypress **50**
jQuery Passthrough **52**
Summary **56**

www.allitebooks.com

Chapter 3: AngularUI – Extended **57**
Embedding Google Maps **57**
Markers on the map 61
Event Binding 62
Managing application dependencies with Bower **63**
Modifying the .gitignore file 65
The calendar component **65**
Using a filter for date formatting 67
Styling the calendar 70
Adapting the controller 71
Adding tests 73
Testing the controller 74
Testing the filter 75
Building the application 76
Summary **77**
Chapter 4: Customizing and Exploring ng-grid **79**
Setting up the project **80**
Creating a service in AngularJS **81**
The simple grid view **82**
Grouping the grid **86**
Using a master/details view **88**
Summary **91**
Chapter 5: Learning Animation **93**
Setting up the project **94**
Creating our first animation – a simple to-do list **95**
Moving elements around on the page **101**
Easing functions 103
Using LESS to scale entire animations **104**
Using animate.css **106**
Staggering animations **108**
Understanding how staggering works 110
JavaScript-defined animations **110**
Summary **113**
Chapter 6: Using Charts and Data-driven Graphics **115**
Understanding the importance of charts **116**
Creating a bar chart 116
Making the bar chart data driven **118**
Converting the bar chart into a widget **122**
Creating a bar-chart directive 122

Using Angular Google chart tools **124**
Building a dashboard using the GitHub REST API **127**
Extending the dashboard 132
Summary **135**
Chapter 7: Customizing AngularJS with CSS and CSS Frameworks **137**
The evolution of responsive design **138**
Introducing media queries **139**
@media 139
The @media expression 140
Better designs with Twitter Bootstrap **142**
The foundation of your application **148**
Summary **155**
Chapter 8: AngularUI Bootstrap Development **157**
Why use AngularUI Bootstrap? **158**
Building a Project Management Application 158
Creating accordion 161
Creating tabs 164
Hiding less relevant content with collapse 169
Setting timelines with datepicker 172
Utilizing buttons 174
Converting priorities in the form of ratings 175
Notifying users with alert messages 176
Using carousel 177
A progress bar to show the status of an issue 181
Efficient suggestions with typeahead **182**
Common housing for application-specific menus with a dropdown **183**
Summary **185**
Chapter 9: Customizing AngularUI Bootstrap **187**
Introduction to external templates **187**
Loading a template via the script tag 188
Loading a template via $templateCache 189
Using an external template 189
Customizing the AngularUI Bootstrap pagination widget **190**
Extending the AngularUI Bootstrap tab widget **196**
Summary **202**

Chapter 10: Mobile Development Using AngularJS and Bootstrap 203
Why bother about mobile? 204
Building a bookmarking app with the mobile-first approach 204
Making the application dynamic 208
Allowing users to search through bookmarks 215
Crafting the application for mobile devices 217
Animation for better user experience 219
Mobile optimization for a better user experience 222
Periodic delay for tap events 222
Accelerated transitions and animations 222
Improving initial page load 227
Summary 235
Index 237

Preface

Many web development frameworks for single-page applications have come out over the years. AngularJS is not just another one of them though; it is different in important ways. Most importantly, it brings the fun back to client-side development.

How is AngularJS different? It is declarative, meaning that we, as developers, do not have to manually manipulate the document object model (DOM) in the browser. Instead, we describe how the data model is supposed to be rendered on a page, and then, let AngularJS handle the rendering on the page when the data model changes.

Through the two-way data binding, the data model updates automatically, for example, when we type text on a page or click on a button. Other UI elements that are also bound to the model will update. This might not sound like much before seeing it in action, but it indeed makes all the difference between massive amounts of jQuery DOM-manipulating code and a clean description of how the current application state is supposed to look on the page. In practice, this results in much cleaner and shorter code.

JavaScript is a very powerful language. It has it quirks though, and improper usage of the language often results in messy code that is really difficult to understand and very error prone. This is not so much with AngularJS. It encourages us to use the good parts of the language, with an emphasis on concise modules and proper testing strategies. Test-driven development is a powerful concept; it allows us to change the code and see problems immediately instead of being embarrassed later on, when users find problems on the page that we have overlooked.

There are other books that are really helpful in understanding these concepts more deeply, most notably *Mastering Web Application Development with AngularJS*, *Packt Publishing*. Nonetheless, we will have a look at these concepts in this book, as they will be beneficial throughout the development process.

jQuery has proven really useful though because so many UI problems can readily be solved by widgets that have been contributed by the community over the years. These widgets should not directly be used in AngularJS because they involve direct DOM manipulation, which is strongly discouraged in AngularJS.

This is where this book comes in: it aims to show you how to solve different kinds of UI-related problems within AngularJS development. In this book, you will learn:

- How to use the AngularUI companion suite to solve common UI problems
- How to adapt and extend the AngularUI library to solve your specific problem
- How to use UI Bootstrap, which allows you to quickly come up with an appealing user interface using Twitter Bootstrap
- How to adapt UI Bootstrap to fit your specific needs
- How CSS works
- How to build custom directives when the companion suite we cover does not solve your problem

What this book covers

Chapter 1, *Setting Up the Environment*, walks through how to set up a very basic but fully tested sample app that will be used as a starting template throughout the rest of the book. This chapter also covers automating the testing and building process.

Chapter 2, *AngularUI – Introduction and Utils*, introduces the AngularUI companion suite and shows how to use the Keypress and Event Binder, jQuery Passthrough, Validate and Mask, Highlight, and Fragment utilities.

Chapter 3, *AngularUI – Extended*, shows you how to use date, calendar, Google Maps, and also the UI-Router modules.

Chapter 4, *Customizing and Exploring ng-grid*, shows you how to build appealing grids, from basic examples to more sophisticated topics such as groupings, custom cell and row templates, paging, mastering seven detail views, and cell selections/editing.

Chapter 5, *Learning Animation*, will show you how to animate things in AngularJS. This includes fading page elements in and out and moving elements around, all based on changes in the AngularJS data model.

Chapter 6, *Using Charts and Data-driven Graphics*, is a more advanced chapter that will show you how to embed dynamic charts on a web page that reflect changes in the AngularJS data model. For this, a custom directive is built.

Chapter 7, *Customizing AngularJS with CSS and CSS Frameworks*, gives you a solid foundation in Cascading Style Sheets (CSS) and how CSS frameworks can simplify the UI development process. This chapter lays the foundation for *Chapter 8*, *AngularUI Bootstrap Development*, and particularly for *Chapter 9*, *Customizing AngularUI Bootstrap*.

Chapter 8, *AngularUI Bootstrap Development*, explains how to seamlessly integrate AngularJS and Twitter Bootstrap by utilizing the AngularUI Bootstrap project. The Twitter Bootstrap CSS framework allows building an appealing and adaptive user interface in much less time than what would be needed if we were to code all this from scratch.

Chapter 9, *Customizing AngularUI Bootstrap*, shows you how to adapt UI Bootstrap for specific needs. The default templates are appealing, but they might not always fit. In this chapter, we will explore how to adapt the directives using custom templates.

Chapter 10, *Mobile Development Using AngularJS and Bootstrap*, explores how to build a mobile single-page application, including touch gestures. It also covers how to optimize for particular requirements of mobile user experience.

What you need for this book

Trying out the examples in the book is highly recommended. If you want to do so, all you really need is your text editor or IDE of choice.

However, it is highly recommended to also install Node.js (`http://nodejs.org`), NPM with Grunt (`http://gruntjs.com`), and Karma runner (`http://karma-runner.github.io`). These will allow us to test our applications and to automate tedious tasks. You will find a detailed description on how to do this in *Chapter 1*, *Setting Up the Environment*.

Also, it is highly recommended to use a source control management system. Although this not a topic in this book, we will briefly introduce Git (`http://git-scm.com`) for this purpose when setting up the environment in *Chapter 1*, *Setting Up the Environment*. Trust me, if you haven't used it yet, you really should. It will save you from a headache (or two).

Other than that, you should bring some curiosity and an interest in building modern web applications. We will be covering some interesting and fun topics.

Who this book is for

This book is for anyone who is interested in solving UI problems with AngularJS. Working knowledge of JavaScript, HTML, and CSS is assumed. You should also have had some basic exposure to AngularJS; however, you might get along with solid skills particularly.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "The `$scope` object is an object that holds the data for the current environment, such as inside a controller or inside a directive."

A block of code is set as follows:

```
body {
    font-family: sans-serif;
    font-size: 1.2em;
    margin: 50px;
}
```

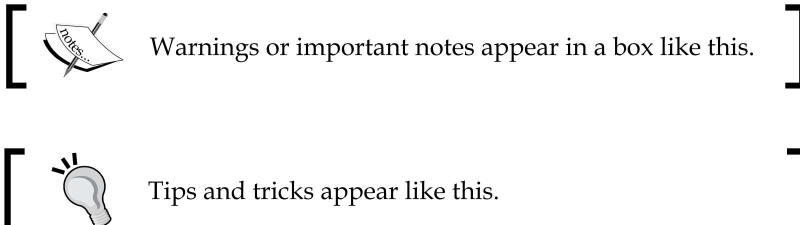
When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
'use strict';
angular.module('myApp', ['myApp.controllers', 'myApp.directives', 'ui.
utils'])
    .value('uiJqConfig', {
        tooltip: {
            placement: 'bottom'
        }
    });
```

Any command-line input or output is written as follows:

```
$ npm install
$ bower install
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Go ahead and select any issue to see its details in a **View Issue** tab."



Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you. The code samples for this title are also hosted on GitHub at `https://github.com/matthiasn/AngularUI-Code`. The GitHub code is maintained and updated, and is the author's preferred repository.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Setting Up the Environment

In this chapter, we will set up the environment to work with AngularJS. We will do a bit of this manually first to understand the concepts before using more automated approaches. By the end of the chapter, you will have a decent understanding of the AngularJS tool chain without having to rely on the magic of scaffolding tools, such as Yeoman (`http://yeoman.io`). There is nothing wrong whatsoever with using tools such as Yeoman later on, but there will be times when a deeper understanding serves you well.

We will follow the tradition of building a **Hello World** application. It would be rather boring to only have this application say *hello* to the world; instead, let's build an application in which we can enter a name to be greeted with. Let's see the two-way data binding in action by automatically updating another element on the page when the name changes.

Then, we will also wrap the markup for this example in a very basic directive that we will develop together. I have the impression that somehow writing custom directives is seen as a very advanced topic. Well, so be it; it will be all the more gratifying to get a basic introduction over with in this initial chapter.

Once this is done, we will build the scaffolding around this Hello World application by giving it a proper and meaningful structure that will allow us to use the result as a template for subsequent chapters. This scaffolding will already contain a test or two plus the means to build the application (bundling JavaScript in one file, for example).

You can download the source for this chapter; however, I cannot recommend it highly enough to actually go through the effort of typing these few lines of code. This always really helps me retain things, which I don't find all that surprising. After all, reading and then typing engages more parts of the brain than just reading something. However, ultimately it is, of course, up to you.

www.allitebooks.com

Even if you do choose to download the source code, please do yourself a favor and play around with it. Change things, break things, and fix them; all this *will* enable you to learn more in a shorter span of time.

In particular, in this chapter, we will:

- Build a very basic Hello World application
- Refine this application by using an object as the model and an additional input field
- Create our first directive
- Install Node.js and NPM (we'll need these for the subsequent tasks)
- Install and use Bower
- Write unit tests and run them using Karma
- Write integration tests using Protractor
- Create a build system with Grunt.js
- Manage client-side dependencies with Bower
- Learn how to use Git as a version control system

So, without further ado, let's get started.

Hello World

I assume you have used AngularJS before. If not, you may want to go through the tutorials at `http://angularjs.org` to get a better understanding of the framework, particularly the famed two-way data binding. You might get along well by just following the examples in this chapter too.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you. The code samples for this title are also hosted on GitHub at `https://github.com/matthiasn/AngularUI-Code`. The GitHub code is maintained and updated, and is the author's preferred repository.

Let's start with the markup; the simplest HTML5 plus some basic CSS for the styling will suffice for now. Create a directory anywhere you want, create an `src` subdirectory, and put a file named `index.html` with the following content in it. We will use the input field later to allow the user to input a name:

```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
    <link rel="stylesheet" href="css/main.css">
  </head>
  <body>
    <div>Hello World!</div>
    <input></input>
  </body>
</html>
```

Then, create a subdirectory named `css` with a file named `main.css` and with the following content:

```
body {
    font-family: sans-serif;
    font-size: 1.2em;
    margin: 50px;
}
```

Okay, this will do for the moment. Now, let's see how we can make this interactive by adding some AngularJS code. First, you need to download AngularJS, which you can find at `https://angularjs.org/`. Download the zip archive; we will need a few files from this. The latest version at the time of writing this book was v1.2.21.

> You can also use a **Content Delivery Network** (**CDN**) instead to serve the `angular.js` files. Google, for example, hosts the AngularJS files; check out this link:
>
> `https://developers.google.com/speed/libraries/devguide#angularjs`
>
> Using a CDN can be advantageous when the file is already in the cache of the user's browser, as it will then reduce the load time for the application.

Now, create a `js` subdirectory; create a `vendor` subdirectory inside the `js` directory, in which you place the `angular.js` file.

Then, create two empty files named `app.js` and `controllers.js` inside the `js` subdirectory. The directory structure of our application now looks as follows:

```
Step01-HelloWorld — bash — 76×14
mn:Step01-HelloWorld mn$ tree
.
└── src
    ├── css
    │   └── main.css
    ├── index.html
    └── js
        ├── app.js
        ├── controllers.js
        └── vendor
            └── angular.js

4 directories, 5 files
mn:Step01-HelloWorld mn$
```

At this point, it does not matter whether you use the minified version of AngularJS or not. In production, it is highly recommended that you use the minified version (of any library) though, as this will reduce the page load, particularly on slow mobile connections.

Then, add the following to `app.js`:

```
'use strict';
angular.module('myApp', ['myApp.controllers']);
```

The preceding code creates an AngularJS module named `myApp` and requires the `myApp.controllers` module, which we will create in the next step by adding the following code to `controllers.js`:

```
'use strict';
angular.module('myApp.controllers', []).
    controller('helloWorldCtrl', function ($scope) {
        $scope.name = "World";
    });
```

The preceding code creates the `myApp.controllers` module with the `helloWorldCtrl` controller. This controller holds the `$scope` object, which is the (admittedly simple) model we are creating for the application. The `name` variable on the `$scope` object will be used for the two-way data binding that AngularJS is famous for.

With the files in the right places, let's edit the `index.html` file so that AngularJS can teach our browser some two-way data binding tricks:

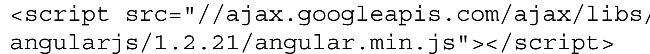
```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
    <link rel="stylesheet" href="css/main.css">
```

```
  </head>
  <body data-ng-app="myApp">
    <div data-ng-controller="helloWorldCtrl">
      <div>Hello {{ name }}!</div>
      <input data-ng-model="name"></input>
    </div>
    <script src="js/vendor/angular.js"></script>
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
  </body>
</html>
```

If you want to use a CDN, consider this line:

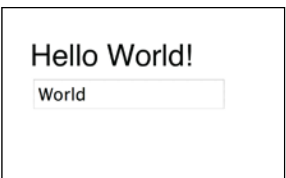
```
<script src="js/vendor/angular.js"></script>
```

Replace it with the following code:



```
<script src="//ajax.googleapis.com/ajax/libs/
angularjs/1.2.21/angular.min.js"></script>
```

This is it for the first version of our Hello World example. Open `index.html` in the browser and you can see two-way data binding in action, as follows:



You can change the name in the input field, and the text in the header will update automatically and immediately, as follows:

Using objects instead of primitives

Let's change one more thing that will save us from trouble later. Data models should reference objects and not primitives, such as a string, because the latter can create a headache when used in directives. Directives have a child `$scope` object that receives the model property by reference in the case of an object but as a copy when a primitive value is used, in which case, we might introduce difficult-to-spot errors.

You might ask what this `$scope` object is. The `$scope` object is an object that holds the data for the current environment, such as inside a controller or inside a directive. Now, you can pass a property of the controller's `$scope` object into the directive, which then becomes a property of the directive's own `$scope` object. For this, it is important to understand how JavaScript deals with parameter passing. When you pass an object to a function, the object is passed by reference, meaning that wherever this passed object is modified, these modifications **reach back the sender**. When passing objects to directives, this is exactly the behavior we want. For example, we have a directive to modify personal details, and we pass a `person` object from the controller to this directive. When we now edit the details, we expect the change to reach the controller. However, when we pass primitives such as `Strings` or `Numbers`, they are copied instead of being passed by reference. When we modify these in the directive, the changes will not be reflected in the controller's `$scope` object.

In `controllers.js`, we only need to change one line, introducing an object with properties for the first and last names:

```
'use strict';
angular.module('myApp.controllers', []).
    controller('helloWorldCtrl',function ($scope) {
        $scope.name = { first: "Jane", last: "Doe" };
    });
```

In `index.html`, let's create an additional field so that each can be edited separately:

```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
    <link rel="stylesheet" href="css/main.css">
  </head>
  <body data-ng-app="myApp">
    <div data-ng-controller="helloWorldCtrl">
      <h1>Hello {{ name.first }} {{ name.last }}!</h1>
      <input data-ng-model="name.first"></input>
      <input data-ng-model="name.last"></input>
    </div>
    <script src="js/vendor/angular.js"></script>
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
  </body>
</html>
```

Building our first directive

With these changes, we are well prepared to create a basic custom directive.

A directive is a way to create a custom element with its own behavior, which we can then use later as a building block in our application. One example of such a directive could be a login form. In this case, we will just create a directive for greeting us, with the div for both the greeting and input fields. Directives can be of arbitrary complexity, and we can use other directives inside the markup of a directive that we define.

For this, add a `directives.js` file to the `js` folder, with the following content:

```
'use strict';
angular.module('myApp.directives', [])
.directive('helloWorld', function () {
    return {
        restrict: 'AE',
        scope: { name: "=name" },
        template:
"<h1>Hello {{ name.first }} {{ name.last }}!</h1>" +
   "<input data-ng-model='name.first'></input>" +
  "<input data-ng-model='name.last'></input>"
        }
    });
```

AngularJS will now place the markup in the previous `template` property, inside the element that uses this directive. This might not seem terribly useful yet, but it will be when elements are repeated, for example.

Note that we can also load the markup from a template file, which is much more convenient than the previous multiline string, particularly as the directive gets more complex. However, for this simple example, this should serve us well.

The `index.html` file now becomes simpler than before:

```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
    <link rel="stylesheet" href="css/main.css">
  </head>
  <body data-ng-app="myApp">
```

```
    <div data-ng-controller="helloWorldCtrl">
      <div hello-world name="name"></div>
    </div>
    <script src="js/vendor/angular.js"></script>
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
    <script src="js/directives.js"></script>
  </body>
</html>
```

> Older versions of Chrome do not support the loading of the template when the `index.html` file is loaded from the filesystem; instead, Chrome complains of cross-origin requests. In order to quickly set up a web server, you can use `SimpleHTTPServer` from Python, which comes with OS X and Linux. All you need for this is to run the following command in the command line inside the application folder:
>
> ```
> python -m SimpleHTTPServer 8000
> ```
>
> Of course, you can use any other server to serve your folder, but this is really simple and often useful.

Nice! We just built our first directive. Much more can be done with them; directives are a really powerful concept for encapsulating UI elements. However, it's a start.

> Directives can apply to elements, classes, and attributes. In this case, we chose an attribute named `hello-world`, as follows:
>
> ```
> <div hello-world name="name">
> ```
>
> We could use an element, shown as follows; try it out by adding this line below the previous line:
>
>
>
> ```
> <hello-world name="name"></hello-world>
> ```
>
>
>
> All we need to change now is this line in `directives.js`:
>
> ```
> restrict: 'AE',
> ```
>
> Here, `A` means attribute, `E` stands for element, and `C` stands for class. Check out step 3a in the accompanying source code to see both versions side by side.
>
> However, using custom elements might not work in older browsers. *You have been warned!*

Installing Node.js and NPM

We will need to install a couple of additional tools to test and build our application. The foundation for this will be Node.js and NPM, the node package manager. Node.js is a platform built on top of V8, Google's JavaScript runtime, which is also used in Chrome. You can also build server-side, networked applications with Node.js, but here, it will only power our test and build the system. For now, think of it as a way to run the JavaScript code outside the browser. NPM is the package manager for Node.js, which makes it quite simple to install additional functionality. It comes bundled with Node.js.

The easiest way to install Node.js is to follow the instructions at `http://nodejs.org`.

OS X

On a Mac, you could either run the aforementioned installer or you could use Homebrew, a package manager for many open source libraries.

More information on Homebrew can be found at `http://brew.sh/`. Run the following command in the Terminal after installing Homebrew:

```
# brew install npm
```

The previous command will automatically install Node.js as a dependency of NPM if necessary.

Shell commands will run in a Terminal window. If you haven't used the Terminal before, go to the magnifying glass icon in the upper-right corner of your screen and type `Terminal`. You will need to open a new window after the installation is complete in order to run Node or NPM from a Terminal window.

Windows

On Windows, the easiest way to install Node.js is to run the Windows installer from `http://nodejs.org`. This installer, by default, now also installs NPM.

Command-line commands will run in the Windows command prompt. On Windows 7, this is available by clicking on the home button and entering `cmd` in the search box. You will need to open a new command-line window after the installation in order to be able to run Node or NPM from the command line.

Linux (Ubuntu)

On Ubuntu, there are two ways of installing Node and NPM, as follows:

1. Preferred: download the source package from the Node.js web page and build it yourself (this may take a few minutes with a lot of output). This way, you are guaranteed to have the latest version. For this, perform the following steps:
    1. Uncompress the package.
    2. Open the folder in the Terminal.
    3. Inside the directory, run the following commands:
       ```
       # ./configure
       # make
       # sudo make install
       ```
    4. You may have to install G++ first:
       ```
       # sudo apt-get install g++
       ```
2. You can also install NPM through Ubuntu's package manager by running the following command in the Terminal:
   ```
   # sudo apt-get install npm
   ```
   This will install Node.js as a dependency of NPM as well. Afterwards, if you want to upgrade NPM itself, then run the following command:
   ```
   # sudo npm install –g npm
   ```
   While this approach works, you may get an outdated version; so, the first approach is generally recommended.

Managing client-side dependencies with Bower

Earlier in this chapter, we downloaded AngularJS and placed the `angular.js` file in our directory structure manually. This was not so bad for a single file, but this process will very soon become tedious and error-prone when dealing with multiple dependencies.

Luckily, there is a great tool for managing client-side dependencies, and it can be found at `http://bower.io`.

Bower allows us to record which dependencies we need for an application. Then, after downloading the application for the first time, we can simply run `bower install` and it will download all the libraries and assets specified in the configuration file for us.

First, we will need to install Bower (potentially with `sudo`):

```
# npm install -g bower
```

Now, let's try it out by running the following command:

```
# bower install angular
```

You will notice an error message when running the previous command on a machine that does not have **Git** installed. In this case, please head to the final part of this chapter and follow the installation instructions.

Okay, this will download AngularJS and place the files in the `app/bower_components/` folder. However, our remaining sources are in the `src/` folder, so let's store the Bower files here as well. Create a file named `.bowerrc` in the project root, with these three lines in it:

```
{
  "directory": "src/bower"
}
```

This will tell Bower that we want the managed dependencies inside the `src/bower/` folder. Now, remove the `app/` folder and run the earlier `bower install` command one more time. You should see the AngularJS files in the `src/bower/` folder now.

Now, we said we wanted to record the dependencies in a configuration file so that we can later run `bower install` after downloading/checking out the application.

Why can't we just store all the dependencies in our version control system? Of course, we can, but this would bloat the repository a lot. Instead, it is better to focus on the artifacts that we created ourselves and pull in the dependencies when we check out the application for the first time.

We will create the configuration file now. We could do this by hand or let Bower do it for us. Let's do the latter and then examine the results:

```
# bower init
```


www.allitebooks.com

This will start a dialog that guides us through the initial Bower project setup process. Give the application a name, version number, and description as you desire. The main file stays empty for now. In the module type selection, just press *Enter* on the first option. Whenever you see something in square brackets, for example, `[MIT]`, this is the default value that will be used when you simply press *Enter*. When we confirm that the currently installed dependencies should be set as dependencies, AngularJS will automatically appear as a project dependency in the configuration file if you have followed the previous instructions. Finally, let's set the package as `private`. There is no need to have it accidentally appear in the Bower registry. Once we are done, the `bower.json` file should look roughly as follows:

```
{
  name: 'Hello World',
  version: '0.0.0',
  homepage: 'https://github.com/matthiasn/AngularUI-Code',
  authors: ['Matthias Nehlsen <mn@nehlsen-edv.de>'],
  description: 'Fancy starter application',
  license: 'MIT',
  private: true,
  ignore: [
    '**/.*',
    'node_modules',
    'bower_components',
    'src/bower',
    'test',
    'tests'
  ],
  dependencies: {
    angular: '~1.2.22'
  }
}
```

Now that we have a dependency management in place, we can use AngularJS from here and delete the manually downloaded version inside the `js/vendor/` folder. Also, edit `index.html` to use the file from the `bower/` directory. Find the following line of code:

```
<script src="js/vendor/angular.js"></script>
```

Replace it with this:

```
<script src="bower/angular/angular.js"></script>
```

Now, we have a package manager for client-side dependencies in place. We will use more of this in the next step.

Testing the Hello World application

We might be tempted to leave the application as it is. It works as expected; we can convince ourselves of this easily by changing the name and then observing if the rendering changes as expected. However, this is not a viable approach for more complex applications. In those scenarios, we can easily break something and then overlook the problem, only to be reminded by customers that something is not working as expected. This can always happen, even in an application with decent test coverage. However, we can dramatically decrease the chance of being called at night because something broke by implementing a proper test strategy, consisting of both unit and integration tests.

Unit tests

Let's start with unit tests for the controller and the directive we just built. Unit tests, as the name suggests, test the individual units of an application, as opposed to integration tests. We will take a look at those later.

We will need additional files and directories for the tests that we are about to create. Our test runner script and some of the directory structure are inspired by the angular-seed project (`https://github.com/angular/angular-seed`).

Create additional files and directories so that your project structure resembles the following. Note that the `src/bower` directory contains additional files and folders, but you can ignore this as these are managed by Bower:

```
Step05-Karma — bash — 76×25
mn:Step05-Karma mn$ tree
.
├── README.md
├── bower.json
├── conf
│   └── karma.conf.js
├── scripts
│   ├── test.bat
│   └── test.sh
├── src
│   ├── bower
│   ├── css
│   │   └── main.css
│   ├── index.html
│   └── js
│       ├── app.js
│       ├── controllers.js
│       └── directives.js
└── test
    └── unit
        ├── controllers.spec.js
        └── directives.spec.js

8 directories, 12 files
mn:Step05-Karma mn$
```

We will need `angular-mocks.js` to run the tests. Now that we have Bower in place, adding `angular-mocks` to our project becomes very simple; just run this single command:

```
# bower install --save angular-mocks
```

The preceding command will not only download the dependency, but thanks to the `--save` option, it will also add it to the `bower.json` file as a project dependency.

Installing Karma and Jasmine

Karma is a test runner provided by the AngularJS team. More on Karma can be found at `http://karma-runner.github.io/0.12/index.html`.

Karma allows us to run all the tests that we write against our codebase so that we can see at a glance when we break something. These tests can be run either once or on a continuous basis so that they always run again when a file changes.

The actual tests will be written with the help of **Jasmine**, a JavaScript testing library. More on Jasmine can be found at `http://jasmine.github.io`.

We will see tests written with Jasmine soon; however, we first need to install Karma and karma-jasmine plus the browser-specific test launchers. From inside your project directory, run the following commands:

```
# npm install karma
# npm install karma-jasmine
# npm install karma-chrome-launcher
# npm install karma-firefox-launcher
```

This will install Karma and karma–jasmine plus the Firefox and Chrome launchers locally, inside the `node_modules` subdirectory. The launchers are just examples; there are more for all the relevant browsers on the market. For the demonstration now, let's use these two.

We could run Karma from inside a nested subdirectory; however, it is not very convenient; it would be better to always have the command available anywhere. We can achieve this by installing `karma-cli` as follows:

```
# npm install –g karma-cli
```

Note that the `-g` option installs Node modules globally. The exact location of where these modules are installed depends on your platform and configuration. You might have to run the previous command with superuser privileges, as follows, depending on your local configuration:

```
# sudo npm install -g karma-cli
```

Only use `sudo` when the command fails.

With these tools in place, let's edit the `karma.conf.js` file so that we can run the unit tests in both Firefox and Chrome:

```
module.exports = function(config){
    config.set({
    basePath : '../',
    files : [
      'src/bower/angular/angular.js',
      'src/bower/angular-mocks/angular-mocks.js',
      'src/js/**/*.js',
      'test/unit/**/*.js'
    ],
    autoWatch: true,
    frameworks: ['jasmine'],
    browsers : ['Chrome', 'Firefox'],
    plugins : ['karma-chrome-launcher',
              'karma-firefox-launcher',
              'karma-jasmine']
})};
```

We specify which files to load and which framework and browsers to use when running the tests. Note that files and subdirectories are matched using asterisks. Then, we specify that `autoWatch` is `true`, meaning that Karma will watch for filesystem changes and rerun whenever a file has changed. Thus, we get a very tight feedback loop. When we break something, we will notice it immediately.

We also specify that we want to use Jasmine as the testing framework. Other frameworks can also be used here; however, Jasmine is a good choice and the one we will be using for this book. Just note that you could use others here. Finally, we specify the plugins, which have the same names as the NPM modules we loaded earlier. Here, you could, for example, delete the Firefox launcher if you only wanted to run the tests in Chrome or vice versa.

Now, we will edit the `test.sh` file:

```
#!/bin/bash
BASE_DIR=`dirname $0`
echo ""
echo "Starting Karma Server (http://karma-runner.github.io)"
echo "-----------------------------------------------------"
karma start $BASE_DIR/../conf/karma.conf.js $*
```

Now all that remains to make the test script runnable is the following command in the Terminal inside our main folder, `chmod +x scripts/test.sh` (this applies to both Mac OS X and Linux). On Windows, you create a `test.bat` batch file instead:

```
@echo off
set BASE_DIR=%~dp0
karma start "%BASE_DIR%\..\conf\karma.conf.js" %*
```

Let's now create the test for the controller by adding the following to the `test/unit/controllers.spec.js` file:

```
'use strict';
describe('controller specs', function() {
  var $scope;

  beforeEach(module('myApp.controllers'));

  beforeEach(inject(function($rootScope, $controller) {
    $scope = $rootScope.$new();
    $controller('helloWorldCtrl', {$scope: $scope});
  }));

  it('should create "name" model with first name "Jane"', function() {
    expect($scope.name.first).toBe("Jane");
  });
});
```

The test is contained in the describe block, in which we create a *fake* `$scope` object and then pass this `$scope` object to an instance of the `helloWorldCtrl` controller. As the it-block suggests, we expect the `$scope` object to now have a name model that contains an object with the first property that is equal to the string `Jane`. If this condition is `true`, the test passes.

Next, we want to create a simple test for the hello-world directive. Add the following to the `test/unit/directives.spec.js` file:

```
'use strict';
describe('specs for directives', function() {
  beforeEach(module('myApp.directives'));
  var $scope;
  beforeEach(inject(function($rootScope) {
    $scope = $rootScope.$new();
     $scope.name = {first: "John", last: "Doe"};
  }));
```

```
  describe('hello-world', function() {
    it('should contain the provided name', function() {
      inject(function($compile) {
        var element = $compile('<div hello-world name="name"></div>')
($scope);
         $scope.$digest();
        expect(element.html()).toContain("John");
      });
    });
  });
});
```

The previous test creates a `$scope` object, this time with a name model whose first property equals `John`. We compile an HTML snippet that contains the directive inside a `<div>` tag, passing the mock `$scope` object. After calling `$scope.$digest()`, we would expect the compiled element to contain the string `John`, and running the test as described later will show that this expectation is indeed warranted.

Now, you can run the test by typing the following in the Terminal inside our application folder. This is how the output looks on OS X:

```
$ scripts/test.sh
Starting Karma Server (http://karma-runner.github.io)
-------------------------------------------------------
INFO [karma]: Karma v0.10.2 server started at http://localhost:9876/
INFO [launcher]: Starting browser Chrome
INFO [launcher]: Starting browser Firefox
INFO [Firefox 24.0.0 (Mac OS X 10.8)]: Connected on socket […]
INFO [Chrome 30.0.1599 (Mac OS X 10.8.5)]: Connected on socket […]
Firefox 24.0.0 (Mac OS X 10.8): Executed 2 of 2 SUCCESS (0.142 secs /
0.025 secs)
Chrome 30.0.1599 (Mac OS X 10.8.5): Executed 2 of 2 SUCCESS (0.541 secs /
0.036 secs)
TOTAL: 4 SUCCESS
```

On Ubuntu, the output looks very similar. The following is the output when the test is configured to only run Firefox:

```
$ scripts/test.sh
Starting Karma Server (http://karma-runner.github.io)
-------------------------------------------------------
INFO [karma]: Karma v0.12.21 server started at http://localhost:9876/
INFO [launcher]: Starting browser Firefox
```

```
INFO [Firefox 31.0.0 (Ubuntu)]: Connected on socket […]
Firefox 31.0.0 (Ubuntu): Executed 2 of 2 SUCCESS (0.054 secs / 0.041
secs)
```

This concludes the unit testing for our simple example. In larger applications, the test scenarios will, of course, be much more complex and potentially complicated, but the principles remain the same.

Integration / end-to-end tests with Protractor

Unit testing is a great start towards feeling more secure that our application will work even after changing parts of the code base. However, unit testing does not cover the entirety of the application. Let's take our simple application that we have written so far as an example. What else can we test beyond looking at individual parts?

We could test interactions with the application and then observe if it reacted the way we expected. In this very simple example, this could be done by opening the application in the browser, changing the first and last names, and then seeing whether the displayed greeting changed accordingly.

So far, we don't need tools for this; we can simply do this by hand. However, it gets increasingly tedious as the application grows, especially when we want to test this interaction in multiple browsers.

Luckily, there is a tool for this called Protractor, which is also built by the AngularJS team. You can find out more about Protractor at `https://github.com/angular/protractor`.

Let's install and configure Protractor according to the tutorial on its GitHub page, and apply the tutorial to our own code. We start with the installation of the module, which will also install `webdriver-manager` as a dependency, and then update it:

```
# npm install -g protractor
# webdriver-manager update
```

Once again, you might have to run the previous commands with superuser (sudo) privileges. Once this is complete, we can start the server with the following command:

```
# webdriver-manager start
```

Note that you will need a Java runtime on your system in order to run webdriver. You can obtain this from `http://java.com/en/download/index.jsp`.

On a Mac, you don't have to worry about this. If you don't have a **JVM** (**Java Virtual Machine**) installed yet, the system will notice this and ask you whether it should download it for you.

On Ubuntu, you can also run the following command instead of using the previous installer:

```
# sudo apt-get install default-jre
```

This will start a server application that controls different browsers. We can now write test scenarios that will call this server application, which will in turn start browsers as specified, run the tests, and allow us to observe if the results are as desired.

We will write such a test soon, but first, we need to get another prerequisite installed: a simple local web server that serves our project over HTTP: http-server.

This is an NPM module that gives us a little tool that we can then call on the command line and that will then serve the content of the working directory over HTTP, on a specified port. We install it as follows (potentially using sudo):

```
# npm install -g http-server
```

Let's try it out right away by starting the following from inside our project directory:

```
# http-server -a localhost -p 8000
```

The previous command will start the server on port `8000` and only listen on the `localhost` interface. You could leave out the `–a localhost` option, but this might irritate your firewall. Once this is running, you can open the following URL in the browser:

```
http://localhost:8000/src/
```

If you have started http-server from the root of the project, you should now see our application loaded and working.

Now, let's write a simple test only to check the title of the page, just to check that the tool chain is working. For this, create a file named `protractor.conf.js` inside the `conf` folder with the following content:

```
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['../test/protractor/spec.js'],
  capabilities: {
      browserName: 'firefox'
  }
}
```

The previous configuration reads as follows:

- The address of our selenium instance (the webdriver) is the same that the server displays when we started it about a page ago, `http://localhost:4444/wd/hub`.
- We will run a single test file for now, `spec.js`, in the `test/protractor` directory of the project root. We will create this one next.
- We will run the test in only one browser for now; let's use Firefox this time. Here, you could, for example, use Chrome instead or select one of the many other options. We could also use multiple browsers for each and every test. The Protractor documentation has all the required information.

Now, create `spec.js` inside the `test/protractor` folder as follows:

```
 describe(hello world app', function() {
  it('should have a title', function() {
    browser.get('http://localhost:8000/src/');
    expect(browser.getTitle()).toEqual('Hello World');
  });
});
```

The previous test will open a browser, Firefox in the case of the current configuration; open the page at `http://localhost:8000/src/` and check if the title of the page is **Hello World**, as expected, let's run it:

```
# protractor conf/protractor.conf.js
```

We should see the following result:

```
Using the selenium server at http://localhost:4444/wd/hub
Finished in 0.315 seconds
1 test, 1 assertion, 0 failures
```

Now, change the title of the page or the expected string in the test, run the test again, and you will see that the output changes:

```
Using the selenium server at http://localhost:4444/wd/hub
Failures:
  1) angularjs homepage should have a title
   Message:
     Expected 'Hello World' to equal 'Hello Worlds'.
   Stacktrace: [...]
Finished in 0.307 seconds
1 test, 1 assertion, 1 failure
```

Great, we can not only see that something went wrong but also exactly which expectation failed.

Now, the previous test alone is not all that useful; let's add some interaction by adding the following to the end of `spec.js`:

```
describe('name fields', function() {
    it('should be filled out and editable', function() {
        browser.get('http://localhost:8000/src/');

        var h1 = element.all(by.css('h1')).first();
        var fname = element.all(by.tagName('input')).first();
        var lname = element.all(by.tagName('input')).get(1);
        expect(h1.getText()).toEqual("Hello Jane Doe!");
        expect(fname.getAttribute('value')).toEqual("Jane");
        expect(lname.getAttribute('value')).toEqual("Doe");

        fname.clear().sendKeys('John');
        lname.clear().sendKeys('Smith');

        expect(h1.getText()).toEqual("Hello John Smith!");
        expect(fname.getAttribute('value')).toEqual("John");
        expect(lname.getAttribute('value')).toEqual("Smith");
  });
});
```


www.allitebooks.com

Let's go through the previous additional test:

- We open a browser window.
- We create variables for the elements because we will use each variable multiple times. Note that there are multiple ways to select elements, for example, here we are using both `by.css` and `by.tagName`.
- We expect the elements in their initial state to have the correct values.
- We clear the input elements and enter text through the `sendKeys` method.
- We expect the elements to have the correct values after the model has changed through our text input.

The previous section has really only scratched the surface of what we can do with the very powerful **Protractor**. Please refer to the useful and detailed documentation and tutorial to learn more. It will be time well spent.

Building the application

Preparing an application for production environments can be a tedious task. For example, it is highly recommended that you create a single file that contains all the JavaScript needed by the application instead of multiple small files, as this can dramatically reduce page load time, particularly on slow mobile connections with high latency. It is not feasible to do this by hand though, and it is definitely no fun.

This is where a build system really shines. We are going to use **Grunt** for this. A single command in the Terminal will run the tests and then put the necessary files (with a single, larger JavaScript file) for the application in the `dist/` folder. Many more tasks can be automated with Grunt, such as minifying the JavaScript, automating tasks by watching folders, running **JsHint** (`http://www.jshint.com`), but we can only cover a fairly basic setup here.

Let's get started. We need to install Grunt first (possibly with `sudo`):

```
# npm install -g grunt-cli
```

Then, we create a file named `package.json` in the root of our application folder with the following content:

```
{
  "name": "my-hello-world",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-concat": "~0.5.0",
```

```
    "grunt-contrib-copy": "~0.5.0",
    "grunt-targethtml": "~0.2.6",
    "grunt-karma": "~0.8.3",
    "karma-jasmine": "~0.1.5",
    "karma-firefox-launcher": "~0.1.3",
    "karma-chrome-launcher": "~0.1.4"
  }
}
```

This file defines the NPM modules that our application depends on. These will be installed automatically by running `npm install` inside the root of our application folder.

Next, we define which tasks Grunt should automate by creating `Gruntfile.js`, also in the root of our application folder, with the following content:

```
module.exports = function(grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    concat: {
      options: {
        separator: ';'
      },
      dist: {
        src: ['src/js/vendor/*.js','src/js/*.js'],
        dest: 'dist/js/<%= pkg.name %>.js'
      }
    },
    copy: {
      main: {
        src: 'src/css/main.css',
        dest: 'dist/css/main.css',
      },
    },
    targethtml: {
      dist: {
        files: {
          'dist/index.html': 'src/index.html'
        }
      }
    },
    karma: {
      unit: {
        configFile: 'conf/karma.conf.js',
        singleRun: true
```

```
      }
    }
  });
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-targethtml');
  grunt.loadNpmTasks('grunt-karma');
  grunt.registerTask('dist', ['karma', 'concat', 'targethtml',
'copy']);
};
```

This file contains different sections that are of interest. In the first part, the configuration object is created, which defines the options for the individual modules. For example, the `concat` section defines that a semicolon should be used as a separator when it merges all JavaScript files into a single file inside the `dist/js` folder, with the name of the application. It is important that `angular.js` comes before the application code inside this file, which is guaranteed by the order inside the `src` array. Files in the `vendor` subfolder are processed first with this order.

The copy task configuration is straightforward; here, we only copy a single CSS file into the `dist/css` folder. We will do more interesting things with CSS later when talking about CSS frameworks.

The `targethtml` task processes the HTML so that it only loads the one concatenated JavaScript file. For this to work, we need to modify `index.html`, as follows:

```
<!DOCTYPE html>
  <head>
    <meta charset="utf-8">
    <title>Angular UI Template</title>
    <link rel="stylesheet" href="css/main.css">
  </head>
  <body data-ng-app="myApp">
    <div data-ng-controller="helloWorldCtrl">
      <div hello-world name="name"></div>
    </div>
    <!--(if target dev)><!-->
    <script src="js/vendor/angular.js"></script>
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
    <script src="js/directives.js"></script>
    <!--<!(endif)-->
    <!--(if target dist)><!-->
    <script src="js/my-hello-world.js"></script>
    <!--<!(endif)-->
  </body>
</html>
```

This, together with the configuration, tells the `targethtml` task to only leave the section for the `dist` task inside the HTML file, effectively removing the section that will load individual files.

[You might be tempted to think that it will not make much of a difference if one or multiple files need to be retrieved when the page is loaded. After all, the simple concatenation step does not reduce the overall size of what needs to be loaded.

However, particularly on mobile networks, it makes a huge difference because of the latency of the network. When it takes 300 ms to get a single response, it soon becomes noticeable whether one or ten files need to be loaded. This is still true even when you get the maximum speed in 3G networks. LTE significantly reduces latency, so the difference is not quite as noticeable. The improvements with LTE only occur in ideal conditions, so it is best not to count on them.]

The `karma` section does nothing more than tell the `karma` task where to find the previously defined configuration file and that we want a single test run for now. Next, we tell Grunt to load the modules for which we have created the configuration, and then we define the `dist` task, consisting of all the previously described tasks.

All that remains to be done is to run `grunt dist` in the command line when we want to test and build the application. The complete AngularJS web application can then be found in the `dist/` folder.

Running Protractor from Grunt

Let's also run Protractor from our `grunt` task. First, we need to install it as follows:

```
# npm install grunt-protractor-runner --save-dev
```

This will not only install the `grunt-protractor-runner` module, but also add it as a dependency to `package.json` so that when you, for example, check out your application from your version control system (covered next) on a new computer and you want to install all your project's dependencies, you can simply run:

```
# npm install
```

[If you follow along using the companion source code instead of typing the source code yourself, you will need to run `npm install` again in the last step's folder.]

Next, edit `Gruntfile.js` so that from the `karma` section, it looks as follows:

```
    karma: {
      unit: {
        configFile: 'conf/karma.conf.js',
        singleRun: true
      }
    },
    protractor: {
      e2e: {
        options: {
          configFile: 'conf/protractor.conf.js'
        }
      }
    }
  });
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-targethtml');
  grunt.loadNpmTasks('grunt-karma');
  grunt.loadNpmTasks('grunt-protractor-runner');
  grunt.registerTask('dist', ['karma', 'protractor', 'concat',
'targethtml', 'copy']);
};
```

Now, Protractor will also run every single time we call `grunt dist` to build our application. The build process will be stopped when either the `karma` or the `protractor` step reports an error, keeping us from ever finding code in the `dist` folder when fails tests.

Note that we will need to have both the `webdriver-manager` and `http-server` modules running in separate windows for the `grunt dist` task to work. As a little refresher, these were started as follows:

```
# webdriver-manager start
# http-server -a localhost -p 8000
```

Both can also be managed by Grunt, but that makes the configuration more complex and it would also mean that the task runs longer because of startup times. These can also be part of a complex configuration that watches folders and runs and spawns all the required tasks automatically. Explore the Grunt documentation further to tailor an environment specific to your exact needs.

We will expand on our basic usage of Grunt and do more sophisticated tasks later, for example, to create a single, minified CSS file using a CSS framework or to apply minification to the concatenated JavaScript file when further optimizing for mobile web applications in *Chapter 10, Mobile Development Using AngularJS and Bootstrap*. Now that you know how the build system works in general, you may already want to explore more advanced features. The project's website is a good place to start (`http://gruntjs.com`).

Managing the source code with Git

You are probably using Git already. If not, you really should. You need to have it installed for working with Bower anyway, so why not use it for your own projects?. Git is a distributed **Version Control System** (VCS). I cannot imagine working without it, neither in a team nor when working on a project myself. We don't have the space to cover Git for team development here in either case; this topic can easily fill an entire book. However, if you are working in a team that uses Git, you will probably know how to use it already. What we can do in this introductory chapter is go through the basics for a single developer.

First, you need to install Git if you do not have it on your system yet.

OS X

On a Mac, again the easiest way to do this is using Homebrew (`http://brew.sh`). Run the following command in the Terminal after installing Homebrew:

```
# brew install git
```

Windows

On Windows, the easiest way to install Git is to run the Windows installer from `http://git-scm.com/downloads`.

Linux (Ubuntu)

On Ubuntu, run the following command in the shell:

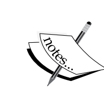
```
# sudo apt-get install git
```

Let's initialize a fresh repository in the current directory:

```
git init
```

Then, we create a hidden file named `.gitignore`, for now with only the following content:

```
node_modules
```



This tells Git to ignore the hundreds of files and directories in the `node_modules` folder. These don't need to be stored in our version control system because the modules can be restored by running `npm install` in the root folder of the application instead, as all the dependencies are defined in the `package.json` file.

Next, we add all files in the current directory (and in all subdirectories):

```
git add
```

Next, we freeze the file system in its current state:

```
git commit -m "initial commit"
```

Now, we can edit any file, try things out, and accidentally break things without having to worry because we can always come back to anything that was committed into the VCS. This adds incredible peace of mind when you are playing around with the code.

When you issue the `git status` command, you will notice that we are on a branch called master. A project can have multiple branches at the same time; these are really useful when working on a new feature. We should always keep master as the latest stable version; additional features (or bug fixes) should always be worked upon in a separate branch. Let's create a new feature branch called `additional-feature`:

```
git branch additional-feature
```

Once it is created, we need to check out the branch:

```
git checkout additional-feature
```

Now, when the new code is ready to be committed, the process is the same as before:

```
git add .
git commit -m "additional feature added"
```

We should commit early and often; this habit will make it much easier to undo previous changes when things go wrong. Now, when everything is working in the new branch (all the tests pass), we can go back into the master branch:

```
git checkout master
```

Then, we can merge the changes back into the master branch:

```
git merge additional-feature
```

Being able to freely change between branches, for example, makes it very easy to go back to the master branch from whatever you are working on and do a quick bug fix (in a specialized branch, ideally) without having to think about what you just broke with the current changes in the new feature branch. Please don't forget to commit before you switch the branch though. You can merge the bug fix in this example back into the master, go back to the feature branch you were working on, and even pull those changes that were just done in the master branch into the branch you are working on. For this, when you are inside the feature branch, merge the changes:

```
git merge master
```

If these changes were in different areas of your files, this should run without any difficulties. If they were in the same lines, you will need to manually resolve the conflicts.

Using Git is a really useful way to manage source code. However, it is in no way limited to code files (or text files, for that matter). Any file can be placed under source control. For example, this book was written with heavy usage of Git, for any file involved. This is extremely useful when you are trying to go back to the previous versions of any file.

Summary

We started this chapter by covering some basics about AngularJS. Then, we built a very simple Hello World app with two-way data binding. We then upgraded the simple application to use an object for the property of the model instead of using string primitives.

Afterwards, we built our first custom directive and moved parts of the markup of our application into this custom directive.

Then, we covered testing. We learned why testing is essential for more complex applications. For testing, we first installed Karma with Jasmine to run the first unit tests. We also created our first integration tests, which are useful for testing how the parts of the application play together.

We then installed and configured Grunt.js as a build system for our application. A properly configured build system frees us from having to run tedious and potentially error-prone tasks over and over again.

Finally, we briefly took a look at the usage of Git, an open source distributed version control system, which is optional for the rest of the book but highly encouraged.

In the next chapter, we will begin with **AngularUI**, a suite of tools and building blocks, for making application development with AngularJS simpler and more productive.

2
AngularUI – Introduction and Utils

In the previous chapter, we created a build system for an AngularJS application, including unit and integration / end-to-end tests. This environment should serve us well even when building much more complicated applications in larger teams. Testing becomes really essential then, but even in small projects, they add peace of mind.

This chapter will introduce the AngularUI companion suite and show some elements of the AngularUI-Utils collection in action. We will be using the final step of the previous chapter as a template for the examples in this chapter because it provides a well-structured application.

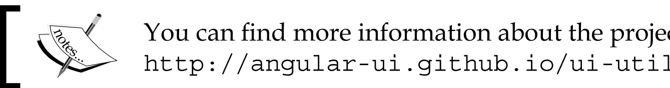
We will cover the following topics in this chapter:

- Downloading and building the suite with **Grunt**
- Testing the suite with **Karma**
- Integrating the suite into our project
- Using the **Event Binder** component
- Using the **Keypress** component
- Using the **Mask** component
- Using the **jQuery Passthrough** component

www.allitebooks.com

Downloading AngularUI

AngularUI is a companion suite for AngularJS that is used to solve common problems related to UI development. The common problems include handling Keypress events, passing through jQuery components, validating and masking inputs, converting string representations, including fragments, formatting strings, and handling checkboxes.

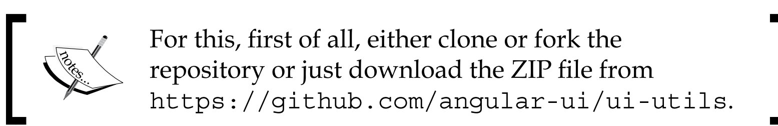


You can find more information about the project at `http://angular-ui.github.io/ui-utils/`.

On the project's website, you can also find a download link for the current version of the JavaScript files. However, since we just discussed building AngularJS applications using Grunt, let's build this project from the project's source on **GitHub** instead.

Building AngularUI-Utils

With what we learned about package managing in *Chapter 1*, *Setting Up the Environment*, let's build AngularUI-Utils ourselves before we start using it. We could just download the JavaScript file(s), but it will be more gratifying to do this ourselves. Learning how to use Grunt will also be very helpful in any larger project later on.



For this, first of all, either clone or fork the repository or just download the ZIP file from `https://github.com/angular-ui/ui-utils`.

For simplicity, I suggest that you download the ZIP file; you can find the link on the right-hand side of the GitHub project page. Once we have unpacked the ZIP file, we first need to install the dependencies. On your command line inside the project folder, run the following commands:

```
$ npm install
$ bower install
```

This will install the necessary files needed for the build process. Let's first check whether all the tests are passing after the previous commands have run through:

```
$ karma start --browsers=Chrome test/karma.conf.js --single-run=true
```

Alternatively, you can also simply run:

```
$ grunt
```

The tests are part of the default task specified in `gruntFile.js`. Take a moment and familiarize yourself with the file by trying to find where the default task is specified. Note that one subtask is the `karma:unit` task. Try to locate this task further down in the file; it specifies which Karma configuration file to load.

If all the tests pass, as they should, we can then build the suite using the following command:

```
$ grunt build
```

This will run the following tasks specified in `gruntFile.js`:

- The `concat:tmp` task concatenates the modules into a temporary JavaScript file, except `modules/utils.js`. Take a look at the configuration for this task; the easiest way is to search for `concat` within `gruntFile.js`.
- The `concat:modules` task concatenates the resulting temporary file from step one into the final JavaScript library file, which you can then find in the `bower_components/angular-ui-docs/build` directory. The configuration for the `concat:modules` task should be right below the previous one. Here, the difference is that there is no absolute file and path; instead, the name is resolved so that common parts, such as the repository name and such, are not repeated within the configuration file. This follows the DRY (don't repeat yourself) principle and makes the configuration file easier to maintain.
- The `clean:rm_tmp` task removes the temporary file, created previously.
- The `uglify` task finally creates a minified version of the JavaScript file for use in production because of the smaller file size.

I highly recommend that you spend some time reading and following through the `gruntFile.js` file and the tasks specified therein. It is not strictly necessary that you follow along in this chapter, as simply building the suite (or even downloading it from the project's website) would suffice, but knowing more about the Grunt build system will always be helpful for our own projects.

Imagine this: you find an issue in some project and add it to the list of known issues on the GitHub project. Someone picks it up immediately and fixes it. Now, do you want to wait for someone (maybe an automated build system) to decide when it is a good time to publish a version that includes the said fix? I wouldn't; I'd much rather be able to build it myself. You never know when the next release will happen.

Integrating AngularUI-Utils into our project

In this step, we will take the `ui-utils.js` file we built in the previous section and use it in a sample project. The UI-Utils suite consists of a large and growing number of individual components. We will not be able to cover all of them here, but the ones we do cover should give you a good idea about what is available. For this, first make a copy of the code we came up with in the previous chapter. Alternatively, you can use the code in `Chapter01/Step08-Grunt-Protractor` as a template.

Now, let's do the following edits:

1. Copy the `ui-utils.js` file from the `ui-utils/bower_components/angular-ui-docs/build/` folder to the `src/js/vendor/` folder of the current project.
2. Open the `package.json` file and change the name of the application as you wish, for example:

```
{
  "name": "fun-with-ui-utils",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-concat": "~0.5.0",
    "grunt-contrib-copy": "~0.5.0",
    "grunt-karma": "~0.8.3",
    "grunt-protractor-runner": "^1.1.0",
    "grunt-targethtml": "~0.2.6",
    "karma": "~0.12.0",
    "karma-chrome-launcher": "~0.1.4",
    "karma-firefox-launcher": "~0.1.3",
    "karma-jasmine": "~0.1.5"
  }
```

3. Edit `src/index.html` by inserting a script tag right below the script tag for the `angular.js` file, and change the name of our concatenated project JavaScript file, as the name change above will result in a different filename. The body of the `index.html` file now looks as follows:

```
<body ng-app="myApp">
  <div ng-controller="helloWorldCtrl">
    <h1 hello-world name="name" id="greeting"></h1>
  </div>
  <!--(if target dev)><!-->
  <script src="bower/angular/angular.js"></script>
  <script src="js/vendor/ui-utils.js"></script>
```

```
  <script src="js/app.js"></script>
  <script src="js/controllers.js"></script>
  <script src="js/directives.js"></script>
  <!--<!(endif)-->
  <!--(if target dist)><!-->
    <script src="js/fun-with-ui-utils.js"></script>
  <!--<!(endif)-->
</body>
```

4. Run `grunt dist` to see whether our tests are still passing and whether the project is built without any problems.

Dist folder

You will notice that the `my-hello-world.js` file is still in the `dist/js/` folder, despite not being used any longer. You can safely remove it. You could also remove the entire `dist` folder and run the `my-hello-world.js` file again:

```
$ grunt dist
```

This will recreate the folder with only the required files. Deleting the folder before recreating it could become a part of the `dist` task by adding a clean task that runs first. Check out `gruntFile.js` of the UI-Utils project if you want to know how this is done.

Grunt task concat

Note that all JavaScript files get concatenated into one file during the `concat` task that runs in our project during the `grunt dist` task. These files need to be in the correct order, as the browser will read the file from beginning to end and it *will* complain when, for example, something references the AngularJS namespace without that being loaded already. So, while it might be tempting to use wildcards as we did so far for simplicity, we shall name individual files in the correct order. This might seem tedious at first, but once you are in the habit of doing it for each file the moment you create or add it, it will only take a few seconds for each file and will keep you from scratching your head later. Let's fix this right away.

Find the source property of the `concat` task in `Gruntfile.js` of our project:

```
src: ['src/js/vendor/*.js','src/js/*.js'],
```

Now, replace it with the following:

```
src: ['src/bower/angular/angular.js',
      'src/js/vendor/ui-utils.js',
      'src/js/app.js',
      'src/js/controllers.js',
      'src/js/directives.js'],
```

We also need to edit the app module so that it loads UI-Utils. To do this, edit `app.js` as follows:

```
'use strict';
angular.module('myApp', ['myApp.controllers', 'myApp.directives', 'ui.
utils'])
```

With these changes in place, we are in a pretty good shape to try out the different components in the UI-Utils suite. We will use a single project for all the different components; this will save us time by not having to set up separate projects for every single one of them.

uiMask directive

Let's add a phone number field for our person model in the application we have so far. Obviously, we want valid phone numbers in our data model so let's make sure the format is valid by using the mask component.

This component will allow us to define a mask that specifies how a correct entry looks, for example, (555) 555-5555; ext 55.

First, let's make some changes so that the page renders equally well on desktop and mobile browsers. We will be using Twitter Bootstrap in later chapters anyway, so why not get started with it right now! Download Bootstrap from `http://getbootstrap.com` and place `bootstrap.css` inside `src/css/vendor/`.

We will need to change `Gruntfile.js` as follows in order to include `bootstrap.css` in the `dist` folder:

```
copy: {
    main: {
        files: [ { expand: true, cwd: 'src/css/',
                   src: ['**'], dest: 'dist/css/' } ]
       }
},
```

This specifies that from the `src/css/` current working directory, all files including subfolders will be copied to `dist/css/`. Now, edit the head tag of `index.html` to look as follows:

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
        user-scalable=no">
    <title>Angular UI Template</title>
    <link rel="stylesheet" href="css/vendor/bootstrap.css">
    <link rel="stylesheet" href="css/main.css">
</head>
```

We will be using the viewport `meta` tag because we want this page to render well on a mobile browser. Then, we also need to include the `bootstrap.css` file. Note that we are not specifying anything different for the `dist` target here. In a production environment, we will surely do so by concatenating and minifying the CSS, but for now, this is fine. We will get to these optimizations in detail later.

Now, let's change `controllers.js`, as follows:

```
'use strict';
angular.module('myApp.controllers', [])
.controller('helloWorldCtrl', function ($scope) {
    $scope.person = {
        firstName: "Jane",
        lastName: "Doe"
    };
    $scope.mask = "(999) 999-9999 ext 99";
    $scope.getModel = function () {
        return JSON.stringify($scope.person, undefined, 2);
    };
});
```

In the previous code, first we added a mask to our model, which we will use for the entry of the phone number. This works as follows:

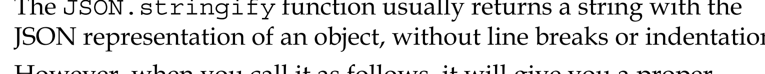
- For any allowed number character, you use a `9`
- For any allowed letter character, you use an `A`
- For numbers or letters, you use `*`
- Anything else is shadowed

Have a look at the following examples:

- For a typical US phone number: `(999) 999-9999`
- For a typical US phone number with an extension: `(999) 999-9999 ext 99`
- For a spreadsheet cell location: `Row: 99 Col: AA`

We will use a `<pre><code>` block with a pretty-printed JSON representation of the data model for demonstration purposes. To obtain this representation from the view, we create a `$scope.getModel` function, which returns the prettified JSON string.

Pretty-printed JSON



The `JSON.stringify` function usually returns a string with the JSON representation of an object, without line breaks or indentations.

However, when you call it as follows, it will give you a proper pretty-printed representation, with two-character indentation and line breaks:

```
JSON.stringify(object, undefined, 2)
```

This representation is much easier for humans to decipher than the regular `JSON.stringify` output.

With these changes in place, let's get to the actual markup. We will create a responsive layout that looks fine on a mobile device as well. We are creating a page with fields for first name, last name, masked phone number, and a representation of the data model. Change the content of the body tag before the scripts, as follows:

```
<div class="container" ng-controller="helloWorldCtrl">
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right"
            for="fname">First Name</label>
        <input class="col-xs-7" type="text" id="fname"
            ng-model="person.firstName">
    </div>
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right"
            for="lname">Last Name</label>
        <input class="col-xs-7" type="text" id="lname"
            ng-model="person.lastName">
    </div>
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right"
            for="phone">Phone</label>
        <input class="col-xs-7" type="tel" id="phone"
            ui-mask="{{ mask }}" ng-model="person.phone">
    </div>
    <div class="row spacer-lg">
        <pre class="col-xs-12"><code>{{ getModel() }}</code></pre>
    </div>
</div>
```

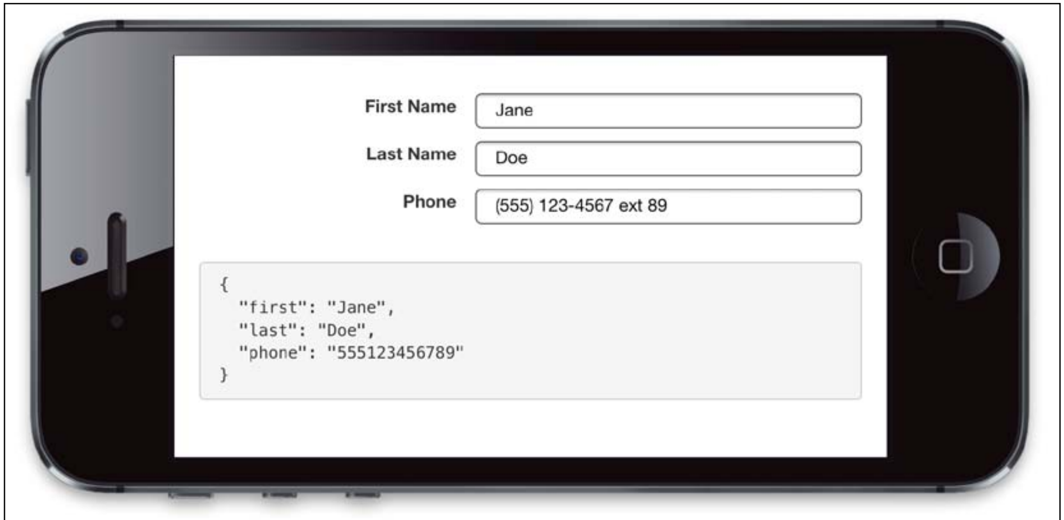
Note that for now, we are not using the `helloWorld` directive but instead integrating these fields directly into the markup, which is a little easier at the moment.

Let's fine-tune the CSS a little bit and edit `main.css` as follows:

```
body {
    margin: 20px;
}

.spacer-sm {
    margin-top: 10px;
}

.spacer-lg {
    margin-top: 30px;
}
```

We now have a page that renders well on desktop and mobile devices, shown as follows:

In the previous image, you see the initial rendering with no phone number entered yet. Note that the JSON representation of the data model only contains two properties, **first** and **last**. In the following image, you can see how the data model changes once a valid phone number has been entered:



Event Binder

In this section, we are going to take a look at event binding. The Event Binder component allows binding to arbitrary events, whereas the KeyPress component is more specific. As the name suggests, it is helpful when we want to know when the user has pressed a key. It is also possible to achieve the same without the AngularUI suite; then, the code required will just be longer.

Let's start with the Event Binder. We will change the color of the background of our data model JSON representation depending on whether the first name field is in focus or not. When it is in focus, we will paint the background light red to reflect that the data model is currently being changed. Once blurred (blur is the opposite of the `focus` event; it happens when the focus leaves the element) we paint the background in light green for a short moment to reflect that the data model editing is over and then go back to gray. We will make use of the `ng-style` attribute here to make this work.

ng-style

With ng-style, we can bind the styling of an element (partially or completely) to the data model. In the data model, a JavaScript object is expected to hold pairs of CSS property and a value that should change based on the changes in the model.

For example, if we want to bind the background color of an element to the data model, the respective model object needs to have a `backgroundColor` property with whatever color you want it to have, as follows:

```
{ backgroundColor: "orange" }
```

The associated element will then repaint itself when this object changes. Note that the properties are Camel Case here, whereas they would be hyphenated inside the style sheet.

We will change the controller first. We need an array with the possible colors that the element can have, a variable holding the current index position within this array, a function that creates the style object as described in the previous information box, and the callback functions for the `focus` and `blur` events:

```
'use strict';
angular.module('myApp.controllers', []).controller('helloWorldCtrl',
function ($scope, $timeout) {
    $scope.person = {
        firstName: "Jane",
        lastName: "Doe"
    };
    $scope.mask = "(999) 999-9999 ext 99";
    $scope.getModel = function () {
        return JSON.stringify($scope.person, undefined, 2);
    };
    var colors = ["#CCC", "#F77", "#9F9"];
    var activeColor = 0;
    $scope.modelStatus = function() {
        return { backgroundColor: colors[activeColor] };
    };
    $scope.focusCallback = function() {
        activeColor = 1;
    };
    $scope.blurCallback = function() {
        activeColor = 2;
        $timeout(function() { activeColor = 0; }, 2000);
    };
});
```


www.allitebooks.com

Note the usage of `$timeout` in the `$scope.blurCallback` function. The `$timeout` function is injected into the controller together with `$scope`, and it takes two arguments: a function and the time in milliseconds after which this function should be run. In this case, we set the color back to the initial gray after 2 seconds.

With these changes in the controller, we can now edit `index.html`:

```
<div class="container" ng-controller="helloWorldCtrl">
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right"
            for="fname">First Name</label>
        <input class="col-xs-7" type="text" id="fname"
            ng-model="person.firstName"
            ui-event="{ focus: 'focusCallback()',
                        blur : 'blurCallback()' }" >
    </div>
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right"
            for="lname">Last Name</label>
        <input class="col-xs-7" type="text" id="lname"
            ng-model="person.lastName">
    </div>
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right" for="phone">Phone
        </label>
        <input class="col-xs-7" type="tel" id="phone"
            ui-mask="{{ mask }}" ng-model="person.phone">
    </div>
    <div class="row spacer-lg">
        <pre ng-style="modelStatus()" class="col-xs-12">
            <code>{{ getModel() }}</code>
        </pre>
    </div>
</div>
```

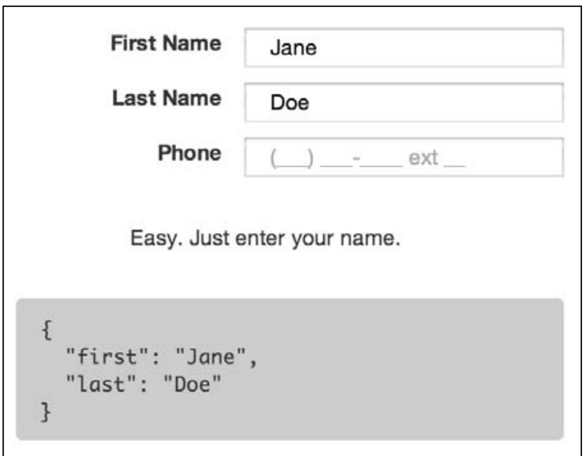
Here is how the result looks:

When the first name input field is in focus, the data model representation is shown with a red background. Once we click on **Done**, it is shown in green for 2 seconds and then goes back to the initial gray. On a desktop browser, clicking anywhere else would have the same effect as clicking on the **Done** button.

If you want, you can assign the same behavior to the other two input fields by copying the `ui-event` attribute from the `fname` input field.

Keypress

Next, we will implement a simple `help` function with the Keypress component. On pressing *F1* inside the **Last Name** field, we want to show help text below the field, shown as follows:



The div that holds the text shall be invisible except for when the *F1* key was pressed and will last only for 10 seconds. We will bind UI-keydown to the input field for the last name, and then we will use `ng-show` to hide the row with the help text unless the model contains help text. Here is the new `index.html` section with the changes highlighted:

```
<div class="container" ng-controller="helloWorldCtrl">
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right"
            for="fname">First Name</label>
        <input class="col-xs-7" type="text" id="fname"
            ng-model="person.firstName"
             ui-event="{ focus: 'focusCallback()',
                         blur : 'blurCallback()' }" >
    </div>
    <div class="row spacer-sm">
        <label class="col-xs-5 text-right" for="lname">Last
            Name</label>
        <input class="col-xs-7" type="text" id="lname"
            ng-model="person.lastName"
            ui-keydown="{112:'helpKeyDown($event)'}">
    </div>
    <div class="row spacer-sm">
```

```
            <label class="col-xs-5 text-right"
                for="phone">Phone</label>
            <input class="col-xs-7" type="tel" id="phone"
                ui-mask="{{ mask }}" ng-model="person.phone">
        </div>
        <div class="row spacer-lg" ng-show="helpText">
            <div class="col-xs-2"></div>
            <div class="col-xs-8">{{ helpText }}</div>
        </div>
        <div class="row spacer-lg">
            <pre ng-style="modelStatus()" class="col-xs-12">
                <code>{{ getModel() }}</code>
            </pre>
        </div>
    </div>
```

The changes necessary in `controller.js` are minimal this time; all we need to add is this function:

```
$scope.helpKeyDown = function($event) {
    console.log($event);
    $scope.helpText = "Easy. Just enter your name.";
    $timeout(function() { $scope.helpText = "" }, 10000);
};
```

Again, we are using `$timeout`, this time to make the help section disappear after 10 seconds.

ng-show

The `ng-show` attribute determines whether an element is visible or not, depending on the evaluation of the specified model. Specifically, the model is evaluated for truthiness.

In our case, we need to know that the empty string in JavaScript evaluates as `false`, which is great because we can use the model itself to determine whether the section is supposed to be visible or not.

If you want to get some practice, you can play around with this feature some more by making it callable from multiple fields. You could, of course, use multiple functions for this, one for each field. It would be straightforward. It is probably not ideal though, as it would lead to repeated code with the messages inside the code.

Or, you could use use a single function that gets called from multiple event-emitting fields. Then, you could load the messages from a centralized place, with a data structure that holds the messages for each element's ID. The ID of the caller element is available in `srcElement` or in the target property of the `$event` object, shown as follows:

```
▼ KeyboardEvent {altGraphKey: false, metaKey: false, altKey: false, shiftKey: false, ctrlKey: false…}
    altGraphKey: false
    altKey: false
    bubbles: true
    cancelBubble: false
    cancelable: true
    charCode: 0
    clipboardData: undefined
    ctrlKey: false
    currentTarget: null
    defaultPrevented: false
    detail: 0
    eventPhase: 0
    keyCode: 112
    keyIdentifier: "F1"
    keyLocation: 0
    layerX: 0
    layerY: 0
    location: 0
    metaKey: false
    pageX: 0
    pageY: 0
    returnValue: true
    shiftKey: false
  ▶ srcElement: input#lname.col-xs-7 ng-pristine ng-valid
  ▶ target: input#lname.col-xs-7 ng-pristine ng-valid
    timeStamp: 1383246924681
    type: "keydown"
  ▶ view: Window
    which: 112
  ▶ __proto__: KeyboardEvent
```

jQuery Passthrough

Usually, it is discouraged in AngularJS to do direct DOM manipulation; any jQuery calls should only happen from inside directives, nowhere else.

The jQuery Passthrough component allows us to use jQuery functions directly without having to wrap them in a directive by providing the binding to our data model.

First of all, you will need to download or install jQuery. Since we already installed Bower for package management, we might as well use it to retrieve the dependency:

```
$ bower install --save jquery
```

Then, as before, you need to add it to the `concat` task inside `Gruntfile.js`. jQuery needs to be in the first position; it needs to be loaded before AngularJS.

jQuery and AngularJS

AngularJS uses jQuery, if available; otherwise, it uses jQLite, which comes with AngularJS. jQLite provides some of the jQuery functionalities, but really only the basics. jQuery must be loaded before AngularJS so that it is used instead of jQLite.

The `concat` task then looks as follows:

```
concat: {
  options: {
    separator: ';'
  },
  dist: {
    src: [ 'src/bower/jquery/dist/jquery.js',
           'src/bower/angular/angular.js',
           'src/js/vendor/ui-utils.js',
           'src/js/vendor/bootstrap.js',
           'src/js/app.js',
           'src/js/controllers.js',
           'src/js/directives.js' ],
    dest: 'dist/js/<%= pkg.name %>.js'
  }
},
```

Next, you need to place the jQuery script tag inside `index.html` before `angular.js`:

```
<script src="bower/jquery/dist/jquery.js"></script>
<script src="bower/angular/angular.js"></script>
```

With these prerequisites in place, we can now use the jQuery Passthrough component for a data-driven tooltip. The idea is to provide a tooltip when hovering over the JSON representation of the data model. The `hover` event will also be triggered on mobile devices when tapping on the element in question. The text of the tooltip will then either ask whether the person's data model has no phone number property, or it will tell us that all is good.

This is how it is going to look:

We need a function in our controller that provides the tooltip text depending on the state of the model. Let's call this function `tooltip()`. We will add it towards the end of the controller declaration in `controllers.js`:

```
$scope.tooltip = function() {
    if(!$scope.person.hasOwnProperty("phone")) {
        return $scope.person.firstName + " has no phone?"
    }
    else { return "All good."}
}
```

Next, we will edit `index.html`:

```
<div ui-jq="tooltip" original-title="{{ tooltip() }}"
    class="row spacer-lg">
    <pre ng-style="modelStatus()" class="col-xs-12">
        <code>{{ getModel() }}</code>
    </pre>
</div>
```

We are assigning the div an attribute that holds the JSON representation of the model and a `ui-jq` attribute with the `tooltip` value. This tells `ui-jq` to pass through to `$.fn.tooltip()`.

Next, we add the `original-title="{{ tooltip() }}"` attribute. This means that the title of the tooltip (the text that we see when hovering over the element) is dynamically generated by the `tooltip()` function depending on the state of the data model.

There is just one more thing we need to do in order to place the tooltip at the bottom of the element it is assigned to: it is to provide the configuration object inside the app's module. To do this, edit `app.js`:

```
'use strict';
angular.module('myApp', ['myApp.controllers', 'myApp.directives', 'ui.
utils'])
    .value('uiJqConfig', {
        tooltip: {
            placement: 'bottom'
        }
    });
```

Summary

This concludes our tour of the `ui-utils` toolbox. There are more components in there, some of which are still under development. Going through this chapter should put you in a good position to explore these additional components as well. Let's recap what we have done in this chapter.

At the beginning of the chapter, we built the AngularUI-Utils library from the GitHub source. Then, we ran the automated tests for the suite. Normally, we would not have to do this, but rather use Bower for package management. However, it can be useful at times to build a project from the source. We then embedded AngularUI-Utils in our sample project.

Next, we started using Twitter Bootstrap CSS and created a responsive layout for desktop and mobile clients, into which we integrated the Mask component for adding a phone number field with a defined format.

Further, we made use of the Event Binder component to detect when an element is in focus and when it loses focus again. Then, we used the Keypress component to create a simple help function when *F1* is pressed inside an input field.

Finally, we used the tooltip functionality from Bootstrap's JavaScript file, and we bound it to dynamic data from our model using the jQuery Passthrough component.

In the next chapter, we will see additional AngularUI components, such as the Google Maps component or the calendar component. We will also use and test filters.

3
AngularUI – Extended

In this chapter, we will continue our tour of the AngularUI companion suite and put additional components to use. We will use the final step of the application we developed in *Chapter 2*, *AngularUI – Introduction and Utils*, as a template. Please refer to the chapter in order to learn how to set up AngularUI and embed it into our project. Alternatively, you can start with the companion code for this chapter. Just beware that we will not cover what we already went through in the previous chapter.

We will cover the following topics in this chapter:

- Using the Google Maps component
- Using the Calendar component
- Using and testing filters

There are more components in the AngularUI suite that we will not cover here, such as a few code editors, for example, Ace, CodeMirror, and TinyMCE. You can find more information on these at `http://angular-ui.github.io`.

Embedding Google Maps

The Google Maps component, as the name suggests, allows us to easily embed Google Maps into our website, with the location as well as markers and such being bound to our data model.

For the installation of this module, we will use Bower. With the `.bowerrc` file in place, as described in the previous section, run the following command inside the root directory of the project:

```
# bower install angular-ui-map
```

www.allitebooks.com

The previous command will install the maps component inside the `./src/bower_components` folder, together with its dependencies. Note that both AngularJS and UI-Utils were installed by the previous command because they are listed as dependencies in the `bower.json` file of the `angular-ui-map` package.

Now, we will link these packages in our `index.html` file, together with the Google Maps script, which we will load directly from Google.

We will use the example from `https://github.com/angular-ui/ui-map` as a template for our efforts. Note that we will have to manually bootstrap our application because the Google Maps `angular-ui` module can only be instantiated after the Google Maps script has been loaded completely. For this, we will get rid of the automatic bootstrapping of the application and trigger this process manually in a **callback** function once the Google Maps script has loaded successfully. This is mostly a Google Maps API issue, but still we have to work around it in order to use the `maps` directive here.

Bootstrapping AngularJS

Normally, an AngularJS application is bootstrapped automatically. What this means is that the application is loaded automatically by declaring the `ng-app` object, as follows:

```
<body ng-app="myApp">
```

However, when we want control over when this process should take place, we can start it manually by calling `angular.bootstrap` with the ID of the element to which we want to bind an AngularJS application.

Let's start by creating the aforementioned callback function inside `app.js`. This file will now look as follows:

```
'use strict';
function onGoogleReady() {
    angular.bootstrap(document.getElementById("myApp"), ['myApp']);
}
angular.module('myApp', ['myApp.controllers', 'myApp.directives', 'ui.utils']);
```

The previous code will defer the application's bootstrapping process until Google Maps has fully loaded. For this, the `onGoogleReady` function will be passed to the Google Maps script in the `index.html` file. Note that this function lives in the global scope of the browser, which is something that is best avoided but cannot be helped in this particular case.

We will add the markup from the `ui-map` example to our `index.html` file, which will then look as follows:

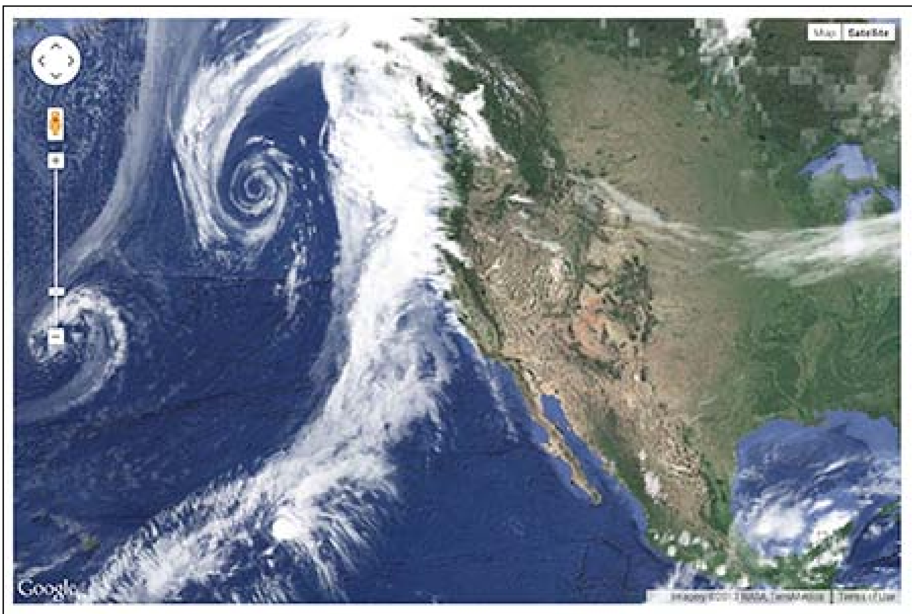
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular UI Maps</title>
    <link rel="stylesheet" href="css/main.css">
</head>
<body id="myApp">
<section id="map" ng-controller="MapCtrl">
    <div ui-map="myMap" ui-options="mapOptions"
        class="map-canvas">
    </div>
</section>
<!--(if target dev)><!-->
<script src="js/vendor/jquery-1.10.2.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script type="text/javascript" src="https://maps.googleapis.com/maps/
api/js?v=3.exp&sensor=false&callback=onGoogleReady"></script>
<script src="bower_components/angular-ui-map/src/map.js"></script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
<script src="js/directives.js"></script>
<!--<!(endif)-->
<!--(if target dist)>
<script src="js/fun-with-ui-modules.js"></script>
<!(endif)-->
</body>
</html>
```

Note that we are also loading `angular.js` and the `events` module from `angular-ui-utils`, present in the `bower_components` folder. These were installed as dependencies when installing the `maps` module, so we might as well use them. We can then delete the respective files from the `js/vendor` folder.

Next, we will modify `controller.js` to look as follows:

```
'use strict';
angular.module('myApp.controllers', []).controller('MapCtrl', function
($scope, $timeout) {
    $scope.myMarkers = [];

    $scope.mapOptions = {
        center: new google.maps.LatLng(37.782,-122.418),
        zoom: 4,
        mapTypeId: google.maps.MapTypeId.SATELLITE
    };

    var cloudLayer = new google.maps.weather.CloudLayer();

    $timeout(function(){
        cloudLayer.setMap($scope.myMap);
    }, 1000);
});
```

The preceding controller creates the model for our map, such as `mapOptions`, `eventBindings`, and `myMarkers`. We also create a cloud overlay that reflects the current cloud layer above the surface of our planet. Note the usage of `$timeout` to defer the attachment of the `cloudLayer` object to the map. This gives the application time to render the map first. The result now looks as follows:



Markers on the map

So far so good, now let's add some functionality. How about setting markers for when you click on the map, listing these markers below the map, and then adding a button to pan the map to the particular marker?

For this, we will need the event binding for the `click` event. Also, we need clickable `div` elements for each marker. Add the following code to the body of our `index.html` file:

```
<body id="myApp">
<section id="map" ng-controller="MapCtrl">
    <div ui-map="myMap" ui-options="mapOptions"
        class="map-canvas" ui-event="eventBinding"></div>
</section>
    <div class="marker" ng-repeat="marker in myMarkers"
         ng-click="myMap.panTo(marker.getPosition())">
         Lat: {{ marker.getPosition().lat() }} <br />
         Lng: {{ marker.getPosition().lng() }}</div>
<!--(if target dev)><!-->
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-ui-utils/modules/event/event.js
"></script>
<script src="bower_components/angular-ui-map/src/map.js"></script>
<script src="https://maps.googleapis.com/maps/api/js?libraries=places,
weather&sensor=false&callback=onGoogleReady"></script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
<script src="js/directives.js"></script>
<!--<!(endif)-->
<!--(if target dist)>
<script src="js/fun-with-ui-modules.js"></script>
<!(endif)-->
</body>
```

Note that in the scripts' section, we are also loading the `events` module, which we need to bind the events.

Event Binding

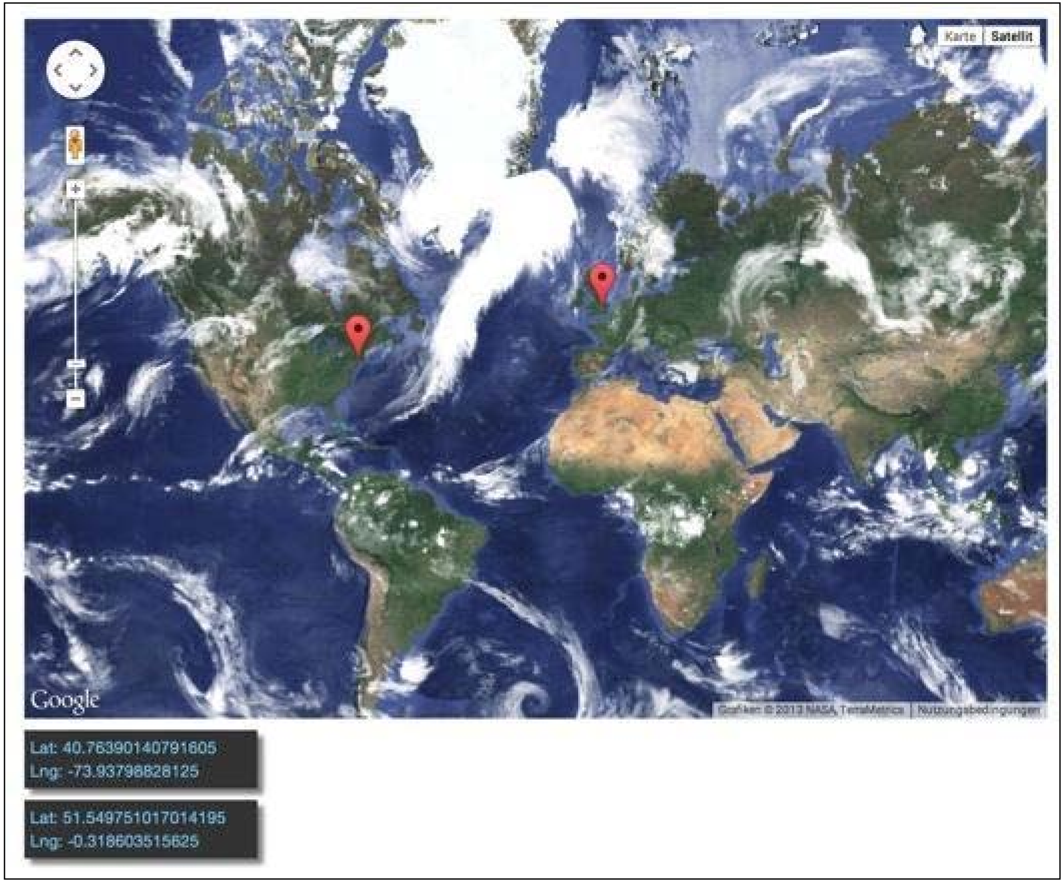
Next, we need to modify `controllers.js` to include the `addMarker` function and the `event-binding` object, as follows:

```
'use strict';
angular.module('myApp.controllers', []).controller('MapCtrl', function
($scope, $timeout) {
    $scope.myMarkers = [];

    $scope.mapOptions = {
        center: new google.maps.LatLng(37.782,-122.418),
        zoom: 4,
        mapTypeId: google.maps.MapTypeId.SATELLITE
    };

    var cloudLayer = new google.maps.weather.CloudLayer();
    $timeout(function(){
        cloudLayer.setMap($scope.myMap);
    }, 1000);

    $scope.addMarker = function($event, $params) {
        $scope.myMarkers.push(new google.maps.Marker({
            map: $scope.myMap,
            position: $params[0].latLng
        }));
    };
    $scope.eventBinding = {'map-click':
       'addMarker($event, $params)'};
});
```

Finally, let's add a little bit of style to this by appending the following to `main.css`:

```
.marker {
    width: 200px;
    color: lightskyblue;
    background-color: #333;
    margin: 10px 0;
    padding: 5px;
    box-shadow: 5px 5px 5px #888;
}
```

Now, the final example shows a map, with a cloud layer, that allows us to add markers by clicking on it and has button-like `div` elements that pan the map to a marker position when clicked:



Managing application dependencies with Bower

In the previous section, we installed the `ui-map` component manually, which then installed additional dependencies. Potentially, we would have to go through a whole list of components that are required when setting up the project on, say, a different machine. Would it not be nice if we could define the application dependencies and then only call a single command?

It turns out that is what the `bower.json` file is for; here we can define all the application dependencies in a single place. Then, when cloning the repository, for example, all you need to do is run the following command once:

```
# bower install
```

We can manually set up the `bower.json` file that is required for this, but there is a simpler alternative: `bower init`, which takes care of setting up the file for us, with the inclusion of the already installed components. Let's walk through the command-line process:

```
# bower init
[?] name: Step02-Maps
[?] version: 0.1.0
[?] description: AngularUI Map Example
[?] main file: src/index.html
[?] keywords: angular, maps
[?] authors: John Doe <john@doe.com>
[?] license: MIT
[?] homepage: http://some-url.com
[?] set currently installed components as dependencies? Yes
[?] add commonly ignored files to ignore list? Yes
[?] would you like to mark this package as private which prevents it from
being accidentally published to the registry? Yes
```

The `bower init` command will then set up the `bower.json` file for us, including the currently installed dependencies, if desired. This will be the result of running the preceding command:

```
{
  "name": "Step02-Maps",
  "version": "0.1.0",
  "homepage": "http://some-url.com",
  "authors": [
    "Your Name Here"
  ],
  "description": "AngularUI Map Example",
  "main": "src/index.html",
  "keywords": [
    "angular",
    "maps"
  ],
  "license": "MIT",
```

```
  "private": true,
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "src/bower_components",
    "test",
    "tests"
  ],
  "dependencies": {
    "angular-ui-map": "~0.5.0"
  }
}
```

Modifying the .gitignore file

Next, you should consider excluding the `bower_components` directory from your `git` repository because all the dependencies are reconstructed by running the `bower install` command. For this, edit `.gitignore`, as follows:

```
node_modules/
dist/
src/bower_components
```

The calendar component

In this section, we will try out the UI `calendar` component, which helps to embed the **Arshaw FullCalendar** into an AngularJS project. You can find more information about this project at `http://arshaw.com/fullcalendar`.

You can duplicate and modify the code from the maps section. This time, we will use Bower to resolve all our dependencies. First, let's do some cleaning up and delete the `src/bower_components` and `src/js/vendor` folders.

Next, we will add calendar-related dependencies to our calendar example. In addition, we will also use `moment.js`, a great little library for anything date-related in JavaScript. Modify `bower.json` as follows:

```
{
  "name": "Step03-Calendar",
  "version": "0.1.0",
  "homepage": "http://some-url.com",
  "authors": [ "John Doe <john@doe.com>" ],
  "description": "AngularUI Calendar Example",
```

```
  "main": "src/index.html",
  "keywords": [
    "angular",
    "calendar"
  ],
  "license": "MIT",
  "private": true,
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "src/bower_components",
    "test",
    "tests"
  ],
  "dependencies": {
     "angular-ui-calendar": "~0.8.1",
     "momentjs": "~2.8.3"
  }
}
```

Now, we can install all our dependencies by running a single command:

```
# bower install
```

Running the previous command will install the AngularUI `calendar` module with its dependencies, such as AngularJS, jQuery, and jQuery-UI, plus the separately specified `moment.js`. We need to add these to `index.html` and `Gruntfile.js` so that they become available in the distribution version of our application. We will also use a `moment.js` based filter in this application, which will live in `filters.js`, so we will create entries for this file as well.

Let's start with `index.html`:

```
<!--(if target dev)><!-->
<script src="bower_components/jquery/jquery.js"></script>
<script src="bower_components/jquery-ui/ui/jquery-ui.js"></script>
   <script src="bower_components/momentjs/moment.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-ui-calendar/src/calendar.js"></
script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
<script src="js/directives.js"></script>
<script src="js/filters.js"></script>
<!--<!(endif)-->
```

Next, we will add the same files to `Gruntfile.js` for the `concat` task:

```
concat: {
  options: { separator: ';' },
    dist: {
   src: [ 'src/bower_components/jquery/jquery.js',
     'src/bower_components/jquery-ui/ui/jquery-ui.js',
     'src/bower_components/fullcalendar/fullcalendar.js',
  'src/bower_components/momentjs/moment.js',
  'src/bower_components/angular/angular.js',
  'src/bower_components/angular-ui-calendar/src/calendar.js',
  'src/js/app.js','src/js/controllers.js','src/js/filters.js'
],
 dest: 'dist/js/<%= pkg.name %>.js'
  }
},
```

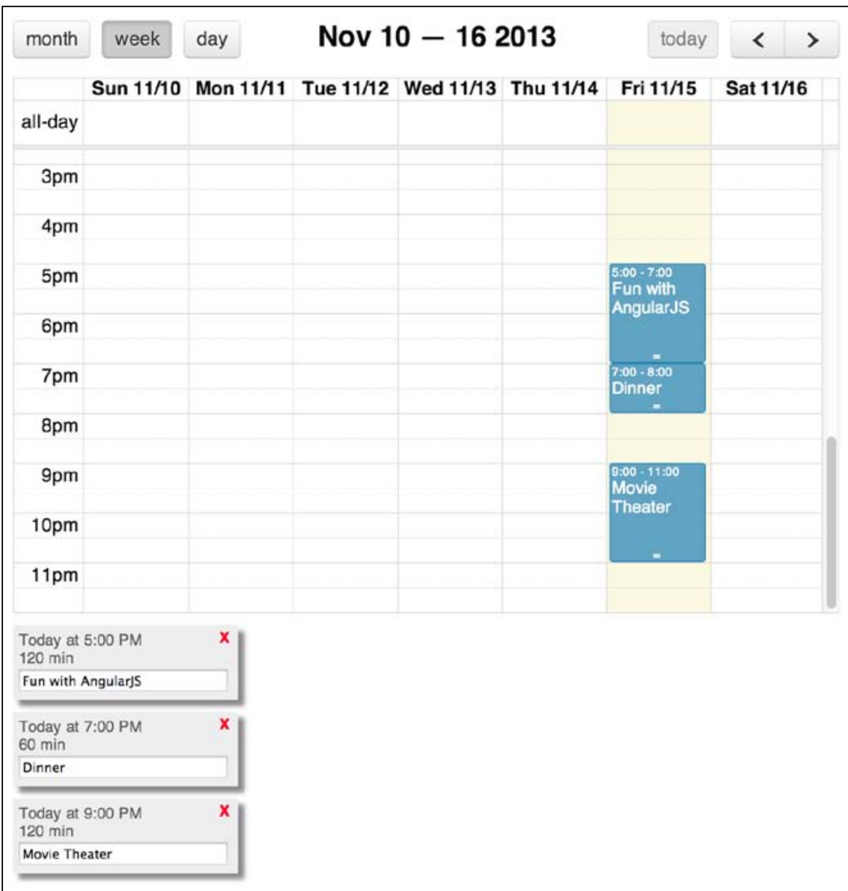
Using a filter for date formatting

We will use a filter to transform raw dates to strings that are easy for humans to understand. Inside this filter, we will be using the aforementioned `moment.js` library. Let's create a file named `filters.js` inside the `src/js/` directory with the following content:

```
'use strict';
angular.module("myApp.filters", [])
    .filter('moment', function () {
        return function (input) {
            return moment(input).calendar();
        };
    })
    .filter("duration", function () {
        return function (event) {
            if (!event.end) { return "All Day"; }
            var start = moment(event.start);
            var end = moment(event.end);
            return end.diff(start, "minutes") + " min";
        };
    });
```

In the previous code block, we defined two filters, one named `moment` and the other named `duration`. In the first filter, we allow `moment.js` to handle the conversation between the date and a human-friendly string, such as **Today** or **Tomorrow**. In the second filter, we test whether the event has an end, and if it does not, we conclude that it is an `all-day` event. Otherwise, we use the `diff` function from the `moment` objects to get the duration of an event in minutes. The `moment.js` file offers *a lot* of functionalities around date formatting; check out `http://momentjs.com` to learn more.

Now that we have the filter for proper time-string formatting, we can work on `index.html`. We are going to set up the calendar itself plus `divs` for each item in our calendar that has input fields in which we can alter the event text. This is how it will look:



In the `index.html` file, we also need to link the `fullcalendar.css` file, both for the development environment and for the distribution version. We will edit the `grunt` task afterwards to cater to this. Let's start with the head of `index.html`:

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular UI Calendar</title>
    <link rel="stylesheet" href="css/main.css">
    <!--(if target dev)><!-->
    <link rel="stylesheet" href="bower_components/fullcalendar/
fullcalendar.css">
    <!--<!(endif)-->
    <!--(if target dist)>
    <link rel="stylesheet" href="css/fullcalendar.css">
    <!(endif)-->
</head>
```

Note that the `targethtml` task in Grunt will remove the link to `fullcalendar.css` inside the `bower_components` folder and instead place a link to `css/fullcalendar.css` inside the `target html` file.

Let's edit the `copy` task in `Gruntfile.js` now because the link will otherwise be invalid for the `dist` version of our application. Find the `copy` task and edit it as follows:

```
copy: {
  main: {
    files: [
      { expand: true, cwd: 'src/css/', src: ['**'],
        dest: 'dist/css/' },
     { expand: true, cwd:'src/bower_components/fullcalendar/',
       src: ['*.css'], dest: 'dist/css/' } ]
  }
},
```

Next, let's focus on the body of `index.html`. We will add the calendar plus the editable entries as follows:

```
<body ng-app="myApp">
<div ng-controller="CalCtrl">
    <div ui-calendar="uiConfig.calendar" ng-model="eventSources"></
div>
    <div class="cal-entry" ng-repeat="event in events">
        <div class="delete" ng-click="remove()">X</div>
```

```
            <div>{{ event.start | moment }}</div>
            <div>{{ event | duration }}</div>
            <input ng-model="event.title" />
        </div>
    </div>

    <!--(if target dev)><!-->
    <script src="bower_components/jquery/jquery.js"></script>
    <script src="bower_components/jquery-ui/ui/jquery-ui.js"></script>
    <script src="bower_components/fullcalendar/fullcalendar.js"></script>
    <script src="bower_components/momentjs/moment.js"></script>
    <script src="bower_components/angular/angular.js"></script>
    <script src="bower_components/angular-ui-calendar/src/calendar.js"></
    script>
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
    <script src="js/filters.js"></script>
    <!--<!(endif)-->
    <!--(if target dist)>
    <script src="js/fun-with-ui-modules.js"></script>
    <!(endif)-->
    </body>
```

The usage of the previous `ui-calendar` directive is fairly straightforward. We will create the `uiConfig.calendar` object that needs to be in the `$scope` object, next in the controller. Note the usage of `ng-repeat` for the calendar entries. Inside these `divs`, we have the start time and duration, both formatted with the appropriate filters, an input field that lets us modify the event text, and a clickable **X** that allows the user to delete the entry by calling `$scope.remove()` when clicked.

Styling the calendar

Let's quickly add some styles before focusing on the controller of our application. For this, we will edit `main.css` as follows:

```
body {
    margin: 20px;
    font-family: sans-serif;
}

.cal-entry {
    width: 180px;
    color: #555;
    font-size: 0.8em;
```

```
        background-color: #EEE;
        margin: 10px 0;
        padding: 5px;
        box-shadow: 5px 5px 5px #888;
        position: relative;
    }
    .cal-entry input {
        width: 170px;
    }
    .cal-entry .delete {
        color: red;
        position: absolute;
        font-weight: bolder;
        right: 7px;
        top: 3px;
    }
```

Adapting the controller

Now let's create the controller to make all this work. Rewrite `controller.js` as follows:

```
'use strict';
angular.module('myApp.controllers', []).controller('CalCtrl', function
($scope) {
    $scope.events = [
        { title: 'All Day Event',
          start: moment().add('days', 3).format('L') },
        { title: 'Fun with AngularJS',
          start: moment().startOf('hour').format(),
          end: moment().endOf('hour').add('hour', 1).format(),
          allDay: false }
    ];
    $scope.eventSources = [$scope.events];

    $scope.dayClick = function(date){
        $scope.$apply(function() {
            $scope.events.push(
                { title: "new event",
                  start: date,
                  end: moment(date).add('hours', 1).format(),
                  allDay: false });
        });
    };
```

```
        $scope.remove = function(index) {
            $scope.events.splice(index,1);
        };

        $scope.uiConfig = {
            calendar:{
                height: 400,
                editable: true,
                header:{
                    left: 'month agendaWeek agendaDay',
                    center: 'title',
                    right: 'today prev,next'
                },
                defaultView: 'agendaWeek',
                dayClick: $scope.dayClick,
                eventDrop: $scope.$apply,
                eventResize: $scope.$apply
            }
        };
    });
```

Let's walk through this section by section:

1. First, we create an array with two sample events. The first one is an `all-day` event that takes place three days after the application startup, whenever that is. The second event, `Fun with AngularJS`, takes place right now, from the beginning of the current hour until the end of the following hour. Note that the duration will show as 119 minutes; you can fix this by adding a second to the end time, as follows:

   ```
   end: moment().endOf('hour').add('second', 1).add('hour',
   1).format()
   ```

2. The `$scope.eventSources` array holds the event sources, with the `events` array in it. Here, we can specify additional event sources, but for the purpose of our example, this will suffice.
3. The `$scope.dayClick` function is called when you click on the calendar and creates a new event at the click position.
4. The `$scope.remove` function is called when we click on the red **X** in the upper-right corner of an event. It removes the associated event from the `events` array.

5. Finally, `$scope.uiConfig` holds the calendar configuration, including the callback functions for particular events. One of these is the aforementioned `$scope.dayClick` function. Note that for the `eventDrop` and `eventResize` events, we are simply passing the `$scope.$apply` function. Moving and resizing events in the calendar modifies an `event` object, but this happens outside AngularJS, so we need to call `$scope.$apply()` for AngularJS to notice the change in the data model. However, there is no reason for us to not allow the `$scope.$apply` calendar call; we can simply pass this function along. For more information about the available configuration options, check out the project's website at `http://arshaw.com/fullcalendar/docs`.

Adding tests

Now, let's finally test the code we have created.

Of course, if you followed proper **test-driven development** (**TDD**), you would want to write these tests first. However, this is not a book on TDD, so we will only focus on the mechanics of writing tests, not on the workflow. Having said that, there is a lot of value in TDD, and you should definitely try it at some point and form your own opinion on the subject.

In order to test the application, we first have to adapt `karma.conf.js` to include all the necessary files. Edit it as follows:

```
module.exports = function (config) {
    config.set({
        basePath: '../',
        files: [
            'src/bower_components/jquery/jquery.js',
            'src/bower_components/jquery-ui/ui/jquery-ui.js',
            'src/bower_components/fullcalendar/fullcalendar.js',
            'src/bower_components/momentjs/moment.js',
            'src/bower_components/angular/angular.js',
            'src/bower_components/angular-ui-calendar/src/calendar.js',
            'test/lib/angular-mocks.js',
            'src/js/app.js',
            'src/js/controllers.js',
            'src/js/filters.js',
            'test/unit/**/*.js'
        ],
```

```
            autoWatch: true,
            frameworks: ['jasmine'],
            browsers: ['Chrome'],
            plugins: ['karma-chrome-launcher', 'karma-jasmine']
        })
    };
```

Testing the controller

Now, we will add three tests for the controller. For this, we will modify `controllers.spec.js`, as follows:

```
'use strict';
describe('controller specs', function () {
    var $scope;
    beforeEach(module('myApp.controllers'));

    beforeEach(inject(function ($rootScope, $controller) {
        $scope = $rootScope.$new();
        $controller("CalCtrl", { $scope: $scope });
    }));

    it("should contain 'Fun with AngularJS' event", function () {
        expect($scope.events[1].title).toBe('Fun with AngularJS');
    });

    it("should contain new event after execution of dayCLick
function", function () {
        $scope.dayClick();
        expect($scope.events[2].title).toBe('new event');
    });

    it("should remove an event", function () {
        $scope.remove(2);
        expect($scope.events.length).toBe(2);
    });
});
```

In the previous spec, we tested the following behaviors of the controller:

- After initialization, `$scope.events` contains an event with the title `Fun with AngularJS`
- The execution of the `$scope.dayClick` function will add an event, titled `new event`

- The execution of `$scope.remove(2)` will remove the event that was added in the previous step and thus, bring the size of the `$scope.events` object back to the original size

Testing the filter

Now, we will finally create a test for the filters. For this, all that we have to do is create a file named `filters.spec.js` in the same directory as `controllers.spec.js`. Karma will run all the tests in this directory, so we don't have to modify any additional files for this. Add the following to `filter.spec.js`:

```
'use strict';
describe('controller specs', function () {
    beforeEach(module('myApp.filters'));

    var tomorrowAtNine = moment().add('days', 1).hours(21).minutes(0).
format();
    var event1 = { start: moment().add('days', 1) };
    var event2 = { start: moment().add('days', 1),
        end: moment().add('days', 1).add('hours', 2) };

    it("should return 'Tomorrow at 9:00 PM' for the appropriate time",
        inject(function (momentFilter) {
            expect(momentFilter(tomorrowAtNine)).toBe("Tomorrow at
9:00 PM");
        }));

    it("should return 'All Day' when event has no specified end date",
        inject(function (durationFilter) {
            expect(durationFilter(event1)).toBe("All Day");
        }));

    it("should return '120 min' for the modified event",
        inject(function (durationFilter) {
            expect(durationFilter(event2)).toBe("120 min");
        }));
});
```

In the previous three tests, we completed the following:

- We set up a few variables to be passed into the tests
- We injected the `moment` filter into a spec, passed the `tomorrowAtNine` variable, and expected the result of the filter to be **Tomorrow at 9:00 PM**

- We injected the `duration` filter, passed the event with no end time, and expected the result to be **All Day**
- We have again injected the `duration` filter, passed the event with an end time that is known to be 2 hours past the beginning of the event, and expected the result to be **120 min**

Building the application

Now, let's see what happens when we allow Grunt to run tests and package the distribution version as follows:

```
mn:Step03-calendar mn$ grunt dist
Running "karma:unit" (karma) task
INFO [karma]: Karma v0.10.2 server started at http://localhost:9876/
INFO [launcher]: Starting browser Chrome
INFO [Chrome 31.0.1650 (Mac OS X 10.8.5)]: Connected on socket Z7TSePUw-yfjRbp-yRp6
Chrome 31.0.1650 (Mac OS X 10.8.5): Executed 6 of 6 SUCCESS (0.24 secs / 0.038 secs)

Running "concat:dist" (concat) task
File "dist/js/fun-with-ui-modules.js" created.

Running "targethtml:dist" (targethtml) task
>> File "dist/index.html" created.

Running "copy:main" (copy) task
Created 1 directories, copied 3 files

Done, without errors.
```

This wasn't that difficult, was it? It is straightforward and not too complicated to achieve decent test coverage with AngularJS.

Summary

We kicked off this chapter by using the Google Maps component with markers for various positions on the map. We installed this component using the Bower package manager.

Then, we went into a little more detail on dependency management with Bower and saw how we can manage all the project dependencies by initializing a Bower project and by editing the managed `bower.json` file.

We also briefly took a look at how to configure Git so that we can keep repository sizes reasonable instead of checking in external artifacts.

Afterwards, we used the `calendar` component to create a calendar with entries from the data managed by AngularJS.

We built a filter based on `moment.js` for date-string formatting so that the dates look more pleasing and used this filter in conjunction with the `calendar` component.

Finally, we created some tests for the application using the calendar. Tests are a great insurance against future trouble. This is because they will alert us or other developers in our team immediately when we break something in an unintended way. This is much more pleasant than having to find this out from an angry customer.

In the next chapter, we will closely examine how we can make use of the `ng-grid` component. The `ng-grid` directive is very useful when it comes to displaying table/grid style data, and as such, it is much more powerful than the good old HTML table. For example, it becomes simple to edit data the way we would in a spreadsheet application such as Microsoft Excel, something that otherwise requires a lot of work.

4
Customizing and Exploring ng-grid

In this chapter, we will take a look at the `ng-grid` component (`http://angular-ui.github.io/ng-grid`), which provides a good and useful data-driven grid. This grid can become quite sophisticated, as we will explore throughout the chapter.

While going through the examples of increasing complexity, we will also introduce the concept of services in AngularJS. Services allow the creation of singleton objects that exist for the life cycle of an application. This becomes useful when a state is being shared between different controllers. It also allows the proper separation of concerns, where data services become a black box from the controller's perspective, with a defined interface for interacting with the data.

With this separation of concerns, we can then start with static data in the service and swap this out for asynchronous loading from a REST API, without having to change the controller.

Let's get started. We have some interesting material ahead:

- Creating our first AngularJS service (in this book, at least)
- Creating a simple grid to display data from the service
- Creating more sophisticated grids that connect to the same controller

Setting up the project

You can copy the `Step03` folder from the previous chapter and modify it to remove all the references to specific parts. This might be a good exercise in itself. In this chapter, we want to talk about `ng-grid`, so we will skip the specific parts. You can use `Step01` of the source code for this chapter as a reference for the exercise or also as a starting point. I have removed all references already in order to have a blank slate for this chapter. I have left a fake test in there, just in case you wonder whether the `grunt dist` task still works instead of failing when there are zero tests.

From this starting point, let's install AngularJS, jQuery, and `ng-grid` through Bower:

```
# bower install ng-grid --save
# bower install angular --save
# bower install jquery --save
```

Remember that the `--save` switch will save the dependency to the `bower.json` file, which is really convenient, because it also automatically finds the latest version for doing so.

While we're at it, we might as well resolve the `angular-mocks` dependencies for our subsequent tests using Bower:

```
# bower install angular-mocks --save
```

Then, we can modify `karma.conf.js` as follows:

```
module.exports = function (config) {
    config.set({
        basePath: '../',
        files: [
            'src/bower_components/jquery/dist/jquery.js',
            'src/bower_components/angular/angular.js',
            'src/bower_components/angular-mocks/angular-mocks.js',
            'src/js/app.js',
            'src/js/controllers.js',
            'src/js/filters.js',
            'test/unit/**/*.js'
        ],
        autoWatch: true,
        frameworks: ['jasmine'],
        browsers: ['Chrome'],
        plugins: ['karma-chrome-launcher', 'karma-jasmine']
    })
};
```

Now for the future, we will get a matching version of `angular-mocks` by running `bower install`, without having to manually manage the file version.

Now, we are ready to get started with a simple grid. First, we need a data source for this. We could just place the data source in the controller, but it might not make much sense in the life cycle of an application. This particular controller might only be active temporarily, but we might want access to the data model from other views as well. A service **singleton** will be a better choice here.

Creating a service in AngularJS

We have previously discussed that a controller might not be a good place to hold on to data. If the controller goes out of the scope, the data residing in the controller will become inaccessible, and garbage will get collected in the JavaScript virtual machine.

Services are singletons, on the other hand. They get instantiated once and stay alive during the entire life cycle of the application. Let's create such a service.

We will make this service known to the application by editing `app.js` as follows:

```
'use strict';

angular.module('myApp', ['myApp.controllers', 'myApp.filters', 'myApp.
services', 'ng-grid']);
```

Then, we create a file named `services.js` that contains the following code:

```
'use strict';

/** data service, manages data model throughout the life cycle of the
application */
angular.module('myApp.services', []).factory('dataService', function
() {
    var exports = {};

    exports.data = ["x", "y"];

    return exports;
});
```

In this service module, we use the `factory` function to initiate the `dataService` singleton object. This object holds an `exports` object, which we will make available to any caller using this service, for example, the controller we will use to access the data.

Now, we will use this from our controller. Modify `controllers.js` as follows:

```
'use strict';
angular.module('myApp.controllers', []).controller('NgGridCtrl',
    ['$scope', 'dataService', function ($scope, dataService) {

    console.log(dataService.data);
}]);
```

Note how we are naming the dependencies as `$scope` and `dataService`. This makes minification possible; otherwise, the application will fall victim to variable name changes during the minification process.

You should now be able to run the application in the browser. You will not see anything rendered, but in the console under the developer tools of your browser, you will see the data array being logged.

Congratulations! We now have a data service. In real-world scenarios, this data would likely come from some server backend. However, this is the beauty of a data service; on the controller side, we need not concern ourselves with this. Instead, we can start with a local data model and then later migrate to fetch it from a REST API without having to change anything in the controller and view at all.

The simple grid view

Now, it is time to do something with the data from our model. After all, we are going to talk about grids in this chapter. Let's work with some data that I find somewhat interesting: the years when different programming languages were invented. We will put this data into the data service, as follows:

```
'use strict';

/** data service, manages data model throughout the life cycle of the
application */
angular.module('myApp.services', []).factory('dataService', function
() {
    var exports = {};

    exports.data = [
        {lang: "Plankalkül", year: 1943, decade: "1940s"},
        {lang: "Fortran", year: 1954, decade: "1950s"},
        {lang: "LISP", year: 1956, decade: "1950s"},
        {lang: "Basic", year: 1964, decade: "1960s"},
        {lang: "Pascal", year: 1970, decade: "1970s"},
        {lang: "C", year: 1972, decade: "1970s"},
```

```
        {lang: "ML", year: 1973, decade: "1970s"},
        {lang: "Postscript", year: 1982, decade: "1980s"},
        {lang: "C++", year: 1983, decade: "1980s"},
        {lang: "Postscript", year: 1982, decade: "1980s"},
        {lang: "Erlang", year: 1987, decade: "1980s"},
        {lang: "Haskell", year: 1990, decade: "1990s"},
        {lang: "Python", year: 1991, decade: "1990s"},
        {lang: "Java", year: 1995, decade: "1990s"},
        {lang: "Javascript", year: 1995, decade: "1990s"},
        {lang: "Scala", year: 2003, decade: "2000s"},
        {lang: "F#", year: 2005, decade: "2000s"},
        {lang: "Dart", year: 2011, decade: "2010s"},
        {lang: "Typescript", year: 2012, decade: "2010s"},
    ];
    return exports;
});
```

Now, we can use this data in the `controller.js` file, which we will modify as follows:

```
'use strict';
angular.module('myApp.controllers', []).controller('NgGridCtrl',
    ['$scope', 'dataService', function ($scope, dataService) {
        $scope.data = dataService.data;
        $scope.gridOptions = {
            data: 'data',
            columnDefs: [
                { field: 'lang', displayName: 'Language',
                  width: "45%" },
                { field: 'year', displayName: 'Year',
                  width: "25%" },
                { field: 'decade', displayName: 'Decade',
                  width: "30%"}
            ]
        };
    }]);
```

Note how we pass the `dataService` object as a dependency of the controller. We then bind this service to `$scope.data`. Then, we define the `gridOptions` object, which defines both the appearance of the grid and the data source. Note in particular how the `data` property is a string that maps how the `data` object is available on the current `$scope` object to the name. This is why we bound `dataService.data` to `$scope.data`. We also specify how the column is labeled and how wide each column is supposed to be as a relative percentage.

Now all that is left to be done is to modify the HTML and the style sheet.
Let's start with `main.css`:

```
body {
    margin: 20px;
    font-family: sans-serif;
}

.gridStyle {
    border: 1px solid #999;
    max-width: 500px;
    height: 600px;
}
```

Here, we defined the grid to be `500px` wide at the most and `600px` tall as well as to have a solid, light-gray border. Specifying `max-width` instead of the `width` will result in the table taking up the entire width of the enclosing element, but only up to a width of at most `500px`.

Finally, we bring it all together in `index.html`, as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular UI NG-Grid</title>
    <link rel="stylesheet" href="css/main.css">
    <link rel="stylesheet" href="bower_components/ng-grid/ng-grid.
css">
</head>
<body ng-app="myApp">

<div ng-controller="NgGridCtrl">
    <div class="gridStyle" ng-grid="gridOptions"></div>
</div>

<!--(if target dev)><!-->
<script src="bower_components/jquery/jquery.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/ng-grid/ng-grid-2.0.7.debug.js"></
script>
<script src="js/app.js"></script>
<script src="js/services.js"></script>
<script src="js/controllers.js"></script>
```

```
<script src="js/filters.js"></script>
<!--<!(endif)-->
<!--(if target dist)>
<script src="js/fun-with-ng-grid.js"></script>
<!(endif)-->
</body>
</html>
```

This is it for our first grid, with `data` properly being provided by a data service. Notice the highlighted code in the previous `<head>` tag. The `<meta name="viewport" content="width=device-width, user-scalable=no">` line is for the mobile-first approach; it sets the content's width to the width of the device and disables zooming.

When you refresh the page, you will now see the following table:

Language	Year	Decade
Plankalkül	1943	1940s
Fortan	1954	1950s
LISP	1956	1950s
Basic	1964	1960s
Pascal	1970	1970s
C	1972	1970s
ML	1973	1970s
Postscript	1982	1980s
C++	1983	1980s
Postscript	1982	1980s
Erlang	1987	1980s
Haskell	1990	1990s
Python	1991	1990s
Java	1995	1990s
Javascript	1995	1990s
Scala	2003	2000s
F#	2005	2000s
Dart	2011	2010s
Typescript	2012	2010s

Pretty good, considering how little we had to do in terms of styling the data. No more dealing with tables when presenting information. Now, click on any of the column headers and you will notice that we will get sorting for free. Even better, we didn't have to write a single line of code for this!

Now, let's make the examples a little more sophisticated.

Grouping the grid

There are two ways of grouping the grid within the `ng-grid` project. Unfortunately, the native version does not work at this point, so we can only take a look at the version relying on jQuery UI.

For the jQuery UI way, we will have to add it to the dependencies of our project. On the command line, run the following command:

```
# bower install --save jquery-ui
```

Once this is done, we need to load it in `index.html`:

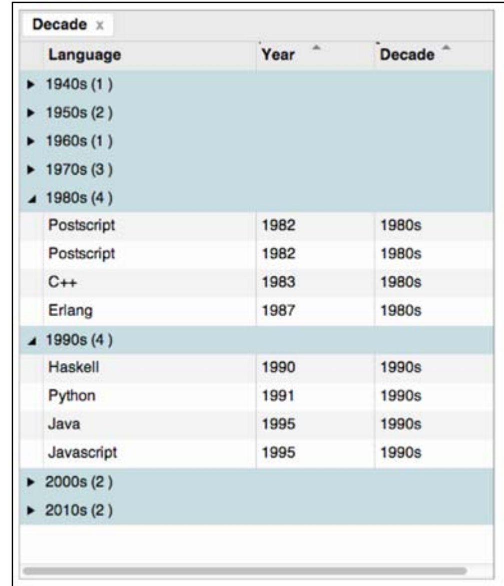
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular UI NG-Grid</title>
    <link rel="stylesheet" href="css/main.css">
    <link rel="stylesheet" href="bower_components/ng-grid/ng-grid.
css">
</head>
<body ng-app="myApp">

<div ng-controller="NgGridCtrl">
    <div class="gridStyle" ng-grid="gridOptions"></div>
</div>

<!--(if target dev)><!-->
<script src="bower_components/jquery/jquery.js"></script>
<script src="bower_components/jquery-ui/ui/jquery-ui.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/ng-grid/ng-grid-2.0.7.debug.js"></
script>
<script src="js/app.js"></script>
<script src="js/services.js"></script>
<script src="js/controllers.js"></script>
<script src="js/filters.js"></script>
<!--<!(endif)-->
```

```
<!--(if target dist)>
<script src="js/fun-with-ng-grid.js"></script>
<!(endif)-->
</body>
</html>
```

Now all that is left is a modification of `controller.js`:

```
'use strict';
angular.module('myApp.controllers', []).controller('NgGridCtrl',
    ['$scope', 'dataService', function ($scope, dataService) {
        $scope.data = dataService.data;
        $scope.gridOptions = {
            data: 'data',
            columnDefs: [
                { field: 'lang', displayName: 'Language',
                  width: "45%"},
                { field: 'year', displayName: 'Year',
                  width: "25%"},
                { field: 'decade', displayName: 'Decade',
                  width: "30%"}
            ],
            showGroupPanel: true,
            jqueryUIDraggable: true
        };
    }]);
```

Now, we can group the rows by dragging a particular column header to the group panel, similar to **Decade** in our example:



Using a master/details view

Next, let's combine what we have so far with a master/details view. We will modify the grid so that we can select a row and then show text input fields below the grid, which will allow us to modify the data model. The changes made here will, of course, be reflected immediately in the grid.

AngularJS is declarative

This is a good place to be reminded of how nice a **declarative approach** to UI development really is. We don't have to concern ourselves with facilitating the data model changes. AngularJS will take care of this for us; all we need to do is define once how the data model is supposed to be rendered in the browser. The framework will then take care of the changes for us.

The changes we need to make are relatively minor. First of all, let's edit `index.html`. We will create a `div` element with the details for selected languages, with an extra comment field that will be reflected in the data model, as follows:

```
<div ng-controller="NgGridCtrl">
    <div class="gridStyle" ng-grid="gridOptions"></div>
    <div class="gridStyle" ng-repeat="item in selection">
        <h1>{{ item.decade}}: {{ item.lang }}</h1>
        <p>{{ item.lang }} was created in {{ item.year }}.
           {{ item.comment }}</p>
        <label for="comment-{{ $index }}">Details:
        <textarea id="comment-{{ $index }}"
         ng-model="item.comment" rows="3"></textarea>
        </label>
    </div>
</div>
```

The following are a few things to note in the previous HTML:

- We use `ng-repeat`, which means that the `div` element will be created for every selected item in the table.
- We create the `comment` property on the item. This didn't exist in the data model, but it will be added when it is filled out and it will persist.
- We create a `<textarea>` tag for this `comment` property. A `<textarea>` tag needs `<label>` that references a particular element ID. However, we potentially have multiple selected items in the detail view, so a static ID will result in conflicts as element IDs have to be unique. We, however, have access to the `$index` object here, which we can use to make the ID unique.

Now, we have to modify the `configuration` object and create a selection array inside `controllers.js`. We will add an additional feature, making the cells editable:

```
'use strict';
angular.module('myApp.controllers', []).controller('NgGridCtrl',
    ['$scope', 'dataService', function ($scope, dataService) {
        $scope.data = dataService.data;
        $scope.selection = [];
        $scope.gridOptions = {
            data: 'data',
            columnDefs: [
                { field: 'lang', displayName: 'Language',
                  width: "45%" },
                { field: 'year', displayName: 'Year',
                  width: "25%" },
                { field: 'decade', displayName: 'Decade',
                  width: "30%" }
            ],
            showGroupPanel: true,
            jqueryUIDraggable: true,
            selectedItems: $scope.selection,
            multiSelect: true,
            enableCellEdit: true
        };
    }]);
```

Now, we could leave things as they are, but let's add some very basic styling inside `main.css` so that the `<label>` and `<textarea>` tags are properly aligned and the detail view is not wider than the grid:

```
.detail {
    max-width: 500px;
}

label textarea{
    vertical-align: middle;
    width: 100%;
}
```

This is how the result will look:

Drag a column header here and drop it to group by that column.

Language	Year	Decade
Erlang	1987	1980s
Haskell	1990	1990s
Python	1991	1990s
Java	1995	1990s
Javascript	1995	1990s
Scala	2003	2000s
F#	2005	2000s

1990s: Javascript

Javascript was created in 1995. It is an interpreted computer programming language. As part of web browsers, implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed.

Details:

It is an interpreted computer programming language. As part of web browsers, implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed.

Note that I have modified the height of the grid for layout purposes. I have selected the JavaScript entry and pasted some text from Wikipedia, which shows immediately in the previous paragraph. You can now deselect the entry, and this text will still be there later; it has been added to the data model. You can also select multiple entries and edit the comment field in any of these. You can also double-click on any cell and edit the cell's contents.

Summary

In this chapter, we took a look at the `ng-grid` component in varying levels of complexity, from simple to complicated. Our first step was to create a simple `ng-grid` view that did nothing but display data from the application's data model.

As the second step, we added the ability to group the grid by column. This allowed us, for example, to group the programming languages by the decade in which they were invented.

Finally, we created a grid with a master/detail view. In the detail view, we added a `comment` property with an associated `textarea` object. This `textarea` object automatically added a property to the objects in the data model, which stayed with the respective object although the row was deselected. In this final version, we also made individual cells editable by double-clicking on the cell.

During the course of this chapter, we also introduced the concept of services. These are important building blocks in web applications because they allow us to manage the state beyond the lifetime of any single controller. This allows us to change the route of the application, to show a different view, and to have the data present when viewing this particular view later again.

In the next chapter, we are going to take a look at animations, which are a neat way to make a web application more sophisticated. Animations are also a lot of fun.

5
Learning Animation

In this chapter, you will learn how to animate DOM elements using AngularJS, which translate into underlying CSS animations but are much simpler to use, as you will see shortly.

We will be covering the following topics in this chapter:

- Creating a simple to-do list that fades in new items and fades out checked items. For this, we will also create a custom filter that ensures that only the items that are not marked as completed are actually shown on the page.
- Using `angular-animate` enables CSS animations without having to do much extra work.
- Moving elements on and off the page.
- Applying **Bezier curves** for more interesting motion patterns.
- Introducing **LESS**; this is helpful to change the duration of animation groups, among many other things.
- Importing readily available animations from `animate.css` and utilizing them inside an AngularJS application.
- Implementing a staggering animation effect.
- Setting up a fallback in JavaScript for CSS animations for incompatible browsers.

Setting up the project

You can copy the final code from the previous chapter and remove the chapter-specific parts if you want to get some additional exercise in navigating around an AngularJS project. Alternatively, you can start with `Step01` from the code provided. Either should work equally well. However, by now, I trust you can find your way around the project files, so I will not cover where to remove which parts from when using the code from the previous chapter as a template. The previous chapters offer much more guidance in doing so.

Let's take a look at what we need to add to the project for the animation to work. First, we need to install `angular-animate`, which we will do by leveraging Bower. We will also use Bootstrap, just to get some good styling of the button. Inside the project folder, run the following command in the shell:

```
bower install --save angular-animate
bower install --save bootstrap
```

This will install the appropriate `bower` component, and it will also update `bower.json` accordingly. Next, we load the JavaScript file in `index.html` and also name the controller for our chapter's project:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular Animation</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
    <link rel="stylesheet" href="css/main.css">
</head>
<body ng-app="myApp">

<div ng-controller="NgAnimateCtrl">
</div>

<!--(if target dev)><!-->
<script src="bower_components/jquery/dist/jquery.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-animate/angular-animate.js"></
script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
```

```
<script src="js/filters.js"></script>
<!--<!(endif)-->
<!--(if target dist)>
<script src="js/fun-with-ng-animate.js"></script>
<!(endif)-->
</body>
</html>
```

Then, we update `controllers.js` and we are good to go:

```
'use strict';
angular.module('myApp.controllers', [])
.controller('NgAnimateCtrl', function ($scope) {
});
```

The modifications described can also be found in `Step01` of the companion code of this chapter if you prefer to start with a clean and prepared slate.

Creating our first animation – a simple to-do list

Prior to AngularJS Version 1.2, it was very difficult to sprinkle animation effects in an AngularJS application as DOM updates occur under the hood by the framework itself, and there is no control over how and when the updates are completed. With AngularJS 1.2, the AngularJS team enables developers to add animations in a snap using `$animate service`. Basically, AngularJS provides animation hooks, also known as events, for some of the built-in directives, such as `ngRepeat`, `ngSwitch`, and so on, as well as for custom directives. These animation events are automatically triggered by AngularJS during the life cycle of various directives. AngularJS animation has evolved as a separate module that is not part of the core: to reduce the overall size of the framework and also because animation is not a necessary piece for every AngularJS application. Imagine that you want to toggle a `div` element using the `ng-hide` directive, AngularJS applies the `ng-hide` CSS class in order to hide the element and removes it to show it again. The `ngHide` directive tracks two events, namely `ng-hide-add` (`ng-hide-add-active`) and `ng-hide-remove` (`ng-hide-remove-active`). For CSS-based animations, the `$animate service` object parses the transitions/animations associated with an element defined in those CSS classes. The service then extracts the animation details, such as transition-property, transition-duration, transition-delay, and so on, that delay DOM updates till the animation is completed. So, this is how animation works and it's no magic.

Going forward, let's explore it with cool examples. In this first example, we will create a simple to-do list with a text input field to add items to the list. Each new item will then be animated upon entering and also upon checking off the item, which will result in the item fading out. For this to work, we first need a text input box. Modify the `NgAnimateCtrl` div inside `index.html`, as follows:

```
<div ng-controller="NgAnimateCtrl">
    <input ng-model="inputText" />
    <button ng-click="addItem()">add</button>

    <div class="item" ng-repeat="item in items | active"><input
type="checkbox" ng-model="item.completed" /> 
        <span ng-bind="item.text"></span>
    </div>
</div>
```

Then, we will need a custom active filter, which as the name suggests, filters our items such that only the active (noncompleted) items are shown on the page.
For this, we modify `filters.js`:

```
'use strict';
angular.module('myApp.filters', [])
    .filter('active', function() {
        return function(items) {
            var filteredItems = [];
            for (var i = 0; i < items.length; i++) {
                if (!items[i].completed) {
                    filteredItems.push(items[i]);
                }
            }
            return filteredItems;
        }
    });
```

Just so the items look a little nicer, we will add some styling in `main.css`. Modify the file as follows:

```
body {
    margin: 20px;
    font-family: sans-serif;
}

.item {
    width: 230px;
    padding: 10px;
    margin: 10px 0;
    background: #cccccc;
    box-shadow: 5px 5px 5px #888888;
    font-weight: bold;
}
```

Now, we need to modify `controllers.js` in order to make the to-do list functional:

```
'use strict';
angular.module('myApp.controllers', ['myApp.filters'])
    .controller('NgAnimateCtrl', ['$scope', function ($scope) {
        $scope.items = [
            { text: "call mum", completed: false },
            { text: "do laundry", completed: false }
        ];
        $scope.inputText = "";
        $scope.addItem = function () {
            $scope.items.push({
                text: $scope.inputText,
                completed: false
            });
            $scope.inputText = "";
        };
    }]);
```

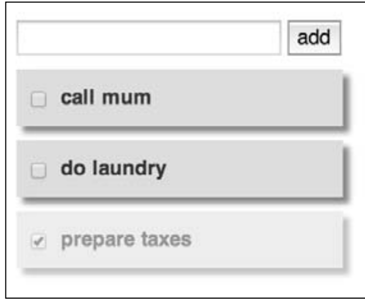
With these modifications, we have a fully functional to-do list application. Nothing is animated yet though. Let's change this by defining a CSS-based animation. First, add the following lines to `main.css`:

```
.item.ng-enter {
    -webkit-transition: 1.2s linear all;
    transition: 1.2s linear all;
    opacity: 0;
}

.item.ng-enter.ng-enter-active {
    opacity: 1;
}

.item.ng-leave {
    -webkit-transition: 1.2s linear all;
    transition: 1.2s linear all;
    opacity: 1;
}

.item.ng-leave.ng-leave-active {
    opacity: 0;
}
```

Next, we need to modify the declaration of the app module to actually use the `angular-animate` module. For this, we modify `app.js`:

```
'use strict';
angular.module('myApp', ['myApp.controllers', 'myApp.filters',
'ngAnimate']);
```

This is all that there is to a simple fade-in / fade-out animation. The following screenshot shows how our application will look during fade-out after you check off the **prepare taxes** task:



So, what actually happens in the preceding example? Let's walk through it step by step:

1. We load the `angular-animate` library.
2. The `angular-animate` library now attaches certain CSS classes, for example, when elements enter or leave, and detaches the classes again when the animation is done.
3. These classes are prefixed with `ng-`, for example, `ng-enter` for the initial state of the element and `ng-enter.ng-enter-active` for the target state of the animation.
4. All that we needed to do then was to define these classes in CSS, as we have done in `main.css` in the case of our to-do list application.
5. We have defined pairs of these classes for our specific item class: `.item.ng-enter`, which defines the initial state of the element, with the duration of how long the animation is supposed to run, and then the `.item.ng-enter.ng-enter-active` class, which defines the target state of the same animation. The initial state has an opacity of `0`, making it fully transparent, and the target state has an opacity of `1`, making it fully visible. During the animation, the specific property transitions between these two values.
6. We have defined the transition to be linear, which means that the changes in the value(s) are distributed evenly over the duration of time. This is called an easing function, and it is specified in this line:

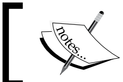
   ```
   transition: 1.2s linear all;
   ```
7. The transition we have used applies to all the eligible properties that can transition. We can omit this, but explicitly specifying this makes it immediately obvious that there are different transitions for different properties. We will see in the next section how to animate different properties differently.

8. Note that we have defined the transition time twice, once in prefixed and again in the nonprefixed form. This is because new CSS features first appear in vendor-specific prefixed form before they are generally adopted into the standard. Check `http://caniuse.com` to find out more about browser support for this feature (and many others): `http://caniuse.com/#search=transition`.

Let's try an other easing function instead. We could try `ease`, but the difference will not be very noticeable in the case of opacity transitions, much more so when changing the size or position. However, distinct steps are very noticeable:

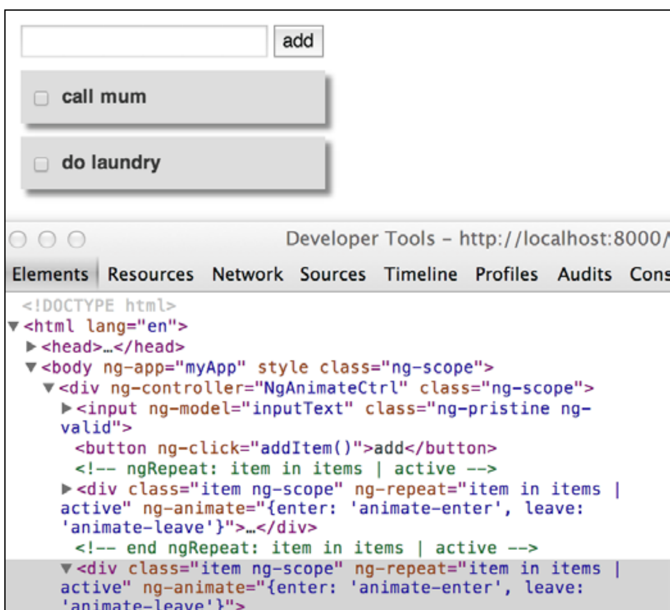
```
transition: 1.2s steps(5) all;
```

In this case, the value transition is not even, but instead takes place in the number of specified, distinct steps, which can give an interesting effect as well.

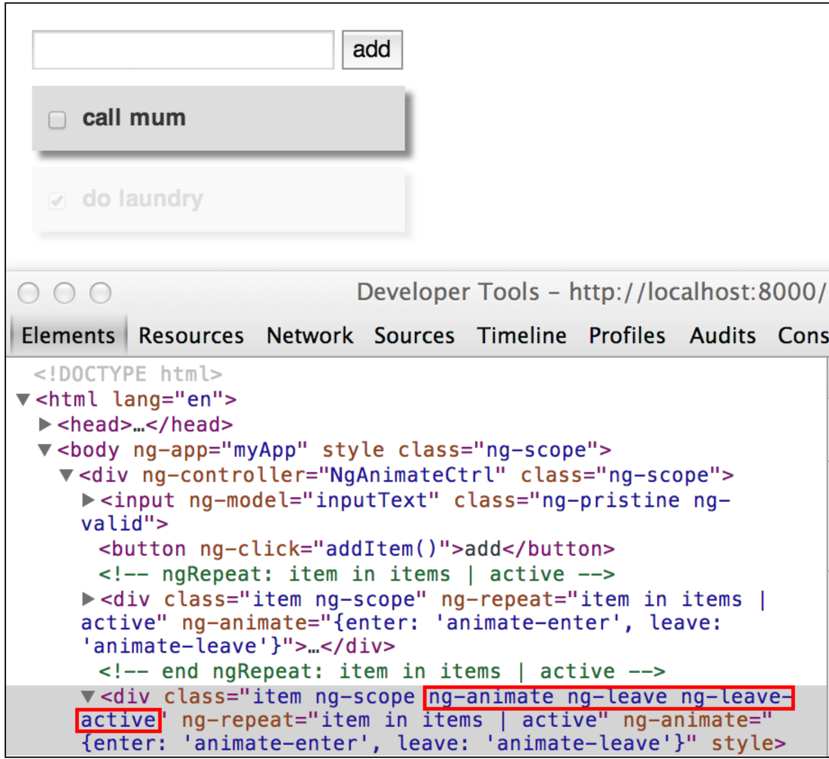


There are different events/hooks used by various directives that you can learn more about in the documentation.

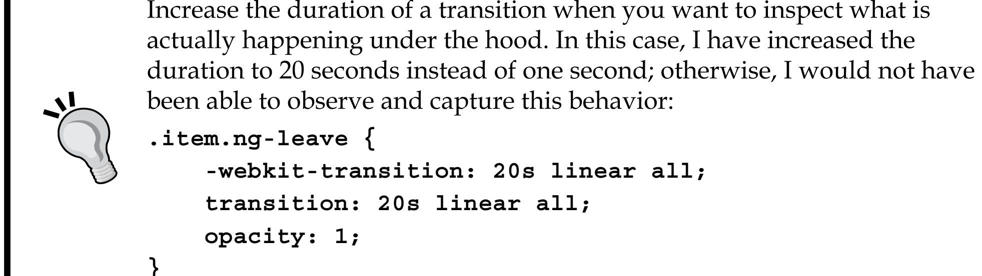
I have claimed that AngularJS assigns these classes automatically, but don't take my word for it; instead let's see this in action by opening the web inspector. In Chrome, for example, right-click on an element and choose the inspect element. We will find that at this moment, when no animation is active, the classes assigned to the element are `item` and `ng-scope`, as shown in the following screenshot:



Now, when we mark an item as completed, we will see that the CSS classes we have defined are automatically assigned to the element for the duration of the animation, as shown in the following screenshot:



This is what `angular-animate` does by adding or removing such CSS classes dynamically to enable the animation in AngularJS applications.



Increase the duration of a transition when you want to inspect what is actually happening under the hood. In this case, I have increased the duration to 20 seconds instead of one second; otherwise, I would not have been able to observe and capture this behavior:

```
.item.ng-leave {
    -webkit-transition: 20s linear all;
    transition: 20s linear all;
    opacity: 1;
}
```

Moving elements around on the page

Let's play around and animate additional properties. Imagine that the items in our to-do application fly in from the top until they reach their correct vertical position and then make a 90-degree turn to slide into their slot horizontally, all while fading in from invisibility to full visibility. Then, on completing an item, it will do the opposite, slide to the right, and then fall from the page while fading away.

All we need to change is the CSS for this scenario; everything else stays the same. Modify `main.css` as follows:

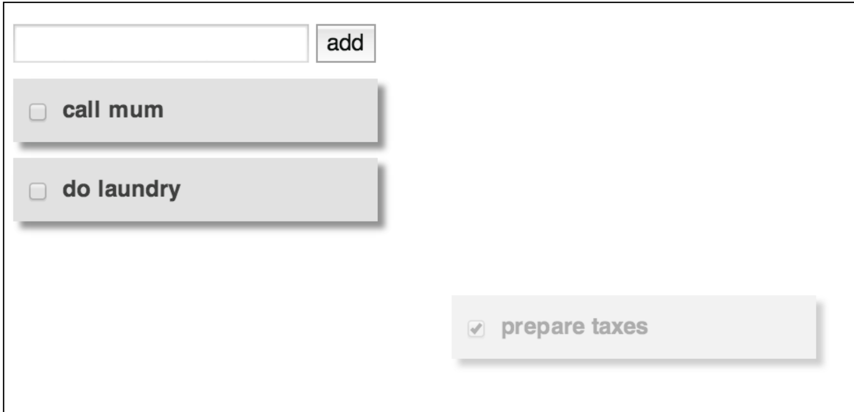
```
body {
        margin: 100px 40px;
    font-family: sans-serif;
}

.item {
    width: 230px;
    padding: 10px;
    margin: 10px 0;
    background: #ddd;
    box-shadow: 5px 5px 5px #888888;
    font-weight: bold;
    position: relative;
}

.item.ng-enter {
    -webkit-transition: left 1s ease 1s, top 1s ease,
                        opacity 2s linear;
    transition: left 1s ease 1s, top 1s ease,
                opacity 2s linear;
    opacity: 0;
    left: 300px;
    top: -500px;
}

.item.ng-enter.ng-enter-active {
    opacity: 1;
    left: 0;
    top: 0;
}

.item.ng-leave {
    -webkit-transition: left 1s ease, top 1s ease 1s,
                        opacity 2s linear;
    transition: left 1s ease, top 1s ease 1s,
                opacity 2s linear;
```

```
        top:0;
        left:0;
        opacity: 1;
    }

    .item.ng-leave.ng-leave-active {
        opacity: 0;
        left: 300px;
        top: 1500px;
    }
```

So, what is going on in the previous CSS?

1. First of all, we have adjusted the margin at the top to make the move from the top a little more visible.
2. Next, we set the `item` class to the `relative` position, which means that the position is relative to where it would usually be (which is in the places where we have previously seen them).
3. Then, we set top: `0` and left: `0` as the target state of `ng-enter` and as the starting point of the `ng-leave` animation. This, in effect, means that during the majority of the life cycle of the `div` element, it will be positioned as expected without the `relative` position property.
4. As the starting point of the `ng-enter` transition, we have chosen a position that is `300px` to the right of the target position and off the page to the top.
5. As the destination of the `ng-leave` transition, we have chosen a place at the bottom of the page, resulting in the illusion of the element falling off the page.

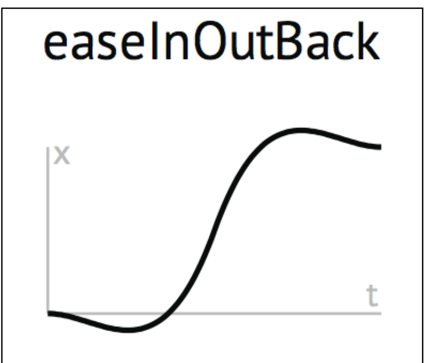
Let's take a look at the result, with the limitation, of course, that we cannot capture motion well in this book:



Easing functions

So, in the preceding CSS, we created some motion on the page, but we can still spice this up by making the movement a little more interesting and less predictable. CSS animations give us a great tool for this, the so-called `cubic-bezier` function, which lets us specify exactly how we want the movement to look. You can find more information about these at `http://en.wikipedia.org/wiki/Bézier_curve`. For our purpose, you should suffice to know that a Bezier curve specifies two control points in two-dimensional space, which defines a curve that will be used to determine any point of the value transition over time. You can find a nice interactive tool to create these curves at `http://cubic-bezier.com`.

There is also a great open source cheat sheet available online, which the curve used in this example originates from. Refer to the following shape:



You can find this and many other interesting curves at `http://easings.net/#easeInOutBack` and the source code at `https://github.com/ai/easings.net`.

For this, we specify a Bezier curve, describing the change of the value in question. Again, we only need to modify `main.css`:

```
.item.ng-enter {
    -webkit-transition: left 1s cubic-bezier(0.68, -0.55,
       0.265, 1.55) 1s, top 1s cubic-bezier(0.68, -0.55,
       0.265, 1.55), opacity 2s linear;
    transition: left 1s cubic-bezier(0.68, -0.55, 0.265,
       1.55) 1s, top 1s cubic-bezier(0.68, -0.55, 0.265,
       1.55), opacity 2s linear;
    opacity: 0;
    left: 300px;
    top: -500px;
}
.item.ng-enter.ng-enter-active {
```

```
        opacity: 1;
        left: 0;
        top: 0;
    }

    .item.ng-leave {
        -webkit-transition: left 1s cubic-bezier(0.68, -0.55,
           0.265, 1.55), top 1s cubic-bezier(0.68, -0.55, 0.265,
           1.55) 1s, opacity 2s linear;
        transition: left 1s cubic-bezier(0.68, -0.55, 0.265,
           1.55), top 1s cubic-bezier(0.68, -0.55, 0.265, 1.55)
           1s, opacity 2s linear;
        top:0;
        left:0;
        opacity: 1;
    }
```

Take a look at your end to see how the animation is orchestrated.

Using LESS to scale entire animations

It can become quite annoying to tweak the duration of an entire animation because we will have to change every single duration inside CSS. Consider a much more complex animation and you can imagine how tedious this would be. However, for this we can use **LESS** (**Leaner CSS**), a language that compiles into CSS and allows us to use variables to scale individual values. We will cover LESS in much more detail later, but it is worth introducing it here already. If you are not using it yet, you really should. Of course, you can also use **SASS**, but if you are using SASS, you will easily be able to achieve the same in SASS. I believe using SASS over LESS or vice versa is a matter of your choice, and it should not really stop you from using SASS just because we have covered LESS in this chapter. There is an in-depth article on SASS versus LESS that I would like you to check out in order to avoid confusion, which can be found at `http://css-tricks.com/sass-vs-less/`. Typically, the compilation of LESS (or SASS) into CSS will take place inside a grunt task, but for development purposes, we can also run the compilation phase in the browser. We will install it using Bower:

```
bower install --save less
```

As the LESS compiler only looks for `*.less files` in the page, we have to rename `main.css` to `main.less` first. Then, we load `less.js` and the new `main.less` file inside `index.html`:

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
       user-scalable=no">
    <title>Angular Animation</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
    <link rel="stylesheet/less"  type="text/css"
        href="css/main.less">
    <script src="bower_components/less/dist/less-
        1.5.1.js"></script>
</head>
```

Now when you load the page, it should look exactly as before. Under the hood, the browser now compiles the LESS into CSS. LESS is a superset of CSS, meaning that it contains the entirety of the CSS language. Thus, the current CSS simply remains unchanged so far.

Now, we can add a scaling variable and also add variables for the individual phases in `main.less`, which we will use as a multiplier for all the durations in our animation example. Edit `main.less` as follows:

```
@time-scale: 10;
@enter-left: 1s;
@enter-left-delay: 1s;
@enter-top: 1s;
@enter-fade: 2s;
@leave-left: 1s;
@leave-top: 1s;
@leave-top-delay: 1s;
@leave-fade: 2s;

.item.ng-enter {
  transition: left (@time-scale * @enter-left)
     cubic-bezier(0.68, -0.55, 0.265, 1.55) (@time-scale *
     @enter-left-delay),
     top (@time-scale * @enter-top) cubic-bezier(0.68, -0.55,
     0.265, 1.55),
     opacity (@time-scale * @enter-fade) linear;
  opacity: 0;
```

```
    left: 300px;
    top: -500px;
  }
  .item.ng-enter.ng-enter-active {
    opacity: 1;
    left: 0;
    top: 0;
  }

  .item.ng-leave {
    transition: left (@time-scale * @leave-left)
       cubic-bezier(0.68, -0.55, 0.265, 1.55),
       top (@time-scale * @leave-top) cubic-bezier(0.68, -0.55,
       0.265, 1.55) (@time-scale * @leave-top-delay),
       opacity (@time-scale * @leave-fade) linear;
    top:0;
    left:0;
    opacity: 1;
  }
```

The preceding LESS code will compile into plain old CSS, with the individual values multiplied by the `@time-scale` variable wherever we have specified. We could have left the individual values in the place where they were, but it is often easier if the variables are in one place. Now, we only need to change a single variable when we want to scale the entire animation. Of course, we could also use separate variables to scale the `ng-enter` and the `ng-leave` groups separately, or whatever other grouping we can think of.

Note that in the previous code, we didn't specify the prefixed `-webkit-` CSS property to make the code a little neater and less repetitive. If you need it, you can just create a copy of the transition with the prefix and the otherwise unchanged syntax.

Using animate.css

Now that we have created a transition manually, let's make things a bit simpler for ourselves by making use of the great open source `animate.css` library, which comes with many interesting animations that we can easily use in our AngularJS application. You can find out more about this library and the source code at `https://github.com/daneden/animate.css`. We install animate.css by running the following in the command line:

```
bower install --save animate.css
```

Next, we modify `index.html` to load `animate.css`:

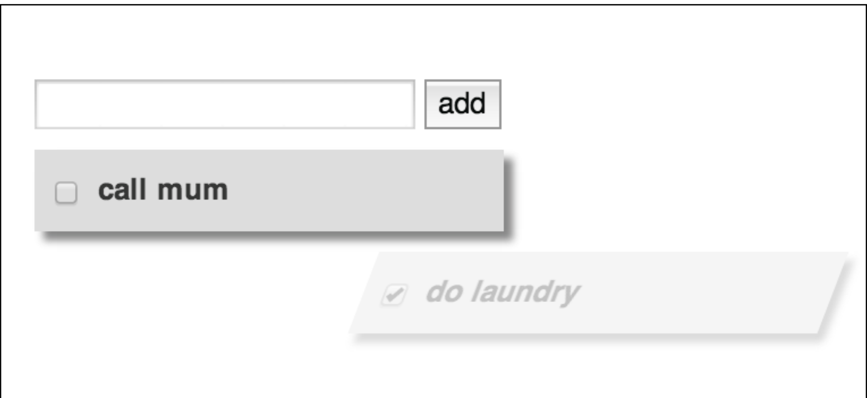
```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular Animation</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
    <link rel="stylesheet"
       href="bower_components/animate.css/animate.css">
    <link rel="stylesheet/less"  type="text/css" href="css/main.less">
    <script src="bower_components/less/dist/less-1.5.1.js"></script>
</head>
```

With these modifications in place, we can now edit our `main.less` file, which will become significantly simpler, despite the fact that we are now also introducing prefixed versions for legacy Mozilla and Microsoft browsers:

```
body {
  margin: 100px 40px;
  font-family: sans-serif;
}

@enter: 1s;
@leave: 1s;

.item {
  width: 230px;
  padding: 10px;
  margin: 10px 0;
  background: #ddd;
  box-shadow: 5px 5px 5px #888888;
  font-weight: bold;
  position: relative;
}
.item.ng-enter {
  -webkit-animation: lightSpeedIn @enter;
  -moz-animation: lightSpeedIn @enter;
  -ms-animation: lightSpeedIn @enter;
  animation: lightSpeedIn @enter;
}

.item.ng-leave {
  -webkit-animation: lightSpeedOut @leave;
```

```
    -moz-animation: lightSpeedOut @leave;
    -ms-animation: lightSpeedOut @leave;
    animation: lightSpeedOut @leave;
  }
```

Note that all we really need to do is call the named animations from `animate.css` inside the `ng-enter` and `ng-leave` classes. This is how the result looks when checking off an item:



Pretty good, particularly considering that all we had to do was to import a library. There are all kinds of great animations available in `animate.css`, and the source code is available too, making it a great way to learn how these animations are actually created with CSS. CSS animations are a mighty feature, and we can only scratch the surface here.

Staggering animations

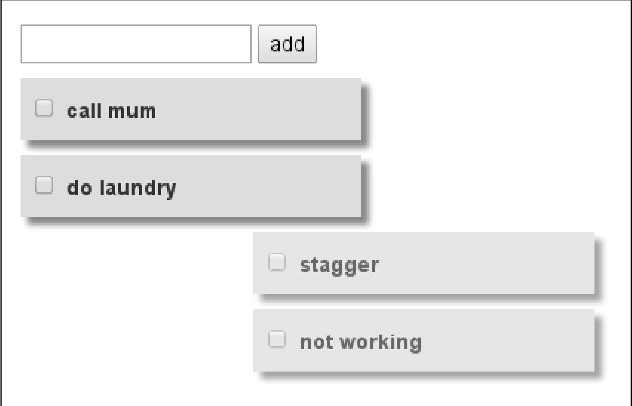
AngularJS 1.2 or higher introduced a new feature in its `ngAnimate` module, that is, a native staggering support to CSS transitions and keyframe animations without much CSS code from our end. The built-in `$animate` service does all the Grunt work to render animations in a staggering fashion. To get more clarity on this, let's take a look at the following example.

Let's modify the `addItem()` method from `controllers.js` to add multiple to-dos at the same time with a comma as the separator. We'll then split the `inputText` object by a comma to add each to-do separately:

```
$scope.addItem = function () {
var arrTodos = $scope.inputText.split(',');

    for(var i = 0; i < arrTodos.length; i++) {
```

```
            $scope.items.push({
                text: arrTodos[i],
                completed: false
            });
        }
            $scope.inputText = "";
         };
```

With this change, we want to see how multiple to-do items are animated simultaneously. If you've done said modifications, then you'll see all the to-dos move at the same time in the following screenshot, which looks quite weird:

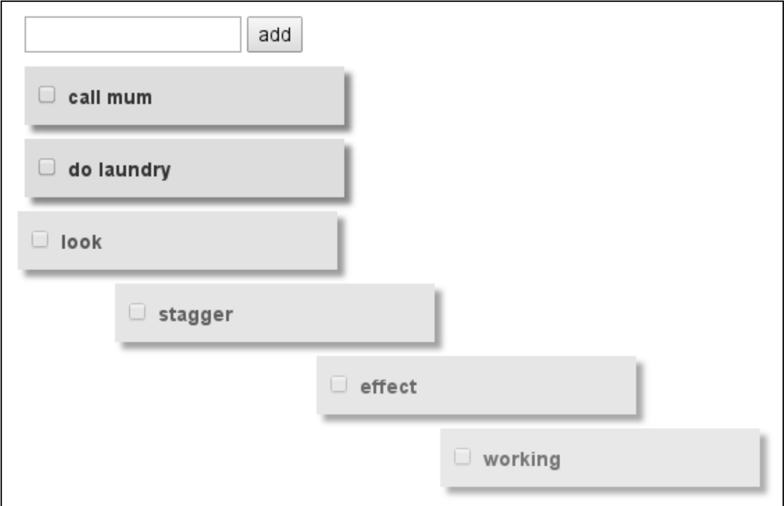


What if we could issue a slight delay in between that would bring a *curtain-like* effect to the animation? This is what staggering animations do. They allow us to add a delay while animating multiple elements.

The stagger effect can be performed by creating a `ng-EVENT-stagger` CSS class and attaching it to the base class used for the animation. The `style` property expected within the `stagger` class can be a `transition-delay` or an `animation-delay` property (or both, depending on your requirement). We'll extend the example we saw in the earlier section, *Using LESS to scale entire animations*, by adding `staggering` classes in `main.less`, as follows:

```
.item.ng-enter-stagger,
.item.ng-leave-stagger,
.item.ng-move-stagger {
-webkit-transition-delay: 0.1s;
    transition-delay: 0.1s;

-webkit-transition-duration:0;
transition-duration:0;
 }
```

Note that a 100 ms delay will be used between each successive enter/leave/move operation. The last two lines are pretty much required to prevent any kind of CSS inheritance issues, as the chances are that the `transition-duration` property is being accidentally inherited and, therefore, the stagger may not work as expected. Here is how the stagger animation will look:



Understanding how staggering works

Well, it's fairly simple, but it is complex under the hood. The `ngAnimate` service looks for the `ng-EVENT-stagger` CSS class associated with the element and extracts the animation/transition details to perform the staggering effect. It also takes care of when the animation starts, ends, and how much delay was proposed for a stagger operation so that each animation is spaced out. Note that the service is only there to parse the timing details in order to delay the addition/removal of the `ng-*` classes accordingly.

JavaScript-defined animations

We may have to rely on JavaScript sometimes to perform the animation because of browser compatibility issues with CSS animations/transitions. As you know, the moment we introduce `ngAnimate` as a dependency to our application, the `$animate` service starts adding/removing the `ng-*` classes on/off the elements during the course of the animation, but within 0 ms, in case we do not have the CSS animations in place. This gives us a control over how we want to enable the animation, that is, either using CSS or JavaScript.

Let's create a JavaScript fallback for the example from the *Using animate.css* section. Add the following to `controllers.js`:

```
.animation('.item', function() {
    return {
        enter: function(element, done) {
            element.css({'opacity': 0, 'margin-left': '-230px'});
            element.animate({'opacity': 1, 'margin-left': 0}, 500,
done);
        },
        leave: function(element, className, done) {
            element.animate({'opacity': 0, 'margin-left': -230}, 500,
done);
        }
    }
 });
```

So, what's happening here?

1. First of all, we've used the `animation` service (such as controller) in AngularJS to enable class-based animation in JavaScript.
2. The animation name (`.item`) should begin with a dot, resembling a CSS class definition.
3. As we are dealing with `ngRepeat`, we have to take care of the `enter` and `leave` events for this particular example. This is applicable to other directives too, such as `ngView`, `ngInclude`, `ngSwitch`, and `ngIf`. For the remaining directives, such as `ngClass`, `ngShow`, `form`, and `ngModel`, there are the `addClass` and `removeClass` events to consider.
4. Going ahead with the `enter` event, we mentioned the `start` and `end` states. Initially, we want an element to be out of the viewport, with a negative margin, and hidden with an opacity of `0`. Moving towards the destination, we reduced the negative margin to `0` by animating within the viewport and making it visible.
5. For the `leave` event, it is reversed.
6. Note that the `done` callback is extremely important to be triggered after the animation is over. For CSS animations, AngularJS calls it internally. The purpose of the `done` callback is to remove the `ng-*` classes on the element upon the completion of the animation. AngularJS may not remove the element (mainly in `ngRepeat`, `ngIf`, and `ngSwitch`), assuming that the animation is still in progress if we fail to call the `done` method, and may not detach those classes.

However, there is one problem: when a CSS animation is supported by the browser, both the CSS and JS animations will collapse. To fix this, we have to know whether the browser has support for transitions/animations. For this, we'll use **Modernizr**, a feature detection library for HTML5/CSS3.

We'll install Modernizr using the following in the command line:

```
bower install --save modernizr
```

This will install the latest version of Modernizr in the `bower_components/` directory and then we'll have to insert it in `index.html`:

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular Animation</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
    <link rel="stylesheet"
       href="bower_components/animate.css/animate.css">
    <link rel="stylesheet/less"  type="text/css" href="css/main.less">
    <script src="bower_components/less/dist/less-1.5.1.js"></script>
    <script src="bower_components/modernizr/modernizr.js"></script>
</head>
```

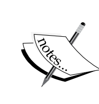
Next, we'll slightly update the defined JavaScript animation to seamlessly fall back. Modernizr has a bunch of tests to check browser support for various features. One of them is `cssanimations`. Let's add it, as follows:

```
.animation('.item', function() {
   if (Modernizr.cssanimations) return angular.noop;
    return {
        enter: function(element, done) {
            element.css({'opacity': 0, 'margin-left': '-230px'});
            element.animate({'opacity': 1, 'margin-left': 0}, 500,
done);
        },
        leave: function(element, className, done) {
            element.animate({'opacity': 0, 'margin-left': -230}, 500,
done);
        }
   }
});
```

In this case, we do the following steps:

1. We first check for the support of `cssanimations` in a browser using `Modernizr.cssanimations`—it will return `true` if there is support; otherwise, it will return `false`.
2. The `animation` service should return an empty object/function if we do not want to overwrite CSS animations. We've returned `angular.noop` to save some keystrokes: a built-in method in AngularJS that performs no operation, which is similar to `function() { }`.
3. If you use the older version of Internet Explorer, that is, version 8 or 9, then you can catch the JavaScript animation in motion.

Now, our animation will work seamlessly in evergreen browsers that support CSS3 animations as well as old browsers with JS fallback.



> To add animation hooks to custom directives, Matias' (the core angular team member) article is worth taking a look at, which can be found at `http://www.yearofmoo.com/2013/08/remastered-animation-in-angularjs-1-2.html#rolling-out-animations-with-your-own-directives`.

Summary

In this chapter, we took a look at the animation functionality in AngularJS and how easy it is to simply use CSS transitions/animations in order to make elements dance or fall back to JavaScript animations otherwise. First, we created a simple to-do list application with an animated entry and exit of items using a CSS-based animation. In this application, we also created a custom filter to only show the items that are not marked as completed. We looked at what `angular-animate` is doing under the hood to facilitate CSS animations. We created motion on the page inside the `ng-enter` and `ng-leave` phases. We created more interesting motion paths by making use of the `bezier-curve` function in CSS. We started using LESS to make our lives easier by introducing variables into CSS and letting us specify values in a central place. We used `animate.css` as a simple way of importing rather sophisticated animations. We also created a staggering animation to space out multiple elements being animated simultaneously and learned how AngularJS manages to do so. Finally, we set up a fallback for CSS transitions/animations in JavaScript. Also, we trained Modernizr to detect the HTML5/CSS3 feature with simple to-use utility functions. In the next chapter, we will be covering data-driven charts.

6

Using Charts and Data-driven Graphics

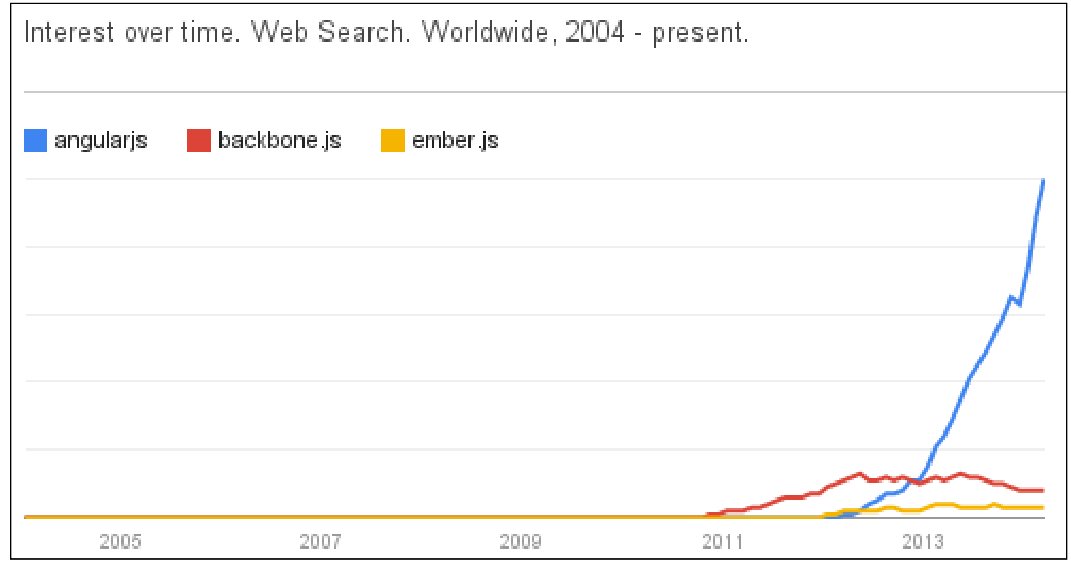
In the previous chapter, we covered animations in AngularJS and the reason why `$animate` service exists in the first place to enable CSS or JavaScript animations seamlessly in AngularJS applications, which was not possible before. In this chapter, we will learn the benefits of using charts, how to implement them in AngularJS, and in the end, we will make it dynamic with real-world data to create a simple dashboard application.

We will cover the following topics in this chapter:

- Creating a simple bar chart in pure CSS
- Extending the same with static data and converting it into a widget using the AngularJS directives API
- Using an open source alternative to our CSS-based chart and learning how charting libraries are better than reinventing the wheel
- Consuming RESTful APIs to create a dashboard application to portray real-world data using various types of charts with the D3 library

Understanding the importance of charts

It's always good to know the purpose prior to learning anything, and I'm pretty sure you must be wondering why we need to learn about charts. Charts are an extremely effective medium to convey the importance of any sort of data in a graphical manner. Let's start with a story. Imagine that you are trying to convince your boss to allow you to use AngularJS for your next project. Although there are many ways to win them over, you might choose a simple path to just compare your current JavaScript framework or library with AngularJS and try explaining to them what AngularJS can do and the current stack cannot. You might also brag about Google backing it, thereby ensuring its authenticity and long-term success, and expect your boss to give you the green signal but they do not even budge. Finally, you would show them the following line chart and the rest will be history:



Creating a bar chart

Let's handcraft a simple bar chart in pure CSS to strengthen our learning before using the ready-made solutions. A bar chart, also known as bar graph, is a chart with rectangular bars with lengths proportional to the values that they represent. It is very useful to display categorical data. In this first example, we'll use static data to display on the graph.

To begin with, we'll use the same template from the `Step01` folder of *Chapter 5, Learning Animation*. Let's replace `index.html` with the following markup:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, user-scalable=no">
<title>Angular Bar Chart: Pure CSS</title>
<link rel="stylesheet" type="text/css" href="css/main.css">
</head>
<body>
<div class='container'>
<div class='bar blue'>Blue - 50%</div>
<div class='bar green'>Green - 80%</div>
</div>
</body>
</html>
```

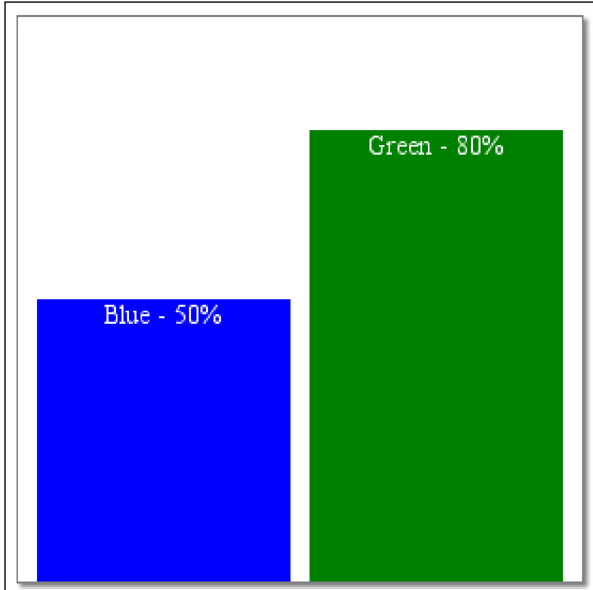
Then, we'll add very straightforward CSS styles in `main.css` to bring these inert `div` objects to life:

```
      .container {
  position: relative;
  width: 300px;
  height: 300px;
  border: 1px solid gray;
  margin: 0 auto;
  box-shadow: 2px 2px 2px gray;
  font-size: 14px;
  color: white;
  padding-right: 10px;
}
.bar {
  position: absolute;
  bottom: 0;
  text-align: center;
  word-break: break-all;
}
.blue {
width: calc(300px/2 - 10px);
  height: 50%;
  background: blue;
  left: 10px;
}
```

```
.green {
width: calc(300px/2 - 10px);
  height: 80%;
  background: green;
 left: calc(300px/2 + 10px);
}
```

You might be familiar with most of the styles here, so I'll just focus on the highlighted ones because we have to come up with a formula to calculate the same in JavaScript in the next section. First of all, we want the width of each bar to be exactly half the container size, excluding some padding around. We'll also have `10px` padding between the bars and the container. Note that the `calc()` method is really handy here to perform calculations and make it readable for developers than using some precalculated arbitrary values in place.

If all is good, then you'll see our CSS bar chart, as follows:



Making the bar chart data driven

By now, you've seen how easy it is to create a static bar chart, so let's customize it to lay multiple bars to showcase a large dataset. In the previous section, we've not used `angular.js` but for the data-driven charts, it will be required. So, let's add it by running the following command on the shell:

```
bower install --save angular
```

This will install the latest version of `angular.js` in the `bower_components` folder. Now update `index.html` to wrap the chart container within `BarChartCtrl` to have a proper encapsulation for the static data, as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, user-scalable=no">
<title>Angular Bar Chart: Data Driven</title>
<link rel="stylesheet" href="css/main.css">
</head>
<body ng-app="myApp">
<div ng-controller="BarChartCtrl">
<div class='container'>
  <div class='bar blue'>Blue - 50%</div>
  <div class='bar green'>Green- 80%</div>
</div>
</div>
<script src="bower_components/angular/angular.js"></script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
</body>
</html>
```

You are aware of the use of `ng-app` to bootstrap the AngularJS application. We'll go straight into updating `app.js` and `controllers.js`. Let's put the following line into `app.js` quickly:

```
angular.module('myApp', ['myApp.controllers']);
```

Add the following code in `controllers.js`. Here, we have defined the container's size details to be further customizable and a collection of data to render on the graph:

```
angular.module('myApp.controllers', [])
.controller('BarChartCtrl', function ($scope) {
  $scope.container = { width: 300, height: 300, gap: 10 };

  $scope.bars = [
    { color: 'blue', percentage: 50 },
    { color: 'orange', percentage: 60 },
    { color: 'red', percentage: 10 }
  ];
});
```

Let's use this data in `index.html`, as follows:

```
<div ng-controller="BarChartCtrl">
<div class='container' ng-style="setContainer()">
<div class='bar' ng-repeat="bar in bars"ng-style="setDetails(bar,
$index)">{{bar.color}} - {{bar.percentage}}%</div>
</div>
</div>
```

The following explains the preceding code:

- First of all, we apply the width and height to `div.container` using `ng-style` and passing a method that will return the expected styles as defined in `$scope.container`. The `ng-style` directive in AngularJS lets you apply in-line styles. You can either pass a function that returns an object that consists of a key/value pair of CSS properties to be applied on the element, or directly pass an object.
- As we have a collection of data to iterate through, AngularJS comes up with a really prolific directive named `ng-repeat` to iterate over a collection and will render the associated element three times for blue, orange, and red bars, respectively.
- The `ng-repeat` directive also gives access to each item in the collection to get the item details. As you can see, we've retrieved `{{bar.color}}` and `{{bar.percentage}}` out of it.

As of now, the methods passed to `ng-style` do not do anything because we've not defined them yet in the controller. Let's add the following methods in `controllers.js`:

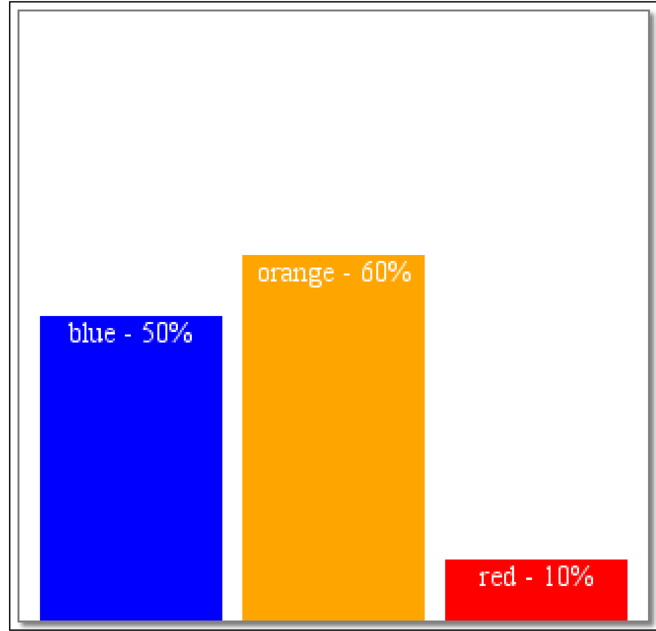
```
  $scope.setContainer = function() {
    return {
      'padding-right': $scope.container.gap + 'px',
      'width': $scope.container.width + 'px',
      'height': $scope.container.height + 'px'
    };
  };
  $scope.setDetails = function(bar, index) {
    var barWidth = $scope.container.width/$scope.bars.length - $scope.
container.gap;
    return {
      height: bar.percentage + '%',
      background: bar.color,
      width: barWidth + 'px',
      left: $scope.container.gap + ($scope.container.gap +
barWidth)*index + 'px'
    };
  };
```

The `setContainer()` method just returns a plain object containing the styles to be applied on the container. In the `setDetails()` method, we calculate the ideal width for each bar in order to fit them all inside the container without overflow. Notice that we've passed bar and index as parameters to get the details for each bar to set the proper height and position.

Finally, we'll just update our `main.css` file to remove the hardcoded styles, as follows:

```
.container {
  position: relative;
  border: 1px solid gray;
  margin: 0 auto;
  box-shadow: 2px 2px 2px gray;
  font-size: 14px;
  color: white;
}
.bar {
  position: absolute;
  bottom: 0px;
  text-align: center;
  word-break: break-all;
}
```

If all is well, then you'll be amazed to see the following chart. Notice how each bar has equal width and is perfectly spaced out:



You can experiment with it by adding more data in the `$scope.bars` collection and see how it fits in.

Converting the bar chart into a widget

Imagine that you want to use this data-driven bar chart in many places in your application and obviously do not want to repeat the same code everywhere by creating a separate controller for each instance. The better solution to this problem is to convert it into a reusable widget with the AngularJS directives API.

Let's update `index.html` to get rid of the controller and add `directives.js` instead:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, user-scalable=no">
<title>Angular Bar Chart:: Widget </title>
<link rel="stylesheet" href="css/main.css">
</head>
<body ng-app="myApp">
<bar-chart
     width="400"
     height="400"
     gap="5"
     data="[{ color: 'blue', percentage: 50 }, { color: 'orange',
percentage: 60 }, { color: 'red', percentage: 10 }]">
</bar-chart>
<script src="bower_components/angular/angular.js"></script>
<script src="js/app.js"></script>
<script src="js/directives.js"></script>
</body>
</html>
```

Creating a bar-chart directive

That's right! We've created our own element, `bar-chart`, to solve the problem. You can pass the container's width and height, a breathing space between a container and bars, and of course, the data to it. Let's revive it by adding the following code in `directives.js`:

```
angular.module('myApp.directives', []).directive('barChart',
function() {
  return {
restrict: 'E',
```

```
template: "<div class='container' ng-style='setContainer()'>\
<div class='bar' ng-repeat='bar in bars' ng-style='setDetails(bar,
$index)'>{{bar.color}} - {{bar.percentage}}%</div>\
</div>",
    link: function(scope, element, attrs) {
scope.container = {
        width:  scope.$eval(attrs.width)  || 300,
        height: scope.$eval(attrs.height) || 300,
        gap:    scope.$eval(attrs.gap)    || 10
      };
scope.bars = scope.$eval(attrs.data) || [];
scope.setContainer = function() {
        return {
          'padding-right': scope.container.gap + 'px',
          'width': scope.container.width + 'px',
          'height': scope.container.height + 'px'
        };
      };
scope.setDetails = function(bar, index) {
        var barWidth = scope.container.width/scope.bars.length -
scope.container.gap;
        return {
          height: bar.percentage + '%',
          background: bar.color,
          width: barWidth + 'px',
          left: scope.container.gap + (scope.container.gap +
barWidth)*index + 'px'
        };
      };
    }
  };
});
```

There are a few things to note here:

- The directive name, `barChart`, should be in Camel Case unlike how it was used in the DOM as `<bar-chart>`.
- The restrict option lets you restrict how the directive should be used in HTML. The **E** stands for Element. Other variations are **C** (CSS Class), **A** (Attribute), and **M** (Comments). If you want to use it both as an Element and Attribute, then you can put **EA** in the option.
- The `template` option takes the markup you want to put inside the directive once it is registered. In our case, we've placed the container- and bar-related markups here without any modification.

- The `link` function is a postdirective registration callback method where you can simply do any sort of DOM manipulation, attach events, and bind methods on the scope to be further used in the template. You can see that we've moved our methods named `setContainer()` and `setDetails()` in it.
- The `attrs` parameter in the `link` function returns all the attributes and their values (post evaluation using `scope.$eval`) associated with the element.

Then update `app.js` as well:

```
angular.module('myApp', ['myApp.directives']);
```

Now, you are free to use as many `chart` instances as you want in your application without duplicating the original piece of code. That's the beauty of AngularJS!

Using Angular Google chart tools

There are different variations of charts to display a large data set; sometimes, a bar chart is not an ideal choice. Google chart tools (`https://developers.google.com/chart`) are powerful, simple to use, and a free rich gallery of interactive charts and data tools. However, Google has not ported it to AngularJS yet, but the AngularJS community, which is so vast and full of enthusiastic people, made this possible. We are going to use a AngularJS wrapper for the same called `angular-google-chart`. Let's build a pie chart with it.

Install `angular-google-chart` using the following commands in the terminal:

```
bower install --save angular-google-chart
```

Now, update `index.html` with the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular Pie Chart: Google Chart</title>
</head>
<body ng-app="myApp">
  <h1>Google Trends: JS frameworks</h1>
  <div ng-controller="ChartCtrl">
    <div google-chart chart="chartObject" style="height:400px;
width:100%;"></div>
  </div>
  <script src="bower_components/angular/angular.js"></script>
  <script src="bower_components/angular-google-chart/ng-google-chart.
js"></script>
```

```
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
  </body>
  </html>
```

Take a note of the following:

- The `google-chart` directive embeds any of the Google chart types
- The `chartObject` object is an AngularJS model that consists of columns and rows to be rendered

To leverage the `angular-google-chart` directives, we need to inject its module in our app; so, add the following in `app.js`:

```
'use strict';
angular.module('myApp', ['googlechart', 'myApp.controllers']);
```

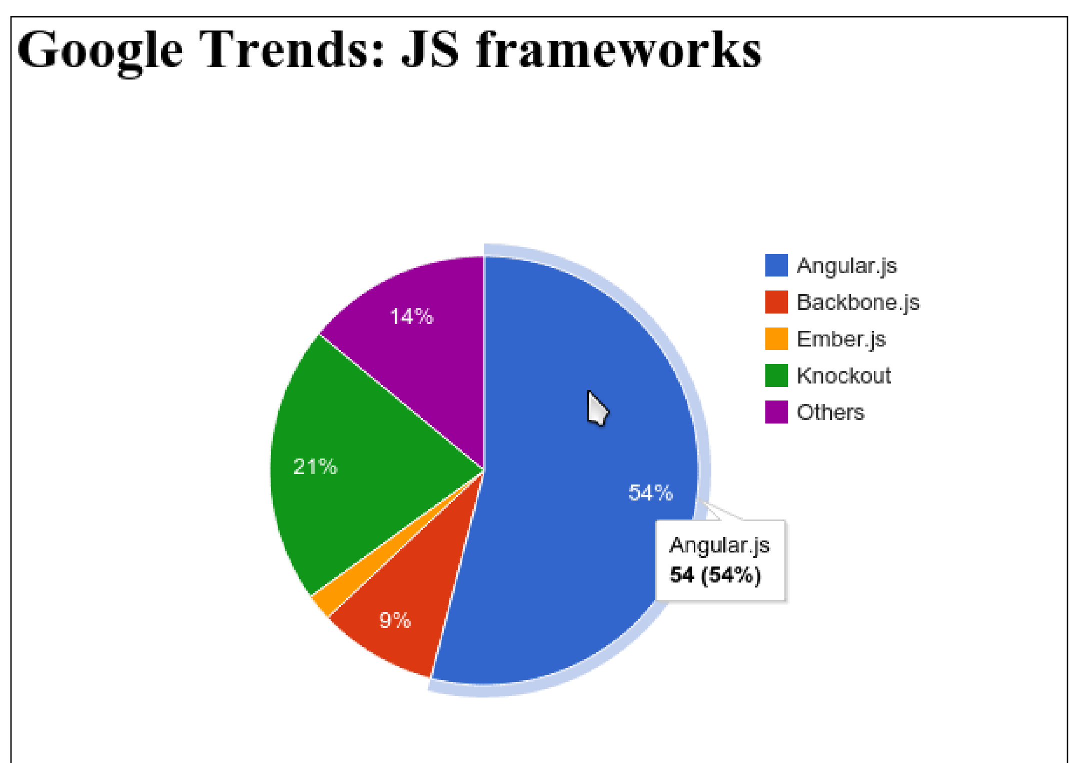
Then, we'll set up our chart's configuration and data to feed to. For this example, we're going to use a real-world Google search data for different JavaScript frameworks for the past 12 months:

```
'use strict';
angular.module('myApp.controllers', []).controller('ChartCtrl',
function ($scope) {
  $scope.chartObject = {
    type: 'PieChart',
    data: {
      "cols": [
        {label: "frameworks", type: "string"},
        {label: "shares", type: "number"}
      ],
      "rows": [
        {c: [{v: "Angular.js" }, {v: 54 }]},
        {c: [{v: "Backbone.js"}, {v: 9  }]},
        {c: [{v: "Ember.js"   }, {v: 2  }]},
        {c: [{v: "Knockout"   }, {v: 21 }]},
        {c: [{v: "Others"     }, {v: 14 }]}
      ]
    }
  };
});
```

Notice that `$scope.chartObject` contains web searches (in percentage) performed on Google for AngularJS, Backbone, Ember, and Knockout MVC JavaScript frameworks. You can check it out at `http://goo.gl/p7kgQn`, if you want to take a look.

We've defined the type of chart and the value field to be read from the data to create sectors. Also, `$scope.chartObject` holds the actual data that consists of the columns, rows, and the title for each sector.

This will render the following beautiful interactive pie chart with these mere lines of JavaScript code:



Try hovering over the pie chart to have each sector pop out to show area covered in percentage in the tooltip.

> You will notice that we have not included the original Google Chart library to render the pie chart. That's because the `angular-google-chart` module itself resolves all the necessary JavaScript and CSS dependencies in the background. Do check out the `<head>` tag of this example in the developer tool to find out.

Building a dashboard using the GitHub REST API

Enough with all the charts. Let's build a cool application using the GitHub REST API. The GitHub API exposes RESTful services in order to fetch details for any software projects, also known as repositories, hosted there. All the data that is stored on the GitHub server can be accessible for developers through these APIs to implement their ideas or build applications atop.

GitHub (`https://github.com`) is a web-based hosting service for software development projects that use the Git revision control system. In addition to it, developers can star/favorite projects they are interested in and collaborate with other developers by reporting bugs or sending patches (also known as pull requests) to improve them.

In this section, we are going to build a simple dashboard to watch over the development of the AngularJS framework and the pace. We'll consume two GitHub services to find the top 10 developers working on the top 100 open issues and the top 100 pull requests merged to get a glance at the developers' productivity during the week.

In this example, we are going to use a different charting library named `nvd3`, which is based on `d3.js`. We'll first install all the dependencies as follows:

```
bower install --save angular
bower install --save underscore
bower install --save d3
bower install --save nvd3
bower install --save angularjs-nvd3-directives
```

D3.js (`http://d3js.org/`) is a JavaScript library used to manipulate HTML based on data with the help of HTML, SVG, and CSS. NVD3 (`http://nvd3.org/`) is a library of reusable chart components for `d3.js`. The angularjs-nvd3-directives project has ported `nvd3.js` charts to AngularJS.

Let's update `index.html` with the following to have a clean slate:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
```

```
    <title>Angular Dashboard: nvd3 charts with external data point</
title>
    <link rel="stylesheet" href="css/main.css">
</head>
<body ng-app="myApp">
  <h1>AngularJS Dashboard</h1>
  <script src="bower_components/underscore/underscore.js"></script>
  <script src="bower_components/angular/angular.js"></script>
  <script src="bower_components/d3/d3.js"></script>
  <script src="bower_components/nvd3/nv.d3.js"></script>
  <script src="bower_components/angularjs-nvd3-directives/dist/
angularjs-nvd3-directives.js"></script>
  <script src="js/app.js"></script>
  <script src="js/controllers.js"></script>
</body>
</html>
```

Then we need to inject nvd3 chart's module in `app.js` as:

```
angular.module('myApp', ['nvd3ChartDirectives', 'myApp.controllers']);
```

We are going to use Twitter Bootstrap, which is a CSS framework, to *themify* our dashboard application in order to save our time from designing but building it. Let's quickly install the same. Run the following command in the terminal:

```
bower install --save bootstrap
```

This will install the latest version of bootstrap. Do not worry about bootstrap now because we're going to cover it in detail in the next chapter. However, we'll have to reference it inside the `index.html` file along with `nv.d3.css`, as follows:

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular Dashboard: nvd3 charts with external data point</
title>
    <link rel="stylesheet" href="css/main.css">
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
    <link rel="stylesheet" href="bower_components/nvd3/nv.d3.css">
</head>
```

Right now our application does not do anything, so let's make it alive by performing GitHub API integration. We are going to fetch the top 100 open issues assigned, group them by contributors, and sort the contributors that have the most issues to get the top 10 developers. Here is how the service looks:

```
var service = "https://api.github.com/repos/angular/angular.js/issues?
state=open&sort=updated&page=1&per_page=100&assignee=*"
```

It's a simple URL that takes a few options, such as:

- `state`: This denotes the status of an issue, that is, open, close, or all.
- `sort`: This sorts all issues by created date, updated date, or total number of comments made on the issue.
- `assignee`: This targets a specific GitHub user. The `*` symbol means any assignee.
- `per-page`: This is the total number of issues the service should return.

The service returns paginated data in a descending order, so you can change the page option to say, `2`, to get a slot of the previous 100 issues.

The service returns lots of information about an issue but I trimmed down the unnecessary part for brevity here:

```
[
  {
    "url": "https://api.github.com/repos/angular/angular.js/
issues/6934",
    "title": "Return type annotation invisible in docs",
    "user": {
      "login": "IgorMinar"
    },
    "state": "open",
    "assignee": {
      "login": "petebacondarwin"
    }
]
```

Let's create an issue controller to encapsulate the issues fetched by the service. We'll first add some necessary markup in `index.html`:

```
<div class="row">
<div class="col-md-6">
<div class="panel panel-default" ng-controller="IssueCtrl">
<div class="panel-heading"><b>Top 100 Open Issues with Assignee</b></
div>
<div class="panel-body">
<div nvd3-pie-chart data="data" width="500" height="500" x="x()"
y="y()" showLabels="true" donut="true" donutLabelsOutside="true"></
div>
</div>
</div>
</div>
</div>
```

Notice that data attributes take the data model to render, whereas the `x` and `y` attributes tell the pie chart to read object properties from the data model to be used on the *x* and *y* axes, respectively. We've added the controller named `IssueCtrl` here and the `nvd3-pie-chart` directive to render the contributors.

Let's define the controller in `controllers.js`, as follows:

```
angular.module('myApp.controllers', [])
.controller('IssueCtrl', function ($scope, $http) {
  $scope.x = function() {
    return function(d) {
      return d.category;
    };
  };

  $scope.y = function() {
    return function(d) {
      return d.value;
    };
  };
})
```

For the moment, we have not passed any data, so you'll not see the pie chart in place. Let's add the following snippet into the same controller in `controllers.js` to fill the pie with remote data:

```
$http.get('https://api.github.com/repos/angular/angular.js/iss
ues?state=open&sort=updated&page=1&per_page=100&assignee=*').
success(function(issues) {
    var assignees = [], data = [];
    issues.forEach(function(issue) {
      assignees.push({category: issue.assignee.login});
    });
    assignees = _.groupBy(assignees, function(assignee) { return
assignee.category; });
    assignees = _.sortBy(assignees, function(assignee) { return
assignee.length; }).reverse();
    assignees = assignees.slice(0, 10);

    assignees.forEach(function(assignee) {
      data.push({category: assignee[0].category, value: assignee.
length});
    });
    $scope.data = data;
  });
```

As soon as we call the API, we'll receive the response in the success callback of the `$http` service. The `$http` service is a wrapper around the native `XHR` object that facilitates communication with the remote HTTP servers. It broadcasts events such as success, error, and then with the data received, from the remote endpoints.

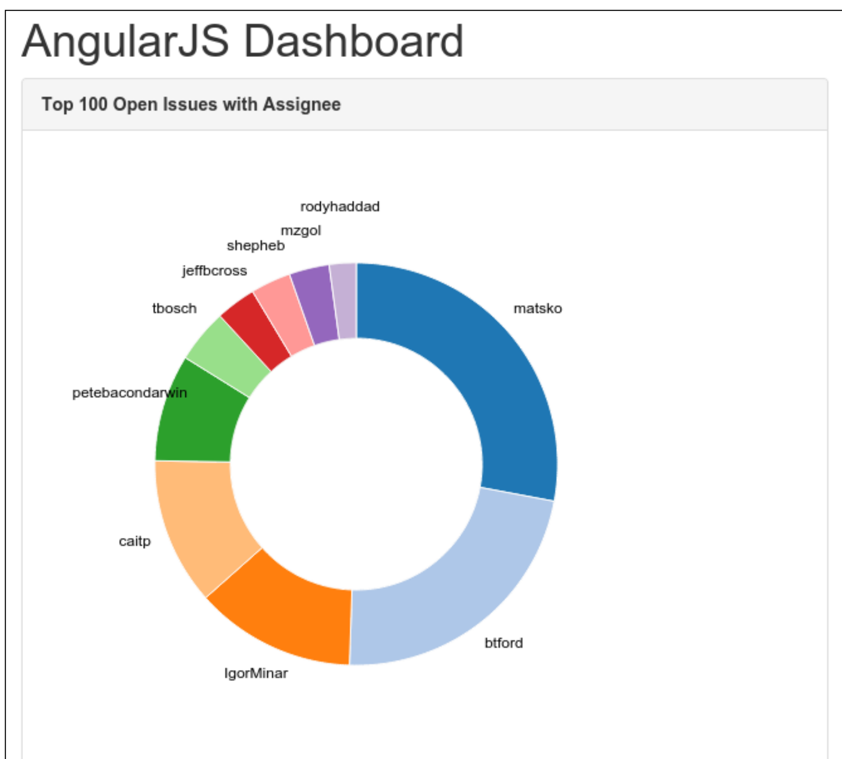
As you saw earlier, the service returns a lot of data but we only need the assignees out of it. So, we first iterate through the issues and extract all assignees as categories out in a collection named assignees.

Then, we group the collection by assignee so that common assignees will be combined together to get us the details about the total number of bugs each one has been taking care of.

Later, we sort the collection by assignees that have the maximum issues in a descending order to get the top 10 assignees by cutting out the remaining.

Once we get our top 10 contributors, we then create one more collection named data to maintain a list of assignees with their respective issues.

Finally, we update the scope `data` model with the data collection that we just created to update the chart, as shown in the following screenshot:



With this, everyday you'll get new updates about the progress of what the AngularJS community is doing.

We have used the donut chart here, which is a variation of the pie chart, and there is a lot of cool stuff you can do with it. Visit `http://cmaurer.github.io/angularjs-nvd3-directives/pie.chart.html` for more details.

Extending the dashboard

Let's extend the dashboard to take a look at how AngularJS contributors are working and when they are most productive during a week. Just update `index.html` first, as follows:

```
<div class="row">
    <div class="col-md-6">
      <div class="panel panel-default" ng-controller="IssueCtrl">
        <div class="panel-heading">
          <b>Top 100 Open Issues with Assignee</b>
        </div>
        <div class="panel-body">
          <div nvd3-pie-chart data="data" width="500"
height="500" x="x()" y="y()" showLabels="true" donut="true"
donutLabelsOutside="true"></div>
        </div>
      </div>
    </div>
    <div class="col-md-6">
      <div class="panel panel-default" ng-controller="PullCtrl">
        <div class="panel-heading">
          <b>Top 100 Pull Requests Closed</b>
          <div class="btn-group">
            <button type="button" class="btn btn-default" ng-
click="page = page + 1;loadData();">&laquo;</button>
            <button type="button" class="btn btn-default" ng-
click="page = page - 1;loadData();" ng-disabled="page == 1">&raquo;</
button>
          </div>
        </div>
        <div class="panel-body">
          <div nvd3-stacked-area-chart data="data" showXAxis="true"
showYAxis="true" width="500" height="500" showLegend="true"></div>
        </div>
      </div>
    </div>
  </div>
```

Here, we've added one more controller named `PullCtrl` to show the top 100 pull requests merged. A pull request is a patch for a bug or a feature request included for the next release of `angular.js`. Notice that we've used the `nvd3-stacked-area-chart` directive to get an area chart for the same and that the collection data is the same. However, the data in `IssueCtrl` will not override this collection of the pull request because of different controllers. Controllers basically create a barricade surrounding the data so that it will not be shared across. Also, notice the navigation buttons to go back to the history to see how many pull requests were closed.

Let's quickly look at a service we're going to use for this chart:

```
Var service2 = "https://api.github.com/repos/angular/angular.js/pulls?
state=closed&sort=updated&page=1&per_page=100&sort=updated"
```

Almost all the parameters passed to the service are the same as we saw earlier. The only service name changes from issues to pulls.

Let's define our `PullCtrl` controller in `controllers.js` and set up the configuration for the area chart:

```
angular.module('myApp.controllers', [])
controller('PullCtrl', function ($scope, $http) {
  $scope.page = 1;
  $scope.loadData();
});
```

Let's go over what is important here:

- The `$scope.page = 1` line of code will return 100 pull requests closed recently and then you can go back and fetch the previous 100 pull requests and so on using the navigation buttons added earlier.
- The `loadData()` method will be called immediately to fetch the first 100 pull requests. It will also be called when a page changes via navigation buttons.

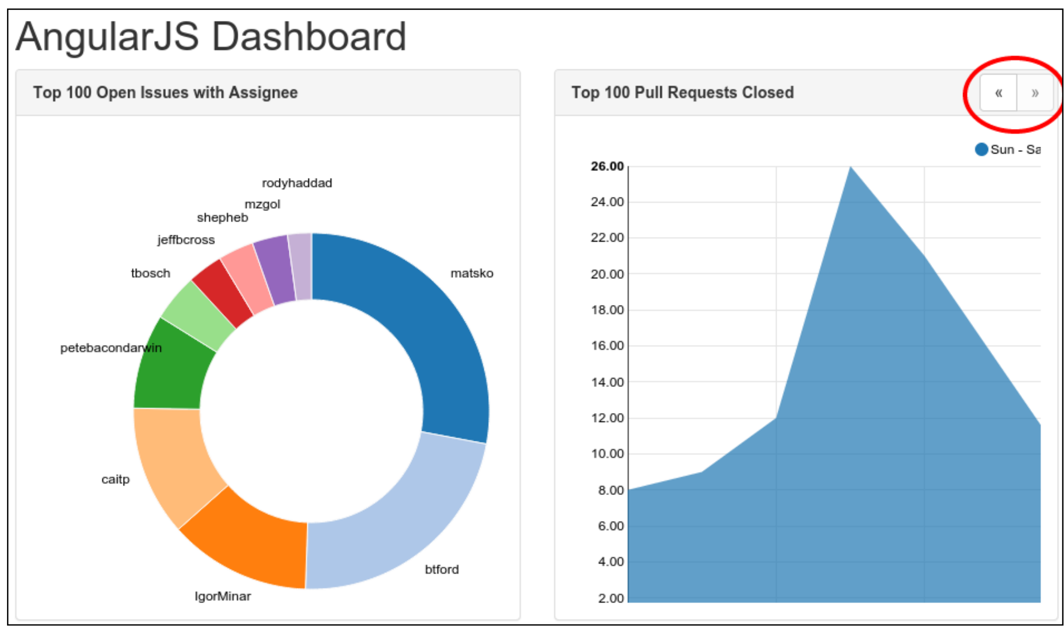
Now we can consume the `service2` defined previously to fetch the data for the chart. Add the following method before the `$scope.loadData()` call in `PullCtrl` because the method should be defined before we call it as:

```
$scope.loadData = function() {
    $http.get('https://api.github.com/repos/angular/angular.js/
pulls?state=closed&sort=updated&page=' + $scope.page + '&per_
page=100&sort=updated').success(function(pulls) {
      var pulley = [], data = [];
      pulls.forEach(function(pull) {
        pulley.push({value: new Date(pull.updated_at).getDay()});
      });
```

```
        pulley = _.groupBy(pulley, function(pull) { return pull.value;
});
        pulley = _.sortBy(pulley, function(pull, i) { return pull[0].
value; });
        pulley.forEach(function(pull, i) {
          data.push([i, pull.length]);
        });
        $scope.data = [{
          "key": "Sun - Sat",
          "values": data
        }];
      });
    };
```

Here, we first store the last updated date in a collection named `pulley`. We then group them by the date to get the number of pull requests merged per day. The week starts with `0`, that is, Sunday. Later, we just sort them in the ascending order. At the end, we make a new collection of data having total number of pull requests merged from `0` (Sunday) to `6` (Saturday).

Here is our minimal dashboard in AngularJS:



Do not forget to click on the navigation buttons to see the graph changing. I'm sure you've learned a lot about charts in this chapter and you will take this dashboard to the next level and create something amazing.

Summary

That was a mouthful! We've learned a lot about different types of charts, writing custom directives, using external widgets, and executing XHR requests. At the beginning of the chapter, we created a simple CSS-based bar chart from scratch and then made it dynamic with AngularJS models. We then converted the chart into a reusable and declarative widget to facilitate code reuse. To avoid reinventing the wheel, we tried the cool framework named Kendo UI and used its data visualization component to create a stunning pie charts. At the end of the chapter, we consumed GitHub APIs to build a simple but useful dashboard application.

In the next chapter, we'll learn how to leverage various CSS frameworks to make AngularJS applications look beautiful without writing CSS on our own.

7

Customizing AngularJS with CSS and CSS Frameworks

After playing with animations for a while in the previous chapter, it's time to find out what it takes to combine the power of CSS frameworks with AngularJS in order to create a responsive web app in a matter of time. In this chapter, we'll get a solid grounding in CSS, which is essential when we are trying to customize the looks of anything in AngularJS.

We'll be covering the following topics in this chapter:

- Solving problems with media queries
- Taking a look at the Twitter Bootstrap CSS framework to quickly mock up well-designed applications
- Creating a simple bookmarking application in Foundation, a mobile-friendly and responsive framework
- Learning how the CSS grid system lays the foundation for any application and what role it plays in designing a flexible layout

The evolution of responsive design

In the early days, HTML was born as a structural markup language to create documents on the Web. At that time, it had various elements that were useful for describing things, such as headings, paragraphs, hyperlinks, bulleted lists, tables for tabular data, and so on. All of a sudden, the power of HTML was obvious to a lot of developers and they started playing with it. While doing so, developers quickly found the missing pieces to write great documents, resulting in a new set of elements to be introduced to focus on emphasizing a bit of text that would improve the presentation in the document. Eventually, the rising demand for presentation increased and a structural language started to become presentational. However, adding new markup elements in every new version of HTML was not possible and reliable. At last, **W3C** (**World Wide Web Consortium**) brought CSS to the world in order to fill the gap.

Since then, CSS has been widely respected and is used more than HTML to bring aesthetic designs on the Web. Developers began using CSS to build everything from simple to lucrative websites and web applications, in order to make them look good and to make it easy to customize them later instead of polluting HTML with presentational markups. Everything was going great, until the mobile web came along. These days, developers have to help their websites/apps to run on mobile devices, and the worst thing is that they come in many shapes/sizes unlike a desktop. Having different versions of the same website/app to support different mobile devices is not only a bad practice, but it can also lead to maintenance hell.

This problem gave birth to **responsive layout**. The responsive design adapts the layout to the viewing environment using fluid- and proportion-based CSS grids, which we are going to explore in later sections. The main difference between fluid and responsive layouts is that fluid layouts are based on proportionally laying out your websites, so elements take up the same percentage of space on different screen sizes; while responsive design uses CSS media queries to present different layouts based on the screen size / type of screen. There is an in-depth article in *Smashing Magazine* on fixed versus fluid versus responsive that is worth checking out; go to `http://www.smashingmagazine.com/2009/06/02/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/`. It's funny if I were to say that an HTML table (having the width in percentage) is the father of responsive layout, but it's very difficult to design a highly responsive design with it, such as Pinterest's card-like interface.

I really liked the book *Mobile First*, *Luke Wroblewski*, *Ingram Publishing*, when I read it for the first time; hence, I will definitely suggest that you give it a try. However, this is not a book with code examples but with a lot of theoretical knowledge and covers the whys more than the hows—which is quite essential as well.

Introducing media queries

W3C created media queries as part of the CSS3 specification to solve the problem. Media queries allow the presentation of content to be tailored to a specific range of output devices without having to change the content itself. A media query consists of a **media type** and at least one **expression** that limits the style sheet's scope by using media features, such as width, height, and color.

@media

The `@media` CSS at-rule lets us control CSS properties to be applicable to a certain media. It allows different style rules for different media/devices. There are plenty of media types, as follows:

- **all**
- **print**
- **screen**

There are many others media types that we're not going to cover here, so feel free to visit `https://developer.mozilla.org` for more information.

The `all` media type is suitable for all devices. The `print` type targets printers, whereas the `screen` type is intended primarily for color computer screens. You can define it as follows:

```
@media <media types> {
  /* media-specific rules */
}
```

Let's quickly create a small demonstration to check whether the screen and print types are working. We'll create `at-media-css.html` as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>@media CSS at-rule</title>
    <style type="text/css">
      @media screen {
        p {
          font-size: 50px;
        }
      }
      @media print {
        p {
          font-size: 10px;
        }
      }
    </style>
</head>
<body>
  <p>AngularJS UI Development</p>
  <blockquote>Try pressing Ctrl+P and see the above text</blockquote>
</body>
</html>
```

It does a simple thing, that is, the `fontSize` object of the `<p>` tag changes to `10px` while printing. You can press *CTRL* + *P* in your browser to see it in action.

The @media expression

Along with the media type, you can also target one or more expressions as media features, which resolve to either `true` or `false`. If it turns to `true`, then the styles defined within a block will be applied to the page/elements.

In the following example, we are targeting all devices with a maximum width of `600px`. If they exceed the width, then the text color will change to green, as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>@media CSS features</title>
    <style type="text/css">
      p {
        font-size: 50px;
        color: red;
      }
      @media all and (max-width: 600px) {
        p {
          color: green;
        }
      }
    </style>
</head>
<body>
  <p>AngularJS UI Development</p>
  <blockquote>Try resizing the Window</blockquote>
</body>
</html>
```

In addition to this, you can even use the CSS media query on a link element by moving all the CSS styles into the `style.css` file and you can reference it in `index.html`, as follows:

```
<link media="all and (max-width: 600px)" href="style.css"
rel="stylesheet" />
```

In the mobile era, we often need to customize the application's look and feel based on orientation. Because of limited space in portrait mode (than in landscape), we might want to hide a few features of an application or group them in a drop-down menu. To do this, you can use the orientation feature, as follows:

```
@media all and (orientation: landscape) {
    button.double-page-mode { display: block; }
}
@media all and (orientation: portrait) {
    button.double-page-mode { display: none; }
}
```

With the previous styles in `index.html`, you can figure out the orientation in devices with the JavaScript API for the same. The following snippet returns `true` if the device is in the portrait mode, and you are free to perform any sort of operations or checks to adjust the behavior of the application:

```
<script type="text/javascript">
    if (window.matchMedia("(orientation: portrait)").matches) {
      document.getElementsByTagName('body')[0].style.display = 'none';
      alert('The Portrait mode is not supported.');
    } else {
      document.getElementsByTagName('body')[0].style.display =
'block';
    }
</script>
```

If you tilt a device or resize a browser window and reload the page again, you will get an alert, suggesting that you use landscape mode.

Media queries come in many flavors to suit your needs. Visit the media queries section on `https://developer.mozilla.org` for more supported media features.

Better designs with Twitter Bootstrap

Designing for applications is a really exciting job, but sometimes browser quirks might drive you crazy and it's daunting to keep track of ways to work around cross-browser issues. Before any CSS framework, various libraries were used to design a UI, which led to inconsistencies and a high maintenance burden. Twitter Bootstrap (`http://getbootstrap.com/`) was developed to solve these kind of problems that developers face in their day-to-day lives and to encourage consistency at a higher level. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation, and other UI components, as well as optional JavaScript extensions. In short, if you have a quick idea for any sort of application and don't want to waste your precious time in designing it, then you can build a prototype of your application with Twitter Bootstrap in no time. There is an **Expo** section on their website that showcases various websites from around the world built using Twitter Bootstrap, which will literally blow your mind. Do check it out!

Moving forward, in this section, we'll revamp the to-do application that we built in *Chapter 5*, *Learning Animation*. We'll only modify the HTML and update the CSS without touching the JS part of the application. Here is the `index.html` file we used earlier:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Angular Animation</title>
    <link rel="stylesheet" href="bower_components/animate.css/animate.
css">
    <link rel="stylesheet/less"  type="text/css" href="css/main.less">
    <script src="bower_components/less/dist/less-1.5.1.js"></script>
    <script src="bower_components/modernizr/modernizr.js"></script>
</head>
<body ng-app="myApp">
    <div ng-controller="NgAnimateCtrl">
      <input ng-model="inputText" />
      <button ng-click="addItem()">add</button>
      <div class="item" ng-repeat="item in items | active">
        <input type="checkbox" ng-model="item.completed" /> 
        <span ng-bind="item.text"></span>
      </div>
    </div>
    <script src="bower_components/jquery/dist/jquery.js"></script>
    <script src="bower_components/angular/angular.js"></script>
    <script src="bower_components/angular-animate/angular-animate.
js"></script>
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
    <script src="js/filters.js"></script>
</body>
</html>
```

So, let's first install Bootstrap by running the following command in the terminal:

```
bower install --save bootstrap
```

Then, reference `bootstrap.css` in the `<head>` tag of `index.html`:

```
<link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
```

Now, replace the highlighted markup in `index.html` with the following:

```
<div class="cover-container">
   <div class="masthead clearfix">
     <div class="inner text-center">
       <h3 class="masthead-brand">TodoStrap</h3>
     </div>
   </div>
   <div class="inner cover">
     // here goes our Todo App
   </div>
</div>
```

In this markup, the `div.cover-container` element is a wrapper for our application in order to make it center-aligned horizontally, which will contain the following styles. Update the `main.less` file, as follows:

```
.cover-container {
  margin-right: auto;
  margin-left: auto;
}
```

The `div.masthead` element denotes the name of the application that uses Bootstrap's `text-align` CSS class to center align the text. It contains:

```
.inner {
  padding: 30px;
}
.masthead-brand {
  margin-top: 10px;
  margin-bottom: 10px;
}
```

Then, add a slight padding around a list of to-dos:

```
.cover {
  padding: 0 20px;
}
```

Apply a background color to the window to make the application stand out:

```
html,
body {
  height: 100%;
  background-color: #333;
```

```
    color: #fff;
    text-shadow: 0 1px 3px rgba(0,0,0,.5);
    box-shadow: inset 0 0 100px rgba(0,0,0,.5);
    overflow: hidden;
  }
```

Finally, adjust the view for portrait and landscape modes in devices in such a way that the whole application will be center-aligned in landscape and will occupy the available space in portrait. So, update `main.less` as follows:

```
@media all and (orientation: portrait) {
  .masthead,
  .cover-container {
    width: 100%;
  }
}
@media all and (orientation: landscape) {
  .masthead,
  .cover-container {
    width: 500px;
  }
}
```

Now, we'll update the empty `div.inner.cover` element with the actual to-do form and a list, as follows:

```
<div class="inner cover">
      <div ng-controller="NgAnimateCtrl">
        <form role="form">
          <input ng-model="inputText" type="text" class="input-lg
form-control" placeholder="What do you wanna do today?" autofocus>
        </form>
        <div class="list-group" style="padding: 0px;margin-top:
10px;">
          <a href="javascript:void(0);" class="list-group-item item"
ng-repeat="item in items | active">
            <span ng-bind="item.text"></span>
          </a>
        </div>
      </div>
    </div>
```

Nothing has changed except that extra `div` tags are added, but there are still a few things to make a note of:

- We've applied two new `bootstrap` CSS classes on the input as `.input-lg` and `.form-control`. The `.input-lg` class is a shortcut for a large input box that makes it big and attractive, whereas `.form-control` lets you enlarge the input to occupy the available space.
- The purpose of `.list-group` is to render complex and customized content in a list, and `.list-group-item` carries an item from the list.

You must be thinking how on earth you'll add a to-do without the add button. Actually, the add button was making the design ugly, so we removed it. Let's track the `keypress` event to add a to-do item when you press the *Enter* key. Replace the input with:

```
<input ng-model="inputText" type="text" class="input-lg form-
control" placeholder="What do you wanna do today?" autofocus ng-
keyup="addItem($event)">
```

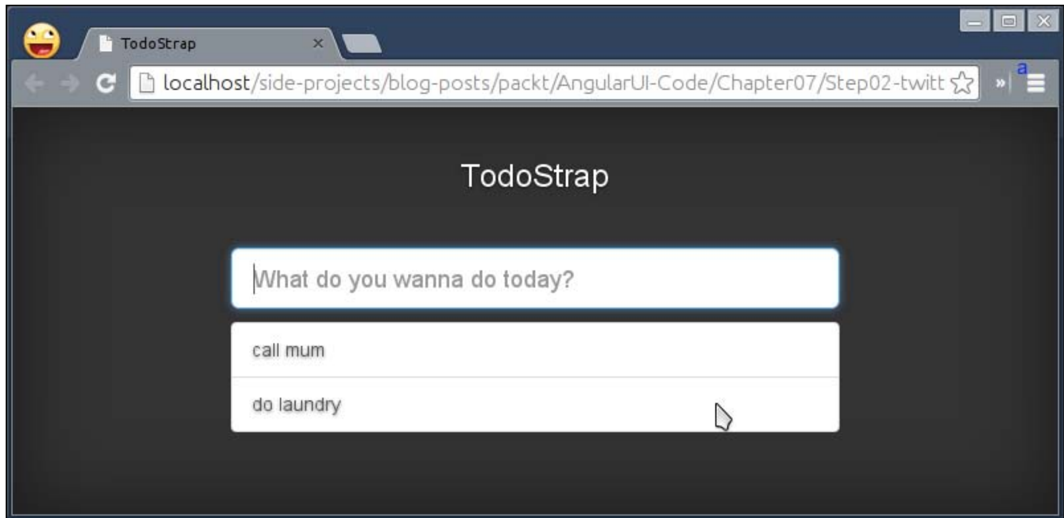
So, on every keyup, the `addItem()` method will be called; this needs to be prevented and that's why we are passing `$event` as a parameter to find out whether the enter key has been pressed or not in order to update the to-do list accordingly.
Let's update the `addItem()` method in `controllers.js`, as follows:

```
$scope.addItem = function ($event) {
      if ($event.keyCode !== 13) return;
      $scope.items.push({
        text: $scope.inputText,
        completed: false
      });
      $scope.inputText = ""
};
```

If you've noticed, we also removed the checkbox next to each to-do item, which is used to clear it, as it's cluttering up the new design. So, now you can just click on the to-do item to remove it from the list; quite simple! Update the list, as follows:

```
<div class="list-group" style="padding: 0px;margin-top: 10px;">
          <a href="javascript:void(0);" class="list-group-item item"
ng-repeat="item in items | active" ng-click="item.completed = !item.
completed">
            <span ng-bind="item.text"></span>
          </a>
 </div>
```

That's it! If all went well, then you will see a beautiful to-do application running on the desktop browser, as follows:



If you look at the same application on iPhone 5, then you will be amazed to see it working there as well:

I believe that if you add a bunch of important features, such as syncing across devices/browsers, e-mails/SMS notifications, and alerts, people would even be happy to pay for the same. Go build something great!

The foundation of your application

Foundation is a frontend framework similar to Twitter Bootstrap, consisting of many useful tools to make responsive, mobile-first websites and web applications. However, it is not recommended that you use it along with Bootstrap as it may affect the layout of the application. Both the frameworks are on a par, so it's quite difficult to judge any of them; however, I would suggest that you try out both to see for yourself. There is a nice article on the comparison by treehouse to get you going, at `http://blog.teamtreehouse.com/use-bootstrap-or-foundation`.

In this section, we are going to design a bookmarking application named PinIt in pure Foundation. Let's begin with the installation first. Run the following command in the terminal:

```
bower install --save foundation
```

Create a fresh copy of `index.html` and reference `foundation.css` in the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Pin It</title>
    <link rel="stylesheet" href="bower_components/foundation/css/
foundation.css">
</head>
<body ng-app="myApp">

</body>
</html>
```

We'll start off by creating a black-colored bar to display the application's name and a few options in the top-right corner, as follows:

```
<nav class="top-bar">
    <ul class="title-area">
      <li class="name"><h1><a href="#">Pin It - <i>The Universal
Bookmarking App</i></a></h1></li>
      <li class="toggle-topbar menu-icon"><a href="#">Menu</a></li>
    </ul>

    /* menu comes here */
  </nav>
```

The things to note here are given as follows:

- The `.top-bar` class creates a fixed position header with black background color and a height of `45px`
- The `.menu-icon` class places a tiny hamburger icon and the `.toggle-topbar` class makes sure that it is hidden until a display is small enough, that is, in devices

Then, we'll add a bookmark **Search** box and **Add** button to store new bookmarks. So, replace the comment with the following markup:

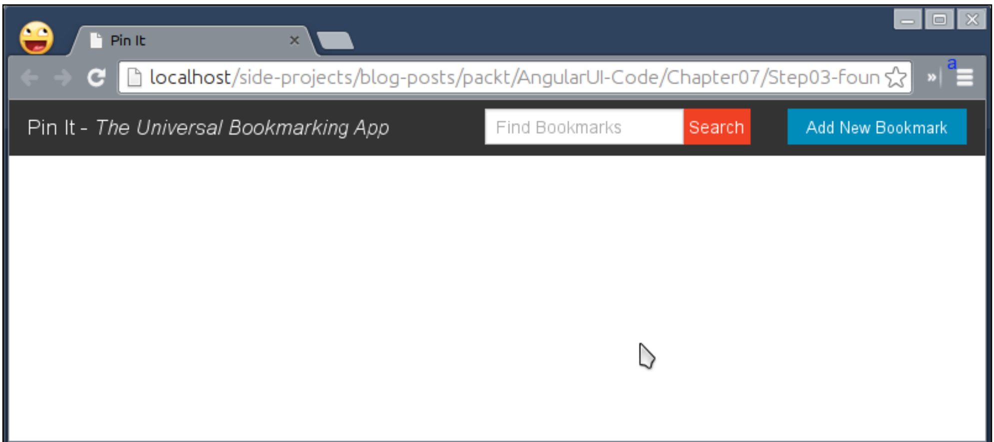
```
<section class="top-bar-section">
   <ul class="right">
     <li class="has-form">
       <div class="row collapse">
         <div class="small-9 columns">
           <input type="text" placeholder="Find Bookmarks">
         </div>
         <div class="small-3 columns">
           <a href="#" class="alert button expand">Search</a>
         </div>
       </div>
     </li>
     <li class="has-form">
       <a class="button" data-reveal-id="addBookmark">Add New
Bookmark</a>
     </li>
   </ul>
 </section>
```

The important bits to note are as follows:

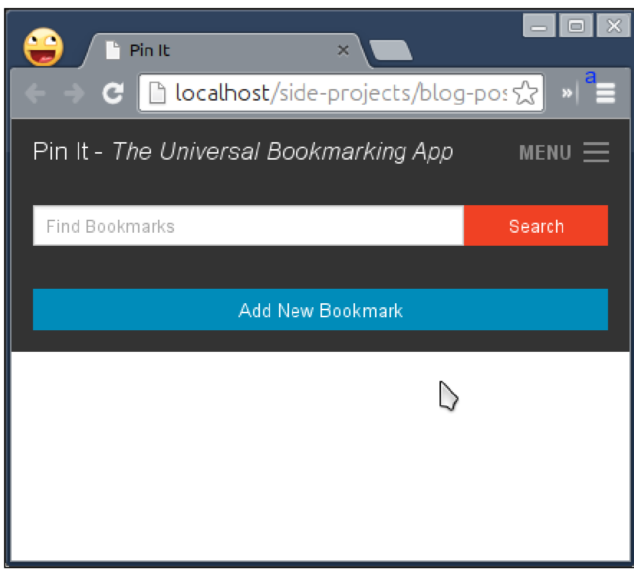
- The framework comes with really useful classes, such as positioning an element to the left or right; just add `.right` to move the **Search** box and the **Add** button to the right.
- The `.has-form` class adds padding between form elements to perfectly space them out. Use it wherever required.
- Like Twitter Bootstrap, Foundation also comes pre-equipped with a built-in CSS grid system that allows us to create tables using the `div` elements. Each grid row has 12 columns. You can add `.small-1` to `.small-12` or `.large-1` to the `.large-12` class in order to split the markup in the columns. These classes only have the width set, so you have to use `.columns` with it to place content side by side.

- The **Search** button has to have the `.button` class, and the `.alert` class is optional to color it in red. Additionally, you can use the `.secondary` and `.success` classes, replacing `.alert` to change the look of the button. The `.expand` class just lets the button occupy the remaining space, if available.
- The **Add** button has the `data-reveal-id` attribute that makes it workable. It points to a modal to open, if clicked.

This minimal markup will create a nice top bar in no time, as follows:



Moreover, it's even responsive. On a small display, the **Search** box and the **Add** button will be hidden under the menu icon, which will be revealed when clicked:



To extend it, we'll now add markups for the **Add New Bookmark** pop up:

```
<div id="addBookmark" class="reveal-modal medium" data-reveal>
    <h2>New Bookmark</h2>
    <form>
      <div class="row">
        <div class="large-12 columns">
          <label class='error'>URL
            <input type="url" name="url" placeholder="Add bookmark
URL" required />
          </label>
          <small class="error">Please enter valid URL - should begin
with http or https</small>
        </div>
      </div>
      <div class="row">
        <div class="large-12 columns">
          <label class='error'>Title
            <input type="text" name="title" placeholder="Add bookmark
Title" required />
          </label>
          <small class="error">Please enter bookmark Title</small>
        </div>
      </div>
      <div class="row">
        <div class="large-12 columns">
          <label class='error'>Tags
            <input type="text" name="tags" placeholder="comma
separated tags i.e. angular,jquery,foundation" required />
          </label>
          <small class="error">Please use comma for more tags</small>
        </div>
      </div>
      <button type='submit' class="button expand">Save Bookmark</
button>
    </form>
    <a class="close-reveal-modal">&#215;</a>
  </div>
```

Now, you are pretty familiar with the markups and CSS classes used in Foundation. We'll dabble into the modal size, which is configured by the `.medium` class that sets the width to 60 percent of the window width. Alternatively, you can use the `.tiny`, `.small`, `.large`, and `.xlarge` classes.

Unlike the top bar, here we're using large grid classes with `.large-12`. The `.error` class on the label will highlight the form labels in red—useful during form validation—and `small.error` points to actual error messages.

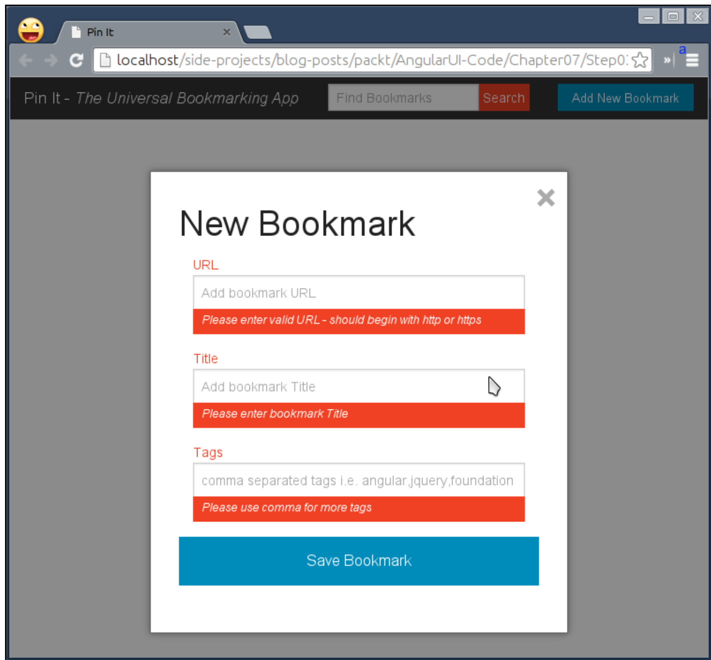
By the end, we have a close icon with `.close-reveal-modal`, which simply hides the modal. To make the modal work, we need to rely on `foundation.js`. So let's add it in `index.html`:

```
<script src="bower_components/jquery/dist/jquery.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/foundation/js/foundation.js"></script>
<script src="js/app.js"></script>
```

Please note that Foundation JavaScript components depend on jQuery. Now, update `app.js` as follows:

```
angular.module('myApp', []).run(function($document) {
  $document.foundation();
});
```

The `$document.foundation()` call activates custom attributes that we used earlier in the top bar, such as `data-reveal-id`, `top-bar`, and so on. With this call, you will see a modal after clicking on the **Add** button, as follows:



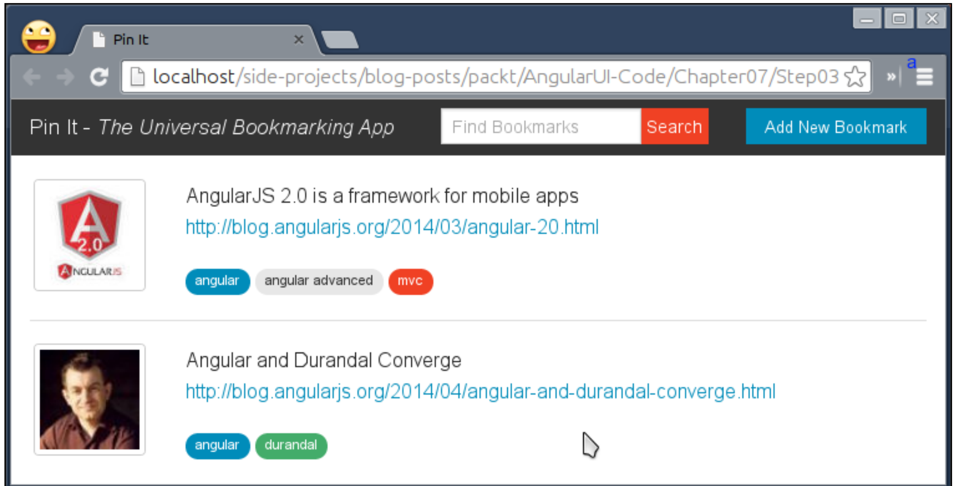
Now, we'll list all the bookmarks. Update `index.html` as follows:

```
<div class="row" style="padding-top: 20px;">
    <div class="large-12 columns">
      <div class="row">
        <div class="columns small-2 text-center"><a class="th
radius"><img src="img/angular20.jpg" /></a></div>
        <div class="columns small-10">
          <p>AngularJS 2.0 is a framework for mobile apps <br><a
href="http://blog.angularjs.org/2014/03/angular-20.html" target='_
blank'>http://blog.angularjs.org/2014/03/angular-20.html</a></p>
          <span class="label round">angular</span>
          <span class="label secondary round">angular advanced</span>
          <span class="label alert round">mvc</span>
        </div>
      </div>
  <hr/>
  </div>
```

Here, we want a two-column layout. In the first column, we are showing a thumbnail related to the bookmarked link, and in the second column, we are showing the bookmarked title, URL, and tags to make them searchable. That's why we've used the columns `.small-2` and `.small-10` for thumbnail and bookmark details, respectively.

Similar to the alerts we saw in the modal, Foundation also has support for labels with different colors, that is, by default, `.secondary`, `.success`, and `.alert`. You can also make them rounded with the `.round` class.

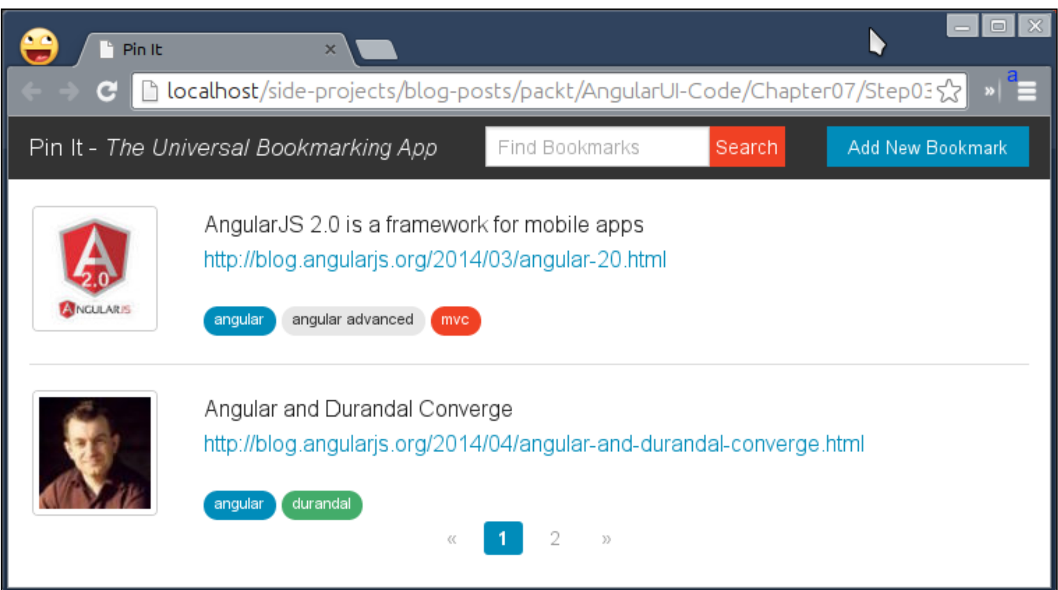
You can repeat the same row after `<hr/>` to show more bookmarks. This is what you'll see in the browser now:



Let's quickly add pagination at the bottom of the bookmarks list in `index.html`, as follows:

```
<div class="pagination-centered">
  <ul class="pagination">
    <li class="arrow unavailable"><a href="">&laquo;</a></li>
    <li class="current"><a href="">1</a></li>
    <li><a href="">2</a></li>
    <li class="arrow"><a href="">&raquo;</a></li>
  </ul>
</div>
```

The pagination uses a `utility` CSS class to center-align the pagination with `.pagination-centered`. Use the `.unavailable` class to disable the previous or next arrow and the `.current` class to activate the page you are on.
Our application design is now ready:



Currently, the application is not usable, but we'll make that happen in *Chapter 10, Mobile Development Using AngularJS and Bootstrap* and also covert the same into Twitter Bootstrap.

Although we have just scratched the surface here, it should not stop you from learning further to build your next ambitious application. Here are couple of books on the Foundation framework (`http://www.packtpub.com/zurb-foundation-blueprints/book`) and Twitter Bootstrap (`http://www.packtpub.com/twitter-bootstrap-web-development/book`) from the same publisher to get you going.

Summary

In this chapter, we got a head start with two of the most popular and capable responsive CSS frameworks. We went through the history of responsive layout and learned how CSS frameworks are born out of the necessity to simplify developers' lives. We learned the usage of media queries to design a responsive layout from scratch. We then dived into an extremely powerful CSS framework named Twitter Bootstrap to incarnate our to-do application to be the stunning TodoStrap. Finally, we designed a full-fledged bookmarking application in an equally compelling and mobile-first, Foundation CSS framework in a short span of time.

In the next chapter, we'll cover the implementation of Twitter Bootstrap JavaScript components in the form of AngularUI Bootstrap.

8

AngularUI Bootstrap Development

In this chapter, you'll learn why the **AngularUI Bootstrap** project has been initiated, what problem it solves, and how to use its components' offerings to build modern web apps.

We'll be covering the following topics in this chapter:

- Using the **accordion** widget to show vast details without information overload
- Creating a Tabbed Interface declaratively using the **tab** component
- Converting traditional radio/checkbox buttons into an intuitive **button** set
- Notifying users with custom messages and progress using **alerts** and **progressbar**
- Implementing the quick search functionality using **typeahead**
- Understanding when to use the AngularUI **collapse**, **calendar**, **ratings**, **carousel**, and **dropdown** plugins

Why use AngularUI Bootstrap?

We saw in the previous chapter how powerful the Twitter Bootstrap CSS framework is. In addition to the CSS helper classes, it also offers a set of useful and extensible JavaScript widgets that we can use to create world-class web apps. However, when it comes to AngularJS, you have to use it declaratively, so you have to add an extra layer on top of the existing libraries by writing custom AngularJS directives. This not only compounds to extra overload for your application (in terms of KBs) but also makes it complex to manage. In fact, rewriting it from scratch solely in AngularJS is ideal to obviate the need for external libraries. This is how the AngularUI project came into existence, which consists of many subprojects that deliver native implementation in pure AngularJS. AngularUI Bootstrap is one of them, which focuses on migrating the Twitter Bootstrap JavaScript widgets to native, lightweight directives, without any dependency on jQuery or Bootstrap's JavaScript. The reason why it is better to rewrite an existing JavaScript or jQuery code into new AngularJS directive is that the resulting directive is smaller in size compared to the original code and better integrated into the AngularJS ecosystem.

All the directives have their own AngularJS module without any dependencies on other modules or third-party JavaScript code, which allows us to easily use a subset of it (as per our requirements) without loading the entire offering.

Building a Project Management Application

In this chapter, we will build a simple yet useful **Project Management Application** (**PMA**), such as Redmine, Pivotal Tracker, and so on, to keep track of software projects and maintain team progress. We'll call it **ShipIt**. Let's begin.

We'll start off with a bare minimal template and extend it to have various AngularUI Bootstrap components to make up the final usable Project Management Application. First of all, clone the template from the `Step01` folder from *Chapter 5*, *Learning Animation*. Let's replace `index.html` with the following markup:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Ship It</title>
    <link rel="stylesheet" href="css/main.css">
</head>
```

```
<body ng-app="myApp">
  <div class="navbar navbar-inverse navbar-fixed-top"
role="navigation">
    <div class="container">
      <div class="navbar-header">
        <a class="navbar-brand" href="#">Ship It - Universal Project
Management Application</a>
      </div>
    </div>
  </div>

  <div class="container-fluid">
    <div class="row">
      <div class="col-xs-4 col-sm-4 col-md-3 sidebar">
        <!-- Navigation Area -->
      </div>
      <div class="col-xs-8 col-xs-offset-4 col-sm-8 col-sm-offset-4
col-md-9 col-md-offset-3 main">
       <!-- Main Area -->
      </div>
    </div>
  </div>
<script src="bower_components/angular/angular.js"></script>
<script src="js/app.js"></script>
</body>
</html>
```

As you can see, we created a header for a **Single Page Application** highlighted as navigation and then designed a two-column layout to showcase handy options on the left and main area to display information when any option is selected. For now, you will not see any such layout as we've not included Twitter Bootstrap CSS yet, so let's quickly install it. Run the following command in the terminal:

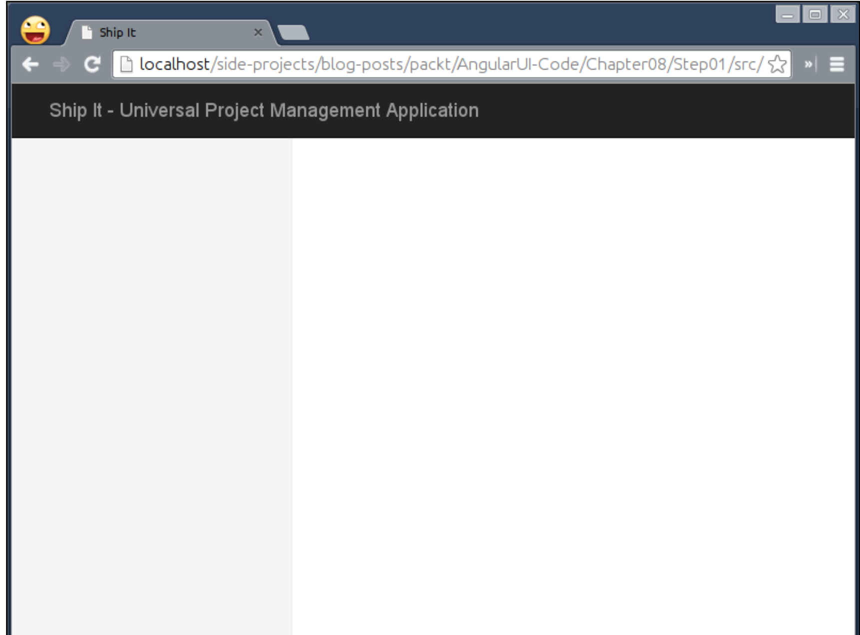
```
bower install --save bootstrap
bower install --save jquery
```

Now, reference `bootstrap.css` in `index.html` inside the `<head>` tag before `main.css` is included and load jQuery before `angular.js`. Then, add the following CSS styles in `main.css` as follows:

```
body {
  padding-top: 60px;
}
```

```
.sidebar {
  position: fixed;
  top: 51px;
  bottom: 0;
  left: 0;
  z-index: 1000;
  display: block;
  padding: 5px 0 0 0;
  overflow-x: hidden;
  overflow-y: auto;
  background-color: #f5f5f5;
  border-right: 1px solid #eee;
}
.tab-content {
  padding: 10px;
  border: 1px solid #ddd;
  border-top: 0px solid transparent;
}
```

Here is an inert but nice two-column layout to get you excited:



Creating accordion

In a PMA, the stakeholders (such as developers, designers, project managers, and so on) often require easy navigation to various projects they are managing to quickly get a good grasp of the overall progress. The accordion UI (`http://en.wikipedia.org/wiki/Accordion_(GUI)`) fulfills this purpose by showing shortcuts to details without an information overload in the first place. First, we'll install `angular-bootstrap` on the command line as follows:

```
bower install --save angular-bootstrap
```

Do not forget to reference this after `angular.js` in `index.html` as follows:

```
<script src="bower_components/angular-bootstrap/ui-bootstrap-tpls.
js"></script>
```

Note that `ui-bootstrap-tpls.js` contains all the necessary templates used by various AngularUI components cached using the `$templateCache` AngularJS built-in service so that you do not have to worry about loading them via AJAX. Feel free to go ahead and take a look at it if you wish.

There are two variants of `ui-bootstrap`—one that comes with inline templates, that is, `ui-bootstrap-tpls.js` (required by the widgets) and another without, that is, `ui-bootstrap.js`. If you include `ui-bootstrap.js`, you will need to provide your own HTML template with all the bindings and directives. These templates allow us to use external templates to redesign any widget using the `$templateCache` service, which you are going to learn in the next chapter.

Now, we'll update `app.js` to make all the AngularUI bootstrap's custom directives available for us:

```
'use strict';
angular.module('myApp', ['ui.bootstrap']);
```

As everything is set up, let's add an accordion component to showcase a list of top projects as it is relevant information everybody needs from a typical PMA. Replace the `<!-- Navigation Area -->` comment block in `index.html` with the following markup:

```
<accordion close-others="false">
  <accordion-group ng-init="project.open = true" is-open="project.
open">
      <accordion-heading>
          Projects
          <i class="pull-right glyphicon" ng-class="{'glyphicon-
chevron-down': project.open, 'glyphicon-chevron-right': !project.
open}"></i>
      </accordion-heading>
      <div class="media">
        <a class="pull-left" href="#">
          <img class="media-object" src="http://placehold.it/32x32">
        </a>
        <div class="media-body">
          <h4 class="media-heading"><a href="#/project/1">Project 1</
a></h4>
        </div>
      </div>
      <div class="media">
        <a class="pull-left" href="#">
          <img class="media-object" src="http://placehold.it/32x32">
        </a>
        <div class="media-body">
          <h4 class="media-heading"><a href="#/project/2">Project 2</
a></h4>
        </div>
      </div>
  </accordion-group>
</accordion>
```

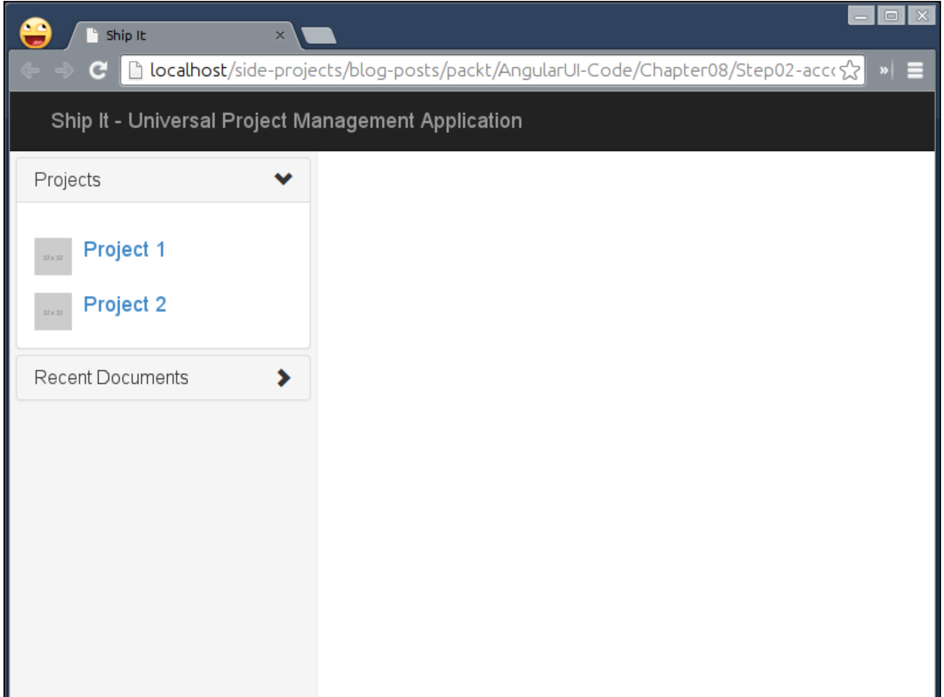
Things to note here are:

- The accordion is a custom AngularJS directive that comprises multiple `accordion-group` elements. Each `accordion-group` element denotes a different accordion panel, and you can add as many you want. Currently, we are just showing the **Projects** panel that should list all the top projects running.
- You can customize the accordion with the `close-others` attribute to restrict it to keep only a single panel open at a time.
- The `is-open` option on `accordion-group` takes a data model that turns to `true` when the panel is open and `false` otherwise. Although it's not necessary, we are using it just for the sake of an arrow icon that is toggled using `ngClass` to make it appealing. So, you will a see a nice down arrow if the panel is closed and an up arrow if it is open.
- The `ng-init` directive is a built-in AngularJS directive to initialize a data model within the template itself. As you can see, we have set `project.open` to `true` to keep the projects panel open by default.
- At the end, `div.media` contains a list of projects that we want to show inside the panel.

Let's add one more panel named **Recent Documents** so that you can actually see collapsible panels in action if `close-others` is set to `true`. Add the following snippet after `accordion-group` ends:

```
<accordion-group is-open="recent.open">
    <accordion-heading>
        Recent Documents
        <i class="pull-right glyphicon" ng-class="{'glyphicon-chevron-
down': recent.open, 'glyphicon-chevron-right': !recent.open}"></i>
    </accordion-heading>
    <ol class="list-unstyled">
      <li><a href="#/document/1">Document 1</a></li>
      <li><a href="#/document/2">Document 2</a></li>
      <li><a href="#/document/3">Document 3</a></li>
      <li><a href="#/document/4">Document 4</a></li>
      <li><a href="#/document/5">Document 5</a></li>
    </ol>
</accordion-group>
```

The only difference here is that we are relying on a different model named `recent.open` to toggle the arrow icon again. Make sure you use a different model for each accordion to avoid any abnormal behavior. Here is how it looks; also, notice the arrow icons for both open and close panels:



Creating tabs

As the left panel started looking nice and clean, we'll show general details and open issues for all the projects in the right column known as tabs. Add the following markup replacing the `<!-- Main Area -->` comment in `index.html`:

```
<tabset>
  <tab heading="Projects Overview">Project Details not Found.</tab>
  <tab heading="Issues">Issues not Found.</tab>
</tabset>
```

Similar to accordion, the `tabset` element comprises all the tabs, which in turn have a tab heading and tab content inside to show when the tab is active. For now, both tabs do not show relevant information, so let's stuff them with some useful information. So, replace the `Project details not found` message with the following markup:

```
<div class="row">
  <div class="col-xs-6">
    <div class="panel panel-default">
      <div class="panel-heading">
        <span class="glyphicon glyphicon-bullhorn"></span>
        <b> Issue Tracking</b>
      </div>
      <ul class="list-group">
        <li class="list-group-item"><span class="label label-
danger">Defect</span> 1168 open / 6313</li>
        <li class="list-group-item"><span class="label label-
success">Feature</span> 2729 open / 5089</li>
        <li class="list-group-item"><span class="label label-
warning">Patch</span> 409 open / 1848</li>
      </ul>
    </div>
  </div>
  <div class="col-xs-6">
    <div class="panel panel-default">
      <div class="panel-heading">
        <span class="glyphicon glyphicon-user"></span>
        <b> Top Developers</b>
      </div>
      <table class="table">
        <tr>
          <td class="text-center"><a href="#/user/1">Sai Padamati</
a></td>
          <td class="text-center"><a href="#/user/2">Akshay
Ravindranath</a></td>
        </tr>
        <tr>
          <td class="text-center"><a href="#/user/3">Dipti Kadge</a></
td>
          <td class="text-center"><a href="#/user/4">Santosh Badge</
a></td>
```

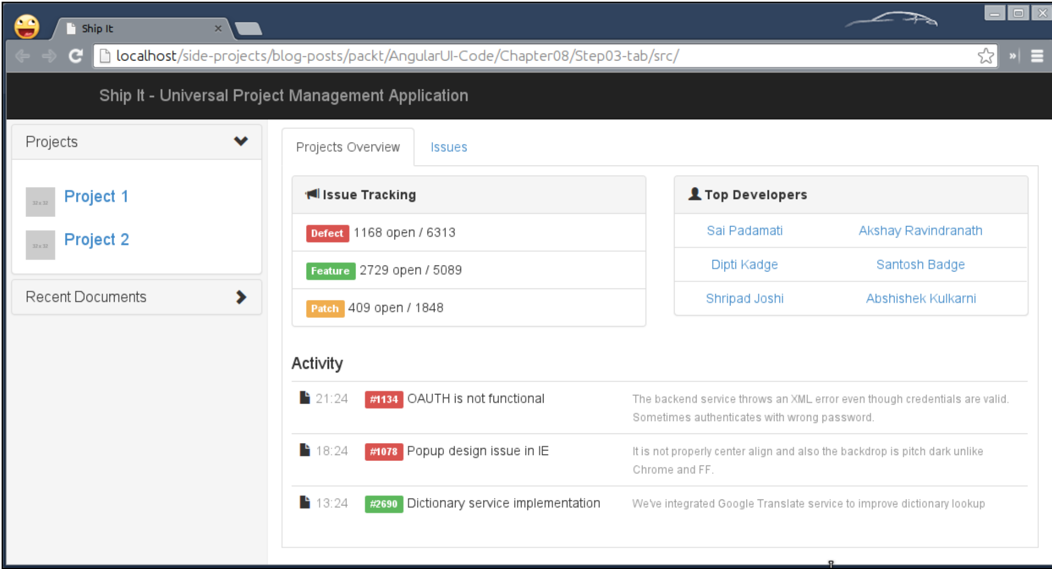
```
        </tr>
        <tr>
          <td class="text-center"><a href="#/user/5">Shripad Joshi</
a></td>
          <td class="text-center"><a href="#/user/6">Abshishek
Kulkarni</a></td>
        </tr>
      </table>
    </div>
  </div>
</div>
```

Here, we have just split the available space into two columns, one for issue tracking and another to list the top six productive developers, by simply using the Twitter Bootstrap table layout, which you learned in the previous chapter, to render tabular data intuitively. Then, we'll show a list of activities going on such as issue fixed, new feature introduced, merged a patch, and so on. Let's add the following markup:

```
<div class="row">
  <div class="col-xs-12">
    <h4>Activity</h4>
    <table class="table">
      <tr>
        <td class="col-sm-1"><span class="glyphicon glyphicon-file"></
span><span class="text-muted">21:24</span></td>
        <td class="col-sm-4"><span class='label label-danger'>#1134</
span> OAUTH is not functional</td>
        <td class="col-sm-6"><small class='text-muted'>The backend
service throws an XML error even though credentials are valid.
Sometimes authenticates with wrong password.</small></td>
      </tr>
      <tr>
        <td class="col-sm-1"><span class="glyphicon glyphicon-file"></
span><span class="text-muted">18:24</span></td>
        <td class="col-sm-4"><span class='label label-danger'>#1078</
span> Popup design issue in IE</td>
        <td class="col-sm-6"><small class='text-muted'>It is not
properly center align and also the backdrop is pitch dark unlike
Chrome and FF.</small></td>
      </tr>
      <tr>
        <td class="col-sm-1"><span class="glyphicon glyphicon-file"></
span><span class="text-muted">13:24</span></td>
        <td class="col-sm-4"><span class='label label-success'>#2690</
span> Dictionary service implementation</td>
```

```
        <td class="col-sm-6"><small class='text-muted'>We've
integrated Google Translate service to improve dictionary lookup</
small></td>
      </tr>
    </table>
  </div>
</div>
```

This readable chunk of markup will generate the following fancy projects overview tab:

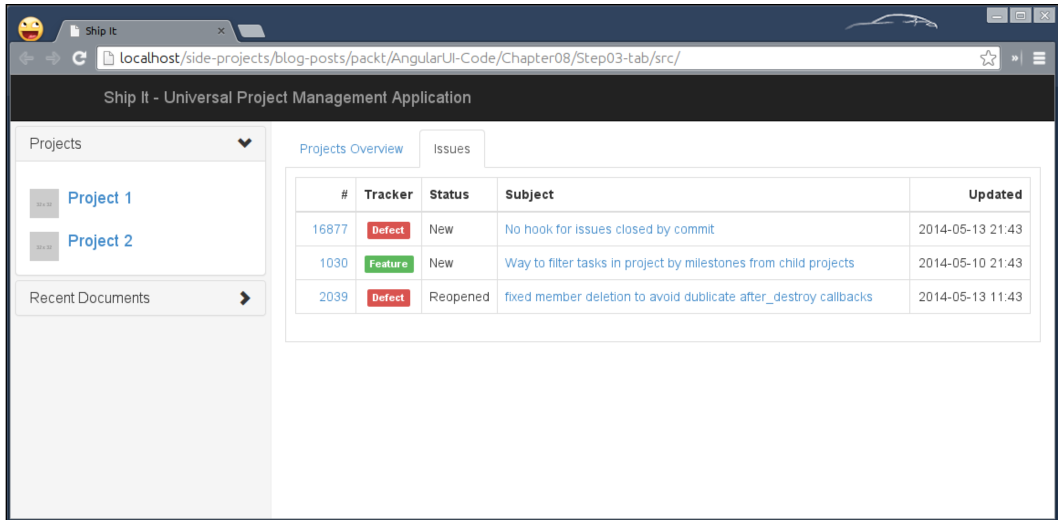


Let's update the **Issues** tab to show a list of open issues in a grid. So, replace the `Issues not found` text inside the second tab with the following markup:

```
<table class="table table-bordered">
  <thead>
    <tr>
      <th class="text-right">#</th>
      <th class="text-center">Tracker</th>
      <th>Status</th>
      <th>Subject</th>
      <th class="text-right">Updated</th>
    </tr>
  </thead>
  <tbody>
```

```
      <tr>
        <td class="col-sm-1 text-right"><a href="#/issue/16877">16877</
  a></td>
        <td class="col-sm-1 text-center"><span class='label label-
  danger'>Defect</span></td>
        <td class="col-sm-1">New</td>
        <td class="col-sm-7"><a href="#/issue/16877">No hook for issues
  closed by commit</a></td>
        <td class="col-sm-2 text-right">2014-05-13 21:43</td>
      </tr>
    </tbody>
  </table>
```

Here is what you'll see if you click on the **Issues** tab:



Notice that this table is not interactive in the sense that we can not sort issues by updated date or status, which is a quite useful requirement.

> We could have used `NgGrid` that you learned about in *Chapter 4, Customizing and Exploring ng-grid*, but I'll leave that to you as an exercise.

Before we finish with tabs, we'll add one more tab without text but with an icon in order to add a new issue. If the tab heading is too complex to handle extra markups (bold/italic) or a custom icon, then AngularUI Bootstrap's tab component also supports another way to achieve this as shown. Just add the following snippet after the last tab we added earlier:

```
<tab>
  <tab-heading>
    <i class="glyphicon glyphicon-plus"></i>
  </tab-heading>
  <!-- new issue form goes here -->
</tab>
```

This will create a tab with the plus sign without any text but we'll make it alive in the next section.

Glyphicons is a library of precisely prepared monochromatic icons and symbols, created with an emphasis on simplicity and easy orientation. Glyphicons contains all the basic symbols for everyday work. These symbols include graphic layouts of websites, mobile, or web applications; they also include symbols for the creation of wireframes and prints. The biggest benefit of Glyphicons is its perfectly prepared data, modern look, and easy usability for graphics of all kinds.

Hiding less relevant content with collapse

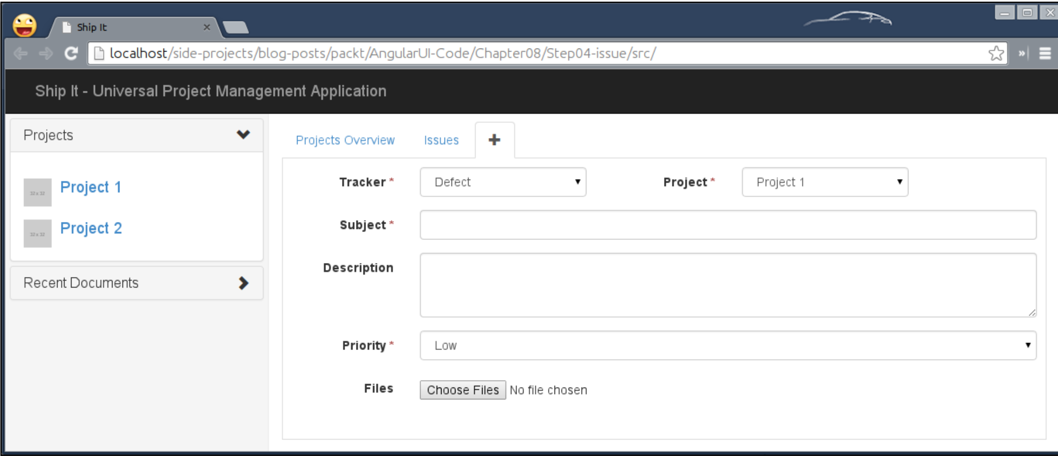
The AngularJS version of Bootstrap's collapse plugin provides a simple way to hide and show an element with a CSS transition. Let's create a simple form to add a new issue to our PMA. So, quickly replace the `<!-- new issue form goes here -->` comment with the following markup:

```
<form class="form-horizontal" role="form">
  <div class="form-group">
    <label class="col-lg-2 control-label">
      Tracker <span class="text-danger">*</span>
    </label>
    <div class="col-lg-3">
      <select class="form-control">
        <option value="defect">Defect</option>
        <option value="feature">Feature</option>
```

```
          <option value="patch">Patch</option>
        </select>
      </div>
      <label class="col-lg-2 control-label">
        Project <span class="text-danger">*</span>
      </label>
      <div class="col-lg-3">
        <select class="form-control">
          <option value="project1">Project 1</option>
          <option value="project2">Project 2</option>
        </select>
      </div>
    </div>
    <div class="form-group">
      <label class="col-lg-2 control-label">
        Subject <span class="text-danger">*</span>
      </label>
      <div class="col-lg-10">
        <input type="text" class="form-control">
      </div>
    </div>
    <div class="form-group">
      <label class="col-lg-2 control-label">
        Description
      </label>
      <div class="col-lg-10">
        <textarea class="form-control" rows="3"></textarea>
      </div>
    </div>
    <div class="form-group">
      <label class="col-lg-2 control-label">
        Priority <span class="text-danger">*</span>
      </label>
      <div class="col-lg-10">
        <select class="form-control">
          <option value="low">Low</option>
          <option value="normal">Normal</option>
          <option value="high">High</option>
          <option value="urgent">Urgent</option>
          <option value="immediate">Immediate</option>
        </select>
      </div>
    </div>
    <div class="form-group">
```

```
        <label class="col-lg-2 control-label">
          Files
        </label>
        <div class="col-lg-2" style='padding-top:8px;'>
          <input type="file" multiple><br>
        </div>
        <!-- Collaspse goes here -->
      </div>
    </form>
```

This will allow you to choose a tracker, set the priority, and upload attachments to provide more clarity about the issue as shown in the following figure:

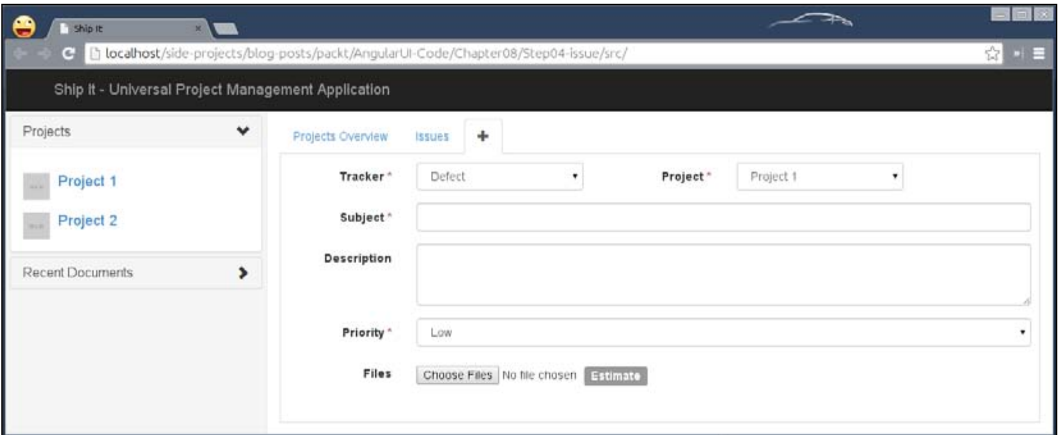


As you have already imagined, there must be a way to add estimates in terms of hours to be spent to fix the issue. However, it's not mandatory for all kinds of issues. So, we'll reveal the details using the collapse widget. Let's replace the `<!-- Collaspse goes here -->` comment with the following:

```
    <label class="col-lg-2 control-label" style="margin-top: -8px;">
    <h4 ng-init="showEstimate = false" ng-click="showEstimate =
    !showEstimate"><span class="label label-default">Estimate</span></h4>
    </label>
    <div class="col-lg-6 form-inline">
      <div class="well" collapse="showEstimate">
        <p><input type="text" class="form-control"> Hours and
        <b>% Done</b>
        <select class="form-control">
          <option value="0">0%</option>
          <option value="10">10%</option>
```

```
        <option value="20">20%</option>
        <option value="30">30%</option>
        <option value="40">40%</option>
        <option value="50">50%</option>
        <option value="60">60%</option>
        <option value="70">70%</option>
        <option value="80">80%</option>
        <option value="90">90%</option>
        <option value="100">100%</option>
      </select></p>
    </div>
  </div>
```

Here, we use an AngularJS model named `showEstimate`, which is set to `false` by default using `ngInit` in order to keep it hidden by default. Then, we just toggle it via `ngClick`. Finally, we pass the model to the `collapse` directive to hide/reveal the detail with animation as follows:



Go ahead and try clicking on the **Estimate** button to see it in action.

Setting timelines with datepicker

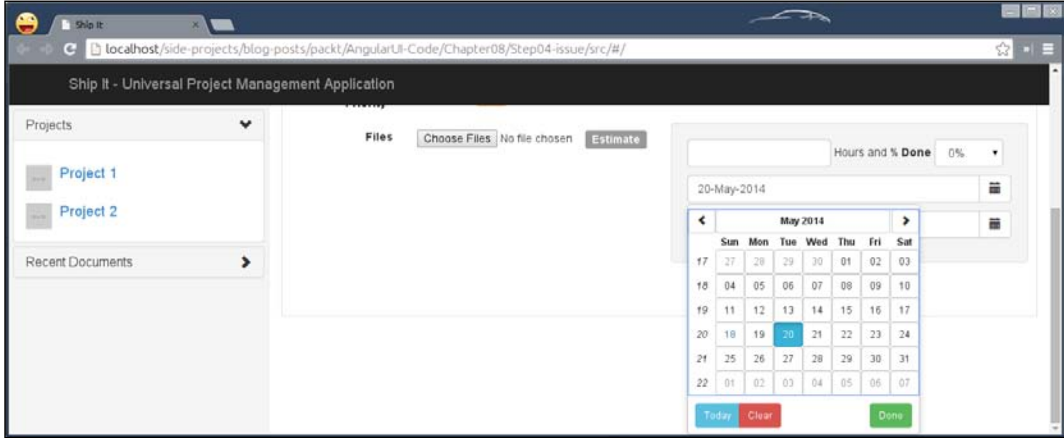
Along with the hours needed to fix the issue, the dates are also necessary so you know when a developer begins his work. The **datepicker** widget will be useful to set a timeline in terms of from and to dates that are required to fix an issue or implement a new feature. Let's add following markup inside the collapse directive as follows:

```
<p class="input-group">
  <input type="text" class="form-control" datepicker-popup="dd-MMMM-
yyyy" ng-model="dateFrom" is-open="dateFrom.opened" placeholder="From
Date"/>
  <span class="input-group-btn">
    <button type="button" class="btn btn-default" ng-click="$event.
stopPropagation();dateFrom.opened = true;"><i class="glyphicon
glyphicon-calendar"></i></button>
  </span>
</p>
<p class="input-group">
  <input type="text" class="form-control" datepicker-popup="dd-MMMM-
yyyy" ng-model="dateTo" is-open="dateTo.opened" placeholder="To
Date"/>
  <span class="input-group-btn">
    <button type="button" class="btn btn-default" ng-click="$event.
stopPropagation();dateTo.opened = true;"><i class="glyphicon
glyphicon-calendar"></i></button>
  </span>
</p>
```

A few things to remember are:

- The `datepicker-popup` option takes a format of the date to be displayed
- The AngularJS model named `dateFrom` holds the actual selected date
- The `dateFrom.opened` AngularJS model keeps track of the datepicker's visibility and that's why we set it to `true` if the calendar button is clicked

You will finally see a nice calendar with this minimal declarative markup:



Utilizing buttons

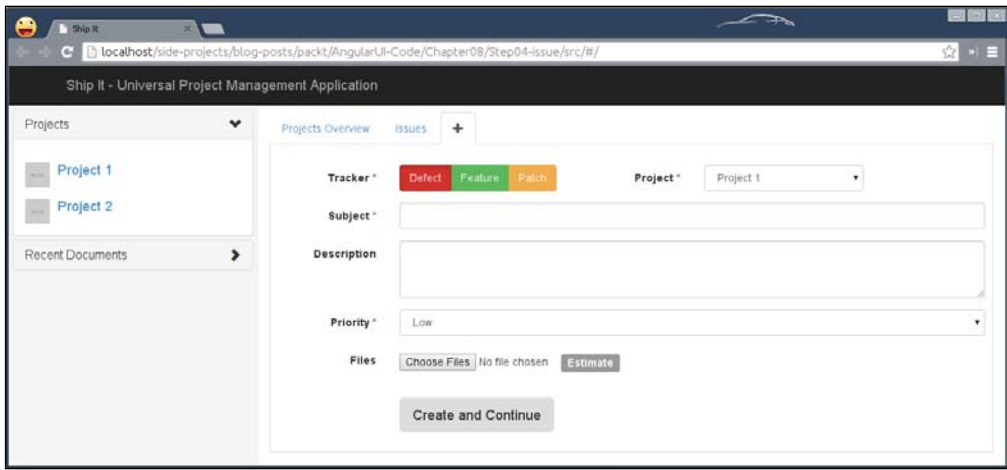
To improve the form and make it fun to add new issues, we'll revamp the tracker option and replace it with the buttons widget to make it visually rich. There are two directives (`btn-checkbox` and `btn-radio`) that can make a group of buttons behave like a set of checkboxes, radio buttons, or a hybrid where radio buttons can be unchecked. Let's use the radio format and replace the select box that we added earlier for the tracker option with the following:

```
<div class="btn-group">
  <label class="btn btn-danger" ng-model="tracker" btn-
radio="'defect'">Defect</label>
  <label class="btn btn-success" ng-model="tracker" btn-
radio="'feature'">Feature</label>
  <label class="btn btn-warning" ng-model="tracker" btn-
radio="'patch'">Patch</label>
</div>
```

The `btn-radio` directive allows us to set the value for the model when clicked so that the respective radio button gets selected by deselecting the previously selected one. We'll also add a save button to quickly save the issue. Add following line of code after the files row:

```
<div class="form-group">
  <label class="col-lg-2"></label>
  <div class="col-lg-10">
    <button type="button" class="btn btn-inverse btn-lg">Create and
Continue</button>
  </div>
</div>
```

This will complete our **add new issue** form as follows:

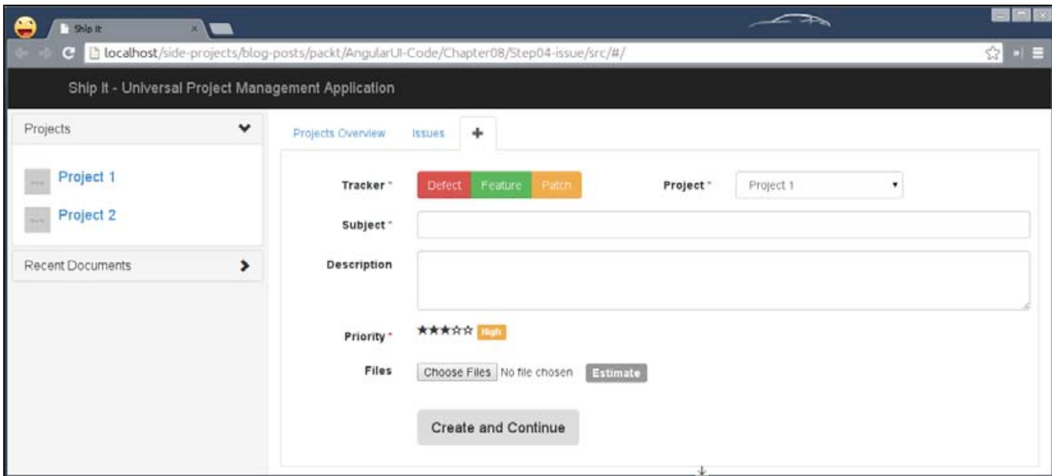


Converting priorities in the form of ratings

To improve usability, instead of having a typical select box to choose a priority for issues from, we could use star ratings. So, replace the previous select box with the following:

```
<rating ng-model="priority" max="5" on-hover="prepriority=value"></
rating>
<span class="label" ng-class="{'label-default': prepriority == 1,
'label-success': prepriority == 2, 'label-warning': prepriority == 3,
'label-info': prepriority == 4, 'label-danger': prepriority == 5}">
{{['Low', 'Normal', 'High', 'Urgent', 'Immediate'][prepriority - 1]}}
</span>
```

On hovering, we can get the value of a star, which we store in a model named `prepriority` and update the label accordingly. The `ngClass` directive lets you apply CSS classes conditionally. Refer to the following screenshot:



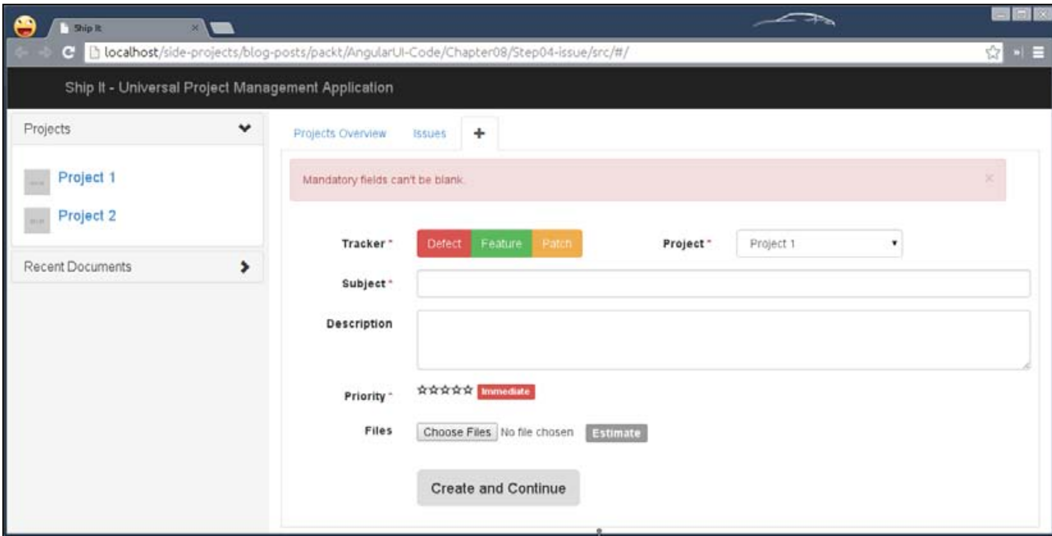
Try hovering over each individual star and notice how priority changes from low to high.

Notifying users with alert messages

The last thing remaining is that as we have some of the fields mandatory in the form; we need to show alert messages in case there is any error. **Alert** is an AngularJS version of Bootstrap's alert. This directive can be used to generate alerts from the dynamic model data (using the `ng-repeat` directive). Let's add a fancy alert message on top of the form as follows:

```
<div class="form-group col-lg-12">
  <alert type="danger" close="hideAlert=true" ng-
hide="hideAlert">Mandatory fields can't be blank.</alert>
</div>
```

Note that the `type` attribute determines the color for the alert. The presence of the `close` attributes figures if a close button is displayed or not. Also, we have extended its behavior by adding a custom action to hide the alert using the `hideAlert` model, which is eventually used with `ngHide` to manually hide the alert. You can see it here:



However, the alert should be displayed only if the form is submitted, which I'll leave to you as an exercise. Go explore!

Using carousel

Now from the **Issues** tab, we'll give you a provision to open any issue to see the details of it. For this, we'll need the `angular-route` module for easy browser navigation and also to maintain its history. We'll first install `angular-route` using the following command on the terminal:

```
bower install --save angular-route
```

Then, update `index.html` by referencing `angular-route.js` after `angular.js` and also update `app.js` to set up a dependency as follows:

```
angular.module('myApp', ['ui.bootstrap', 'ngRoute');
```

Now, we can create a route definition to map the URL (#/issue/1234) in order to load the relevant view. The routes should be defined before the application bootstraps, so the AngularJS configuration phase is the ideal place for this:

```
angular.module('myApp', ['ui.bootstrap', 'ngRoute'])
.config(function($routeProvider) {
      $routeProvider.when('/issue/:issue', {templateUrl: 'views/view-
issue.tpl.html'});
  $routeProvider.otherwise({redirectTo: '/'});
});
```

As you can see, `$routeProvider` takes a simple key/value store, that is, the key is the actual route we want to map and the value is an object that consists of various options such as a template to load, an AngularJS controller to bind, and so on. Here, we simply want to load `view-issue.tpl.html` as our template.

Now, create a new `controllers.js` file in the `js/` directory and put the following snippet in it:

```
angular.module('myApp.controllers', [])
.controller('ViewIssueCtrl', function($scope, $route, $element,
$timeout) {
  $scope.$watch(
    function() { return $route.current && $route.current.params.issue;
},        function(newVal, oldVal) {
    $scope.showTab = !!newVal;
    $scope.issueId = newVal;
    if ($scope.showTab) {
      $timeout(function() {
        $element.find('a').click();
      }, 0, false);
    }
  });
});
```

Note the following:

- The `$route` object in the controller gives access to the current route information, that is, the issue number in this case.
- We then created a custom `$watcher` object to make sure that the tab is loaded only if user navigates to any issue. That's why `showTab` is there to keep track of the tab's visibility and immediately select the same tab dynamically if it's visible.
- Using `$element`, we find the tab and open it dynamically. The `$timeout` object is an AngularJS wrapper around `window.setTimeout`, which delays the execution until the tab is rendered.

So, let's update `index.html` to create a new tab in order to view any issue that the user wants to check out. Add the following before the **Add Issue** tab we added earlier:

```
<tab ng-controller="ViewIssueCtrl" ng-show="showTab">
  <tab-heading>
    <i class="glyphicon glyphicon-warning-sign"></i>
    Issue #{{issueId}}
  </tab-heading>
  <div ng-view></div>
</tab>
```

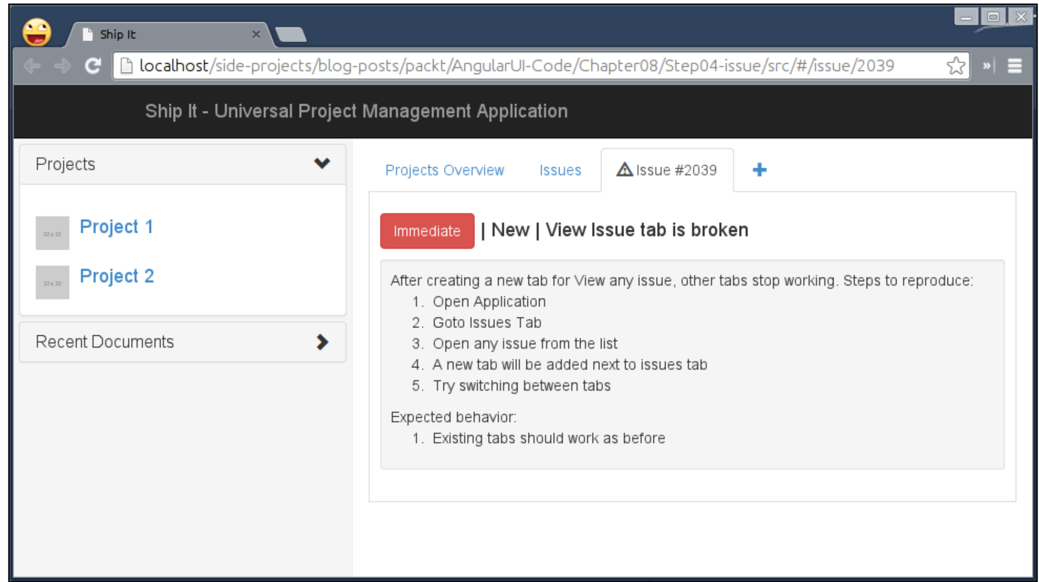
There is nothing to be scared of; we've just used `showTab` model in the DOM, which is set to `true`/`false` in the `$watch` method to the show/hide the tab. The `ng-view` directive is a helper directive in AngularJS that allows us to load the template for our route. So, our `view-issue.tpl.html` file will be loaded into it.

Now, create `view-issue.tpl.html` in the `views/` directory, containing the following code:

```
<h4><span class='btn btn-danger'>Immediate</span> | New | View Issue
tab is broken</h4>
<div class="well well-sm">
  After creating a new tab to View any issue, other tabs stop working.
Steps to reproduce:<br/><br/>
  <ol>
    <li>Open Application</li>
    <li>Goto Issues Tab</li>
    <li>Open any issue from the list</li>
    <li>A new tab will be added next to issues tab</li>
    <li>Try switching between tabs</li>
  </ol>
```

```
        Expected behavior:<br><br>
    <ol>
      <li>Existing tabs should work as before</li>
    </ol>
  </div>
```

This just displays dummy information about the issue, as follows:



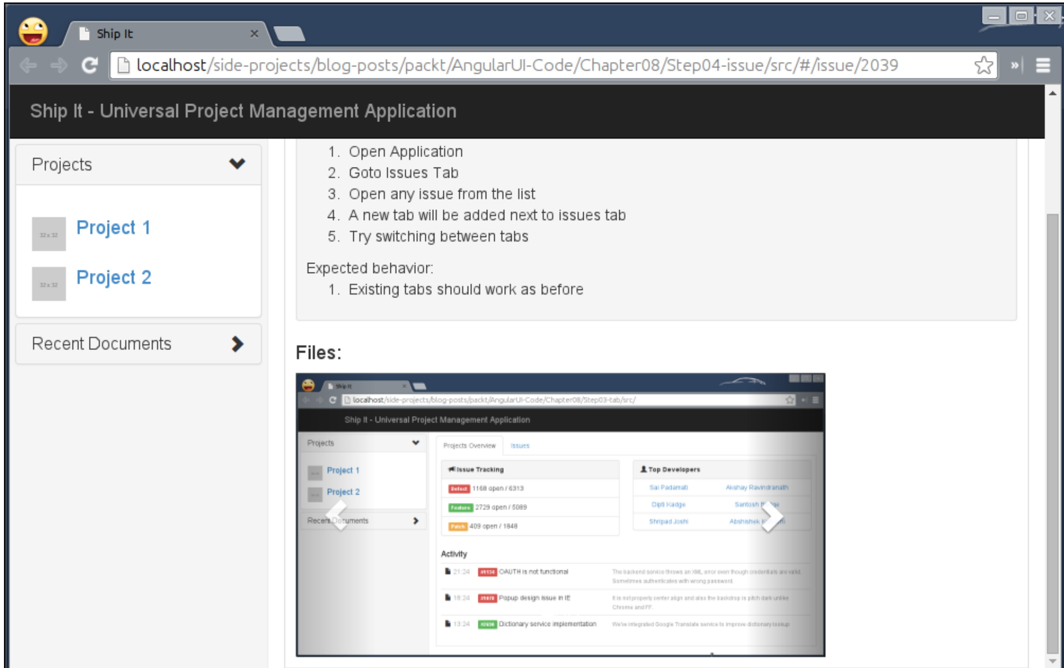
Remember that we also allow the attaching of images to add more clarity on the issue to make it easy for developers to fix it. Let's create a carousal of uploaded images against this issue in `view-issue.tpl.html` as follows:

```
<h4>Files: </h4>
<carousel style="width: 500px;">
  <slide ng-repeat="item in carousel" active="item.active">
    <img ng-src="{{item.image}}">
    <div class="carousel-caption">
      <p>{{item.text}}</p>
    </div>
  </slide>
</carousel>
```

We are just rendering a list of images uploaded against the issue, which are wrapped within the `carousel` directive to render a beautiful carousel in a snap. To enable the carousel, we'll create a collection named `carousel` in our `ViewIssueCtrl` object as follows:

```
$scope.carousel = [
  { image: 'images/1.png' },
  { image: 'images/2.png' },
  { image: 'images/3.png' }
];
```

Let's look at the nice image carousel in action:



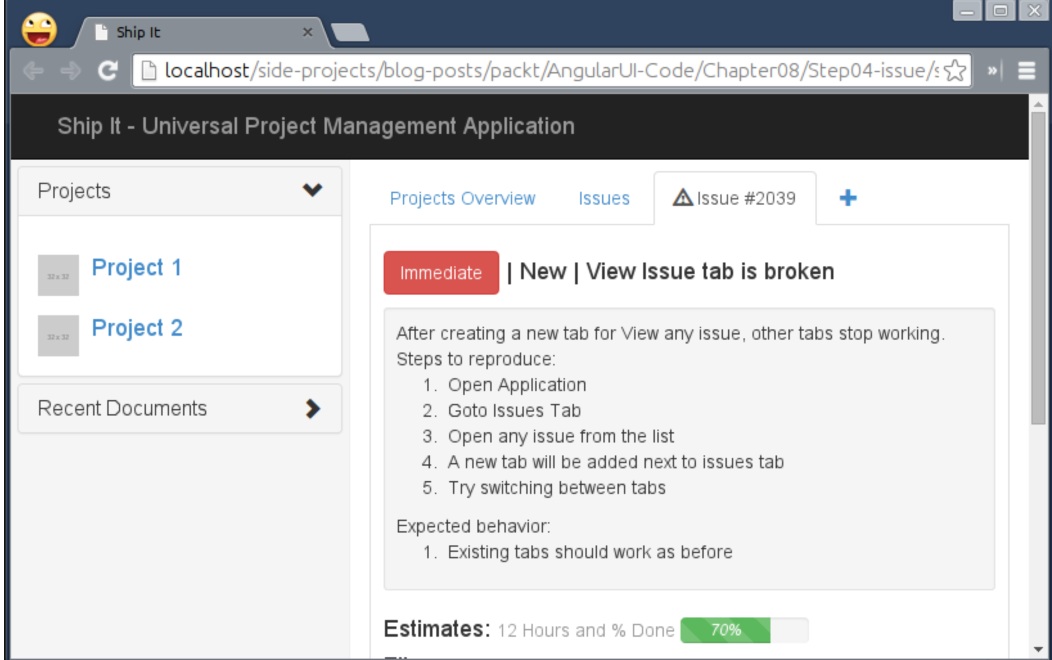
You can even add intervals in seconds so that it will automatically rotate images. The carousel also offers support for touchscreen devices in the form of swiping. To enable swiping, load the `ngTouch` module in the way we added the `angularRoute` module earlier in the chapter.

A progress bar to show the status of an issue

If you remember, we added the `estimates` option (optional) to add the number of hours to be spent fixing the issue and its progress as a percentage. We'll convert those percentages into a progress bar to highlight them visually. So, add the following markup before the carousel in `view-issue.tpl.html`:

```
<h4>Estimates:
  <small>12 Hours and % Done </small>
  <progressbar class="progress-striped active" max="100" value="70"
type="success" style="width: 100px;display: inline-block;margin-
bottom: -5px;"><i>70%</i></progressbar>
</h4>
```

We often need to show feedback of the progress with different colors. You can apply types such as success for green, danger for red, and so on, to highlight the progress. You can also customize it with predefined options, such as maximum range, current range, and so forth. This is what it could look like:



Efficient suggestions with typeahead

Users will not always just go to the issues tab to find a particular issue, so there has to be a way to search quickly. Using AngularUI Bootstrap's `typeahead` directive, you can easily achieve that. Let's add a search field in the header next to our application name (`.navbar-header`) as:

```
  <form class="navbar-form navbar-right" ng-controller="FindCtrl">
    <input type="text" class="form-control" placeholder="Find
Issues" ng-model="find" typeahead="issue.name for issue in issues |
filter:$viewValue" typeahead-on-select='onSelect($item)'>
  </form>
```

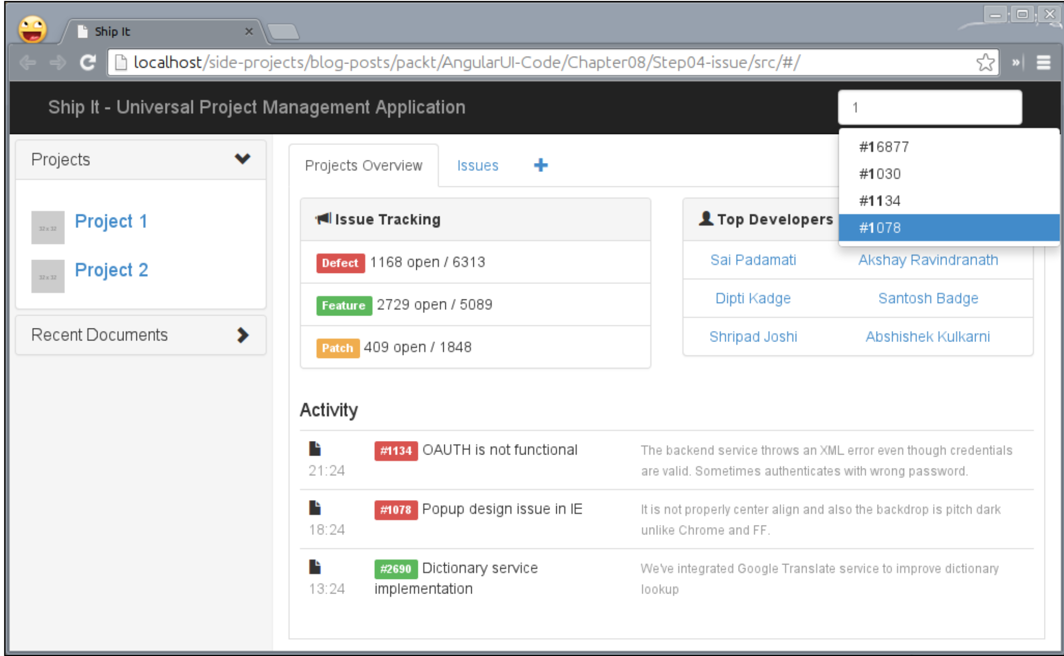
Things to remember here are:

- The `typeahead` directive takes a collection of items to search through, which is then filtered based on a search criteria such as `$viewValue`
- Finally, when a user selects any of the options listed, we simply call a method, `onSelect`, to change the route to `#/issue/:issueNumber`

As you might have noticed that we are relying on a new controller for this, let's create it and put the following code in `controllers.js`:

```
angular.module('myApp.controllers', [])
.controller('ViewIssueCtrl', function($scope, $route, $element,
$timeout) {
   // removed this for brevity
}).controller('FindCtrl', function($scope, $location) {
  $scope.issues = [
    {'id': 16877, 'name': '#16877'},
    {'id': 1030, 'name': '#1030'},
    {'id': 2039, 'name': '#2039'},
    {'id': 1134, 'name': '#1134'},
    {'id': 1078, 'name': '#1078'},
    {'id': 2690, 'name': '#2690'}
  ];

  $scope.onSelect = function(item) {
    $location.path('/issue/' + item.id);
    $scope.find = '';
  };
});
```

To change the route, we have used the `$location` built-in AngularJS service to manipulate URLs. Although it offers many methods such as `.hash`, `.replace`, `.search`, `.host`, and so on, we'll just use `$location.path` to change the route, say, from `#/` to `#/issue/1078`. Let's play with it:



Go ahead and select any issue to see details about it in a **View Issue** tab.

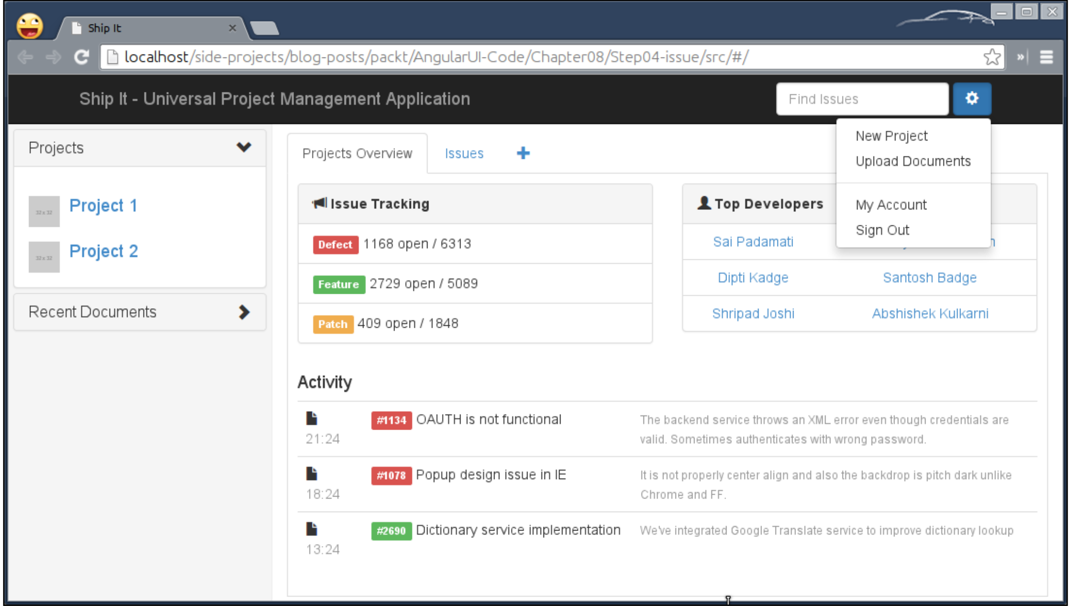
Common housing for application-specific menus with a dropdown

In every application, we often need an interface to show a bunch of choices floating (unlike collapse) but we want to keep them hidden if they are not needed. AngularUI Bootstrap's dropdown is a simple directive that will toggle a drop-down menu when clicked or programmatically. Let's quickly add the following markup after `typeahead`:

```
<div class="btn-group" dropdown >
  <button type="button" class="btn btn-primary dropdown-toggle">
    <span class='glyphicon glyphicon-cog'></span>
  </button>
```

```
    <ul class="dropdown-menu" role="menu">
      <li><a href="#/">New Project</a></li>
      <li><a href="#/document">Upload Documents</a></li>
      <li class="divider"></li>
      <li><a href="#/settings">My Account</a></li>
      <li><a href="#/login">Sign Out</a></li>
    </ul>
  </div>
```

Note that the attribute named `dropdown` is mandatory to activate it. The button with the `.dropdown-toggle` class becomes a handler that shows a menu sitting next to it as `.dropdown-menu`. Check it out in the following screenshot:



You can even track when the dropdown opens/closes with `on-toggle='callback(open)'` to perform further actions. So far, it is a static application, not communicating to any server to fetch data, but this can easily be accomplished with the REST APIs integration that we saw in *Chapter 6, Using Charts and Data-driven Graphics*. Go take this application to the next level!

This chapter was all about integration of AngularUI Bootstrap to build complex web apps in a matter of time that would have taken weeks. Although the application we built in this chapter does not talk to any server to fetch live data because of space constraints, I would like to share some insights into how you can make it dynamic to look like a real PMA. As you already learned about fetching dynamic data in *Chapter 6*, *Using Charts and Data-driven Graphics*, you can leverage this knowledge to fuel it using the `$http` service that enables the consumption of REST-based APIs.

You probably have to use `node.js` or any other server-side programming language that can lay out the structure for the REST APIs by talking to a database such as **MongoDB**, **CouchDB**, **MySQL**, and so on.

In addition, you can even store all the static data used in the application in JSON files, which you can consume via the `$http` service to create an illusion of being a dynamic application and later set up the base to glue the actual REST APIs.

Summary

In this chapter, we went through the history a bit and figured why projects such as AngularUI came into existence to create dependency-free widgets in pure AngularJS. We also went through some of the very useful components such as accordion, tab, ratings, carousel, and so on to use them declaratively. Finally, we built a well-designed PMA in no time.

In the next chapter, we'll look at customizing AngularUI Bootstrap.

9

Customizing AngularUI Bootstrap

In this chapter, you will learn how AngularUI Bootstrap can be customized by editing the underlying templates of the directives that are part of the project. This knowledge is crucial when trying to become an expert in UI development using AngularJS.

We'll cover the following topics in this chapter:

- Understanding what external templates are and ways to load them
- Customizing the existing AngularUI Bootstrap pagination widget to make it look similar to the one used by Google Search
- Understanding how the AngularUI Bootstrap tab component works internally
- Upgrading the same to have few extra features such as closing a tab or adding a new tab dynamically

Introduction to external templates

The awesome PMA that we built in the previous chapter was written on top of AngularUI Bootstrap's components and all the widgets we used had precompiled templates that we did not even touch. If you remember, we just added the `ui-bootstrap-tpls.js` file to use them as is. Every widget or component consists of a logic (or behavior) and underlying markup (or template) that shapes it. In *Chapter 6*, *Using Charts and Data-driven Graphics*, when we built a pie chart directive for the first time, we relied on the template option of the directive as:

```
angular.module('myApp.directives', []).directive('barChart',
function() {
  return {
```

```
        restrict: 'E',
        template: "<div class='container' ng-style='setContainer()'>\
          <div class='bar' ng-repeat='bar in bars' ng-
    style='setDetails(bar, $index)'>{{bar.color}} - {{bar.percentage}}%</
    div>\
        </div>"
      };
    });
```

If we want to build a complex widget, it is difficult to maintain huge markup within the directive definition as it is now. So, keeping the template outside is a flexible as well as robust approach to follow.

Loading a template via the script tag

The easiest way to abstract the template out is to keep it inside a pair of script tags. Usually, we use text/JavaScript as a type to load any JavaScript code for a browser to recognize and parse it to be available during the life cycle of a web app. The AngularJS template does not contain JavaScript but declarative HTML, which has to be evaluated in the context of AngularJS to make it alive. This is why AngularJS encourages the use of a special type of attribute such as `text/ng-template`. Browsers ignore it but AngularJS does compile it before using a special type of attribute. This is how we'll keep the template for the `barChart` directive in a script tag:

```
    <script type="text/ng-template" id="bar-chart.html">
        <div class='container' ng-style='setContainer()'>
          <div class='bar' ng-repeat='bar in bars' ng-
    style='setDetails(bar, $index)'>{{bar.color}} - {{bar.percentage}}%</
    div>
        </div>
    </script>
```

Note that the template must have a unique ID, which is used as a key by AngularJS internals to cache for later use. Also, it is not mandatory to keep this script tag inside a head tag. Alternatively, you can even create a new file named `bar-chart.html` and have the same markup in it by getting rid of the script tags altogether. The next time if you request the template via a URL, AngularJS triggers an XHR or AJAX call to grab its content, compile, and cache it for later use. Both behave in the same way with only one difference, that is, a round-trip to the server to fetch the template can be avoided if script tags are used.

Loading a template via $templateCache

You already learned what AngularJS does when you request a template, but the real trick here is that it uses a `$templateCache` service under the hood to cache the request and its response. Note that the cache content stays during the life cycle of AngularJS and will be wiped out if the page is reloaded but will be around when routes change. AngularJS lets us use the `$templateCache` service directly to avoid any mediators such as a script tag or template URL. The `$templateCache` service is just a service with the setter and getter methods to cache a template and retrieve it later, respectively. This is how you can cache a template:

```
angular.module('myApp.directives', []).run(function($templateCache) {
    $templateCache.put('bar-chart.html',
        <div class='container' ng-style='setContainer()'>
           <div class='bar' ng-repeat='bar in bars' ng-
style='setDetails(bar, $index)'>{{bar.color}} - {{bar.percentage}}%</
div>
        </div>
    );
});
```

As it's an AngularJS service, you need to inject it before using. The setter method named `put` takes two parameters, that is, the template ID and the template itself. This benefits in a way that the template is retrieved from the cache when requested. It is useful to keep all your templates in one place similar to `angular-bootstrap-tpls.js` to organize your application well.

Using an external template

In order to use an external template, we'll need a slight modification to our `pieChart` directive. The AngularJS directive definition object allows us to load an external template via the `templateUrl` option. Let's quickly update the directive as follows:

```
angular.module('myApp.directives', []).directive('barChart',
function() {
    return {
       restrict: 'E',
       templateUrl: "bar-chart.html"
    };
});
```

This way, when the directive is compiled, the AngularJS compiler will first look for the template in a cache with the same key, that is, `bar-chart.html`. If it is not available, it will initiate an AJAX request to fetch the same. Additionally, you can pass a function that returns a template reference conditionally as follows:

```
angular.module('myApp.directives', []).directive('barChart',
function() {
    return {
       restrict: 'E',
       templateUrl: function(element, attrs) {
           return attrs.touch == 'false' ? "bar-chart.html" : "bar-
advance-chart.html";
       }
    };
});
```

Assuming that the environment is a touch device, you might just want to load a simpler version of the bar chart and an advanced one on desktop browsers.

Customizing the AngularUI Bootstrap pagination widget

Now you understand how external templates can be cached and used, you are pretty much aware of the fact that AngularUI Bootstrap uses the same technique for all the components it offers. Let's look at how pagination widget works. To begin with, we'll use the same template from the `Step04-issue` folder of *Chapter 8, AngularUI Bootstrap Development* and replace `index.html` as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Google Pagination</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
    <link rel="stylesheet" href="css/main.css">
</head>
<body ng-app="myApp">
  <div ng-controller="GooglePaginationCtrl">
    <pagination total-items="totalItems" ng-model="currentPage"></
pagination>
  </div>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-bootstrap/ui-bootstrap-tpls.
js"></script>
```

```
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
</body>
</html>
```

Here, we are simply using AngularUI Bootstrap's pagination widget with the default template. The `ng-model` directive takes a selected current page number.

Let's update `controllers.js` as follows:

```
angular.module('myApp.controllers', []).controller('GooglePaginationCt
rl', function($scope {
  $scope.totalItems = 100;
  $scope.currentPage = 2;
});
```

By default, it shows 10 items per page, so we'll see 10 page numbers to navigate as shown in the following figure:

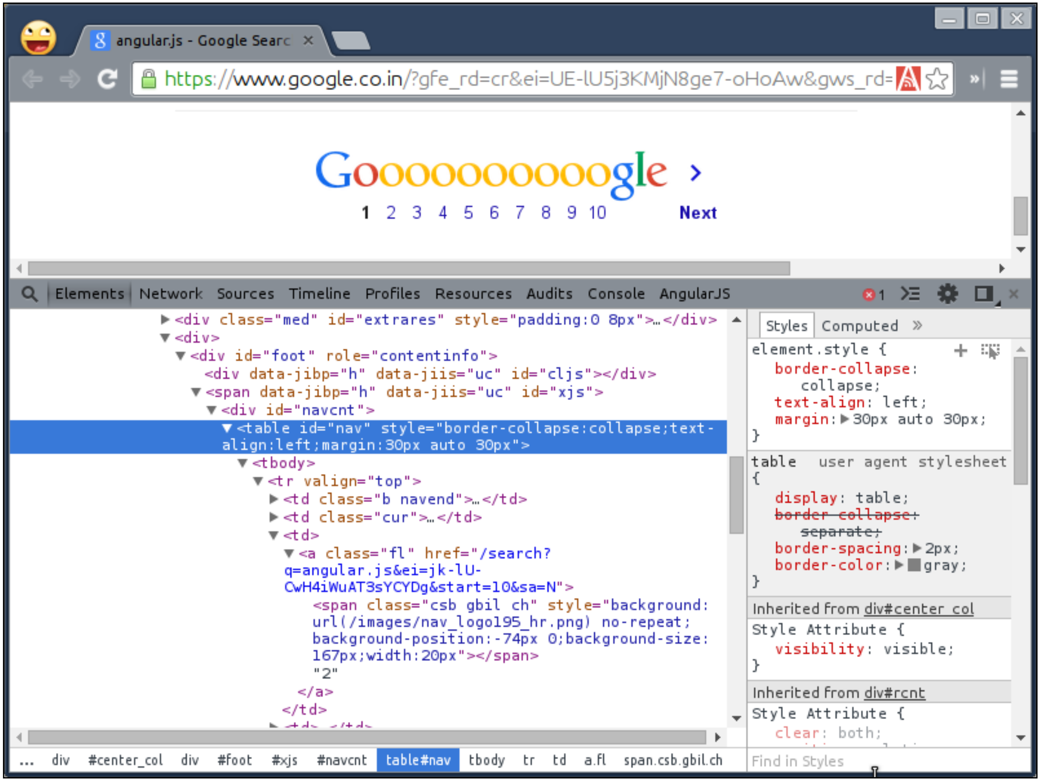
If you want to take a look at how AngularJS rendered it, go to `ui-bootstrap-tpls.js` and search for `template/pagination/pagination.html`. Here it is for your reference:

```
angular.module("template/pagination/pagination.html", []).
run(["$templateCache", function($templateCache) {
  $templateCache.put("template/pagination/pagination.html",
    <ul class="pagination">
      <li ng-if="boundaryLinks" ng-class="{disabled: noPrevious()}"><a
href ng-click="selectPage(1)">{{getText('first')}}</a></li>
      <li ng-if="directionLinks" ng-class="{disabled:
noPrevious()}"><a href ng-click="selectPage(page -
1)">{{getText('previous')}}</a></li>
```

```
        <li ng-repeat="page in pages track by $index" ng-class="{active: page.active}"><a href ng-click="selectPage(page.number)">{{page.text}}</a></li>
        <li ng-if="directionLinks" ng-class="{disabled: noNext()}"><a href ng-click="selectPage(page + 1)">{{getText('next')}}</a></li>
        <li ng-if="boundaryLinks" ng-class="{disabled: noNext()}"><a href ng-click="selectPage(totalPages)">{{getText('last')}}</a></li>
      </ul>);
}]);
```

Keep an eye on the methods used to enable/disable navigation buttons or handle click events because even though we use custom template, it should behave the same. This means we need to use the same method calls in the custom template wherever possible or exclude any if not required. As this chapter is all about customizing AngularUI Bootstrap; we'll convert this typical pagination to a fancy one. Imagine that you want to use Google-like pagination, which we often notice on their search page; let's see how to do it.

Go to `http:// google.com`, search for something, and inspect their pagination in the developer tool. You will notice that they are using an HTML table to render it as follows:



Here, I've abstracted it for you to understand:

```
angular.module('myApp.controllers', []).controller('GooglePaginationCt
rl', function($scope, $templateCache) {
  $templateCache.put("template/pagination/pagination.html",
    '<table class="google-pagination">\
    <tbody>\
      <tr valign="top">\
        <td><!-- Prev Navigation  --></td>\
        <td><!-- Page Numbers --></td>\
        <td><!-- Next Navigation --></td>\
      </tr>\
    </tbody>\
  </table>'
  );

  $scope.totalItems = 100;
  $scope.currentPage = 2;
});
```

Now replace `<!-- Prev Navigation -->` with:

```
<td>\
  <span ng-if="noPrevious()" class="csb noprev"></span>\
  <a -if="!noPrevious()" ng-click="selectPage(page - 1)"><span
class="csb prev"></span></a>\
</td>\
```

The first `span` element will be shown if the previous action is disabled and the second `span` element lets you navigate to the previous page otherwise. Similarly, replace `<!-- Next Navigation -->` with:

```
<td>\
  <span ng-if="noNext()" class="csb nonext"></span>\
  <a href ng-if="!noNext()" ng-click="selectPage(page + 1)"><span
class="csb next"></span></a>\
</td>\
```

Same thing here, we just want to disable the next button if we are on the last page or keep it enabled otherwise. Finally, replace the pagination column, `<!-- Page Numbers -->`, with:

```
<td ng-repeat="page in pages track by $index" ng-
click="selectPage(page.number)">\
  <span ng-if="page.active"><span class="csb active"></span>{{page.
text}}</span>\
  <a ng-if="!page.active"><span class="csb nonactive"></span>{{page.
text}}</a>\
</td>\
```

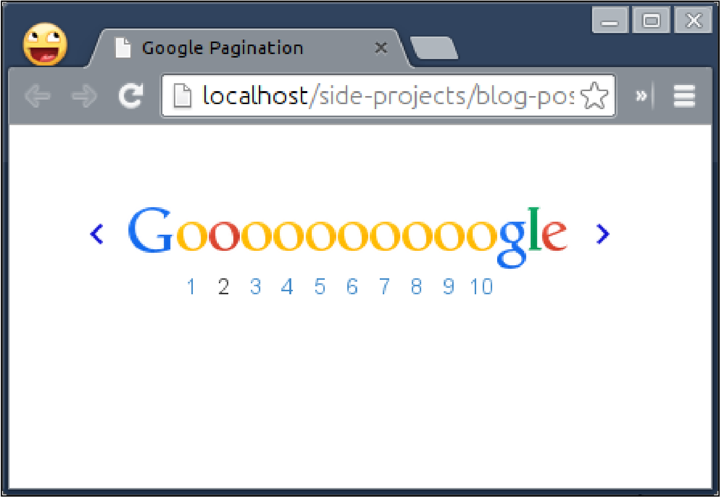
This basically renders a long list of page numbers, depending upon total number of items to list. Then, we attach a click handler on the `td` element to navigate to the particular page when clicked on. Note that we have not written any of these methods as they are already a part of the pagination directive.

Let's spice it up with a pinch of CSS. Add the following in `css/main.css`:

```
    body {
  padding: 50px;
}
.google-pagination {
  text-align: center;
}
.google-pagination .csb {
  height: 40px;
  display: block;
}
.google-pagination .csb.prev {
  background:url(https://www.google.co.in/images/nav_logo195_hr.png)
no-repeat;
  background-position:0 0;
  background-size:167px;
  width:53px;
}
.google-pagination .csb.noprev {
  background:url(https://www.google.co.in/images/nav_logo195_hr.png)
no-repeat;
  background-position:-24px 0 !important;
  background-size:167px;
  width:28px !important;
}
.google-pagination .csb.active {
  background:url(https://www.google.co.in/images/nav_logo195_hr.png)
no-repeat;
  background-position:-53px 0;
  background-size:167px;
  width:20px;
}
.google-pagination .csb.nonactive {
  background:url(https://www.google.co.in/images/nav_logo195_hr.png)
no-repeat;
  background-position:-74px 0;
  background-size:167px;
```

```
    width:20px;
  }
  .google-pagination .csb.nonext {
    background:url(https://www.google.co.in/images/nav_logo195_hr.png)
  no-repeat;
    background-position:-96px 0 !important;
    background-size:167px;
    width:45px !important;
  }
  .google-pagination .csb.next {
    background:url(https://www.google.co.in/images/nav_logo195_hr.png)
  no-repeat;
    background-position:-96px 0;
    background-size:167px;
    width:71px;
  }
```

We've borrowed the same styles that Google uses for their pagination. We've just added some extra CSS classes, `next`, `nonext`, `prev`, and `noprev`, to avoid clutter with inline styles, and our Google pagination is ready:



Even though it looks like the Google Search pagination now but it is still a Bootstrap pagination widget, which works as before.

Extending the AngularUI Bootstrap tab widget

So far, you have seen how easy it is to skin a particular AngularUI Bootstrap widget, but we have not yet explored an area wherein we can extend the functionality of the existing widget to expect more from it. The AngularUI Bootstrap tab component is the best place to start with as it does not allow us to remove a tab out of the box. Allowing us to close the opened tabs will be a really useful feature to have in our Project Management Application called **ShipIt** that we built in the previous chapter. Imagine, every project, issue, or document you want to have a look at will be opened in a new tab, which you can close if not needed.

Before we begin, let's look at what the `tab` component is made up of. The `tab` component is a combination of two nested directives named `tabset` and `tab`. A `tabset` directive is just a wrapper that comprises a list of tabs and their contents. Here is the default template used:

```
<div>
  <ul class="navnav-{{type || 'tabs'}}" ng-class="{'nav-stacked': vertical, 'nav-justified': justified}" ng-transclude></ul>
  <div class="tab-content">
    <div class="tab-pane" ng-repeat="tab in tabs" ng-class="{active: tab.active}" tab-content-transclude="tab"></div>
  </div>
</div>
```

A few things to note:

- The tabs are created using the `li` HTML elements that are included inside the `ul` element highlighted previously using a built-in AngularJS directive called `ngTransclude`. The `ngTransclude` directive allows us to feed the external markup, which was added while defining in the DOM.
- The `tab-content-transclude` directive is a custom one that facilitates the inclusion of tab content in a special wrapper called `tabPane`.

When the `tabset` and `tab` directives get compiled, both will be replaced by this template. As said earlier, the `tab` directive uses a simple `li` element to make up the tab interface as follows:

```
<li ng-class="{active: active, disabled: disabled}">
  <a ng-click="select()" tab-heading-transclude>{{heading}}</a>
</li>
```

This goes right into the `ul` element that we saw before. The heading attribute set on the `tab` directive is wrapped by an anchor element using the `tabHeadingTransclude` custom directive. This makes the picture clear for us that in order to customize the tab interface, we need to update the template used by the `tab` directive. Let's do it then.

Create an `index.html` file for an advanced tab example as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-
scalable=no">
    <title>Advance Tab</title>
    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
    <link rel="stylesheet" href="css/main.css">
</head>
<body ng-app="myApp">
<!-- tabs go here -->
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-bootstrap/ui-bootstrap-tpls.
js"></script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
</body>
</html>
```

Now, let's have a collection for tabs that consists of the tab heading and tab content in `js/controllers.js` as follows:

```
angular.module('myApp.controllers', []).controller('AdvanceTabCtrl',
function($scope, $templateCache) {
  $templateCache.put("template/tabs/tab.html",
     '<li ng-class="{active: active, disabled: disabled}">\
       <a ng-click="select()" tab-heading-transclude>\
         <button type="button" class="close" ng-click="$parent.
close($parent.$index)" ng-style="{true: {visibility: \'hidden\'}}
[!active]">&times;</button>\
         {{heading}}\
       </a>\
     </li>'
  );
  $scope.tabList = [{
```

```
      heading: 'Tab 1', content: '1.html'
    }, {
      heading: 'Tab 2', content: '2.html'
    }];
    $templateCache.put('1.html', '<h1>I am One</h1>');
    $templateCache.put('2.html', '<h1>I am Two</h1>');
    $scope.close = function(tabScope) {
      $scope.tabList.splice(tabScope, 1);
    };
  });
```

There are a few things to note here:

- First of all, we override the existing template to have a close button next to each tab in order to close it. We make sure that it is only visible for the selected tab using `ngStyle`.
- We bind a `click` handler to the close icon, which is defined outside the existing directive but in our controller named `AdvanceTabCtrl`. Note that the `tabset` directive creates an isolate scope that means it is not prototypically inherited from the parent scope, that is, the scope for the `AdvanceTabCtrl` controller. In JavaScript, objects can be prototypically inherited, which means that if a method or function is not available on the object, then it goes up the prototype chain to find the same on the parent object from which the current object is derived. Similarly, a `close` method is not available on the current scope; it will go up the scope chain to find the same. However, a tab widget does not inherit the scope in which it is defined to have complete isolation for the widget as well as to avoid accidental read/write. Even though it is not prototypically inherited from the parent scope, AngularJS still maintains a reference of its parent scope via the `$parent` object. So, to access the close method defined in `AdvanceTabCtrl`, we need to use `$parent.close`.
- We then have a collection of tabs and their templates defined. Additionally, you can keep the tab templates in separate files for brevity.

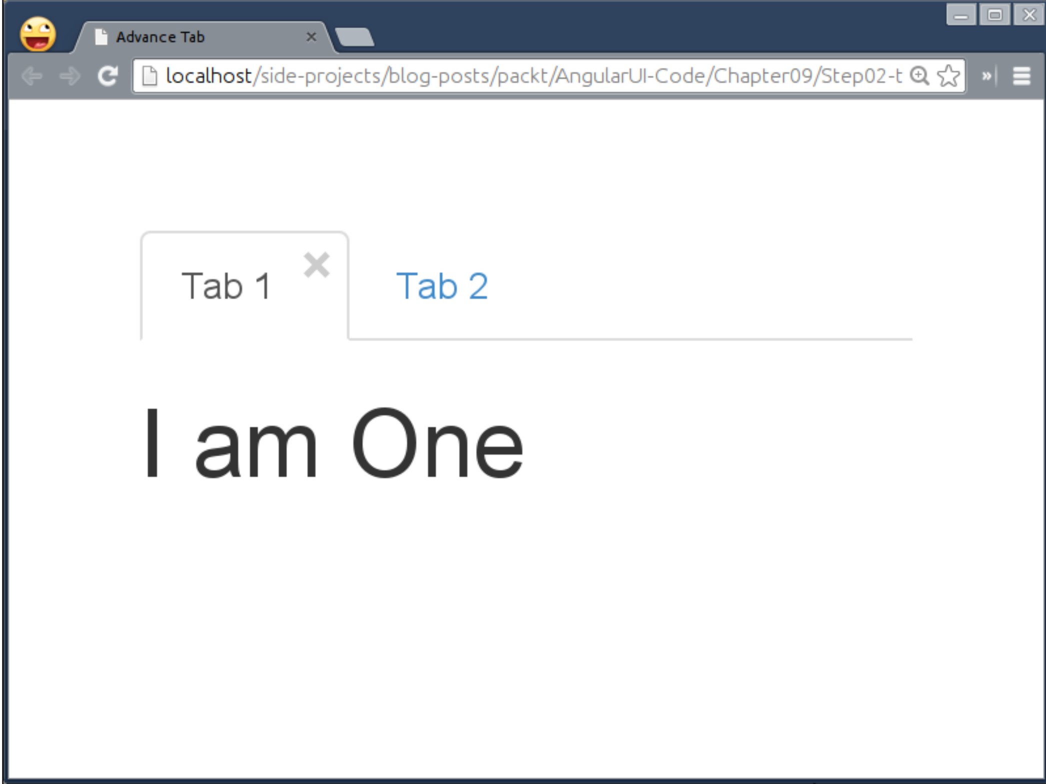
Now, replace `<!-- tab goes here -->` with the following in `index.html`:

```
<div ng-controller="AdvanceTabCtrl">
    <tabset>
      <tab heading="{{tab.heading}}" ng-repeat="tab in tabList">
        <div ng-include="tab.content"></div>
      </tab>
    </tabset>
  </div>
```

In this, we are simply iterating over a collection of tabs to render them dynamically. The `ngInclude` directive allows us to inject the content of each tab defined by `$templateCache` in the controller. Finally, add minor CSS styles to position the icon correctly in `css/main.css` as follows:

```
.close {
  position: relative;
  top: -8px;
  right: -10px;
}
```

Now, take a look at your end:



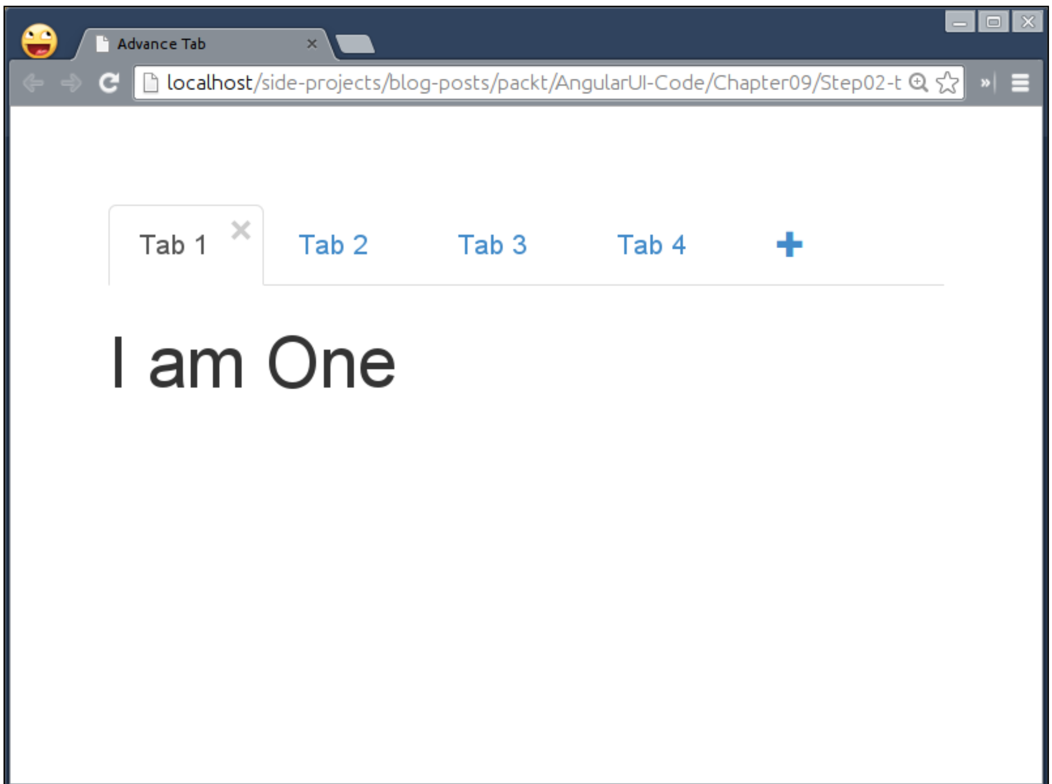
Wouldn't it be great to have an add button to create tabs dynamically? Let's do it. We'll add a custom tab at the end with a plus icon on it. To distinguish it from the other tabs and have it behave differently, we'll set `new_tab` as a heading for it and check the same to toggle the behavior for other tabs. Let's add the following in `controllers.js` as follows:

```
$scope.tabList = [{
    heading: 'Tab 1', content: '1.html'
  }, {
    heading: 'Tab 2', content: '2.html'
  }, {
    heading: 'new_tab',
    content: ''
  }];
$scope.add = function() {
    $scope.tabList.splice($scope.tabList.length - 1, 0, {heading: 'Tab
' + $scope.tabList.length, content: ''});
  };
```

This will add a new tab at the end but before `new_tab`. Now, update the template a bit like this:

```
$templateCache.put("template/tabs/tab.html",
  '<li ng-class="{true: {active: active, disabled: disabled}}[heading
!= \'new_tab\']">\
    <a ng-show="heading != \'new_tab\'" ng-click="select()" tab-
heading-transclude>\
      <button type="button" class="close" ng-click="$parent.
close($parent.$index)" ng-style="{true: {visibility: \'hidden\'}}
[!active]">&times;</button>\
      {{heading}}\
    </a>\
    <a ng-show="heading == \'new_tab\'" ng-click="$parent.add()">\
      <i class="glyphicon glyphicon-plus"></i>\
    </a>\
  </li>'
);
```

Nothing has changed. We just hide the default tab interface if the heading equals `new_tab`. That's it! Here is how it looks after adding few more tabs:



Now, imagine ShipIt for a second and you will realize that having a close icon for all tabs does not make sense at all because sometimes, we need a few tabs available all the time. We can achieve this easily by introducing a special attribute called `closeable` as follows:

```
$scope.tabList = [{
  heading: 'Tab 1', content: '1.html', closeable: false
}, {
  heading: 'Tab 2', content: '2.html'
}, {
  heading: 'new_tab',
  content: ''
}];
```

We'll keep all tabs closeable by default, use `closeable: false` otherwise. Then, update the template slightly to hide the close icon if `closeable` is set to `false`:

```
$templateCache.put("template/tabs/tab.html",
  '<li ng-class="{true: {active: active, disabled: disabled}}[heading
!= \'new_tab\']">\
    <a ng-show="heading != \'new_tab\'" ng-click="select()" tab-
heading-transclude>\
      <button type="button" class="close" ng-click="$parent.
close($parent.$index)" ng-style="{true: {visibility: \'hidden\'}}
[!active || $parent.tab.closeable == false]">&times;</button>\
      {{heading}}\
    </a>\
    <a ng-show="heading == \'new_tab\'" ng-click="$parent.add()">\
      <i class="glyphicon glyphicon-plus"></i>\
    </a>\
  </li>'
);
```

Here, we want to hide the close icon if the tab is not active or not allowed to close. Cool! Now you are free to use the upgraded tab widget in our ShipIt application to have a dynamic tab when any project, issue, or document is opened. I'll leave that to you as an exercise.

Summary

In this chapter, you got to know that the external templates are nothing but an HTML snippet consumed by widgets/components. With this new learning, we tried to change the look and feel of the pagination widget to make it look like the Google Search pagination for fun. Then, we looked at the complex AngularUI Bootstrap `tab` component to extend its behavior to allow users to close the tab manually and add a new one dynamically. We analyzed the scope hierarchy to have the widgets interact with the outside world for advanced interactions.

In the next chapter, we'll discover mobile development using AngularJS and Bootstrap.

10
Mobile Development Using AngularJS and Bootstrap

In this chapter, you will learn how to build a mobile single-page application that adapts nicely to the varying screen sizes of current smartphones. We'll also include page optimizations for a slick user experience over a slow network connection, which extends what we covered when we set up the build environment in the first chapter.

We'll be covering the following topics in this chapter:

- Why mobile application?
- Creating a bookmarking application with mobile users in mind
- Making the application dynamic so that bookmarks will be persisted in `localStorage`, browser's internal storage
- Adding touch gestures to quickly navigate through bookmarks on mobile devices instead of relying on standard pagination
- Optimizing animations used while navigating through bookmarks by offloading it to GPU to make it buttery smooth
- Using various Grunt plugins to optimize the production build size from 1.4 MB to 370 KB to improve initial page load on low bandwidth

Why bother about mobile?

For years, many of us designed websites or web applications for the desktop PC. At that time, smartphones were costly and bandwidth costed a lot, so the majority of people were using desktop browsers to browse the Internet. However, things have changed dramatically over the past few years; even developing countries have access to smartphones at an affordable price these days. According to the trend, people use smartphones more than desktops on a daily basis now, which puts a question mark on whether we should focus on mobile first and gracefully degrade it to work on desktop as well. The answer to this is yes, because designing for mobile first not only prepares you for the explosive growth and opportunities in this space, but it also forces you to focus and enables you to innovate. So, keep in mind that whatever you build on the Web should work seamlessly on mobile devices too. Refer to *Chapter 7*, *Customizing AngularJS with CSS and CSS Frameworks*, to understand the benefit of responsive design over fixed layout if you have not. Let's build a mobile application in the next section.

Building a bookmarking app with the mobile-first approach

In this section, we are going to resurrect the bookmarking application built on top of **Foundation** in *Chapter 7*, *Customizing AngularJS with CSS and CSS Frameworks*. We'll port it on Twitter Bootstrap in this chapter to make it mobile friendly. To begin with, you can take the `Chapter07/Step03-foundation/` source code from the companion suite as a base. Assuming you have made a copy of the same as `Chapter10/Step01-mobile`, it's time to get our hands dirty. However, I'll not go over how to design an application in Twitter Bootstrap again as it was covered in *Chapter 7*, *Customizing AngularJS with CSS and CSS Frameworks* already.

Go into the `Chapter10/Step01-mobile` directory and run following commands to install Twitter Bootstrap:

```
cd Chapter10/Step01-mobile
bower install --save bootstrap
```

With Bootstrap, we'll try to keep the layout same as what we designed in Foundation earlier. First of all, replace the content of `index.html` with the following markup:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, user-scalable=no">
<title>Pin It</title>
```

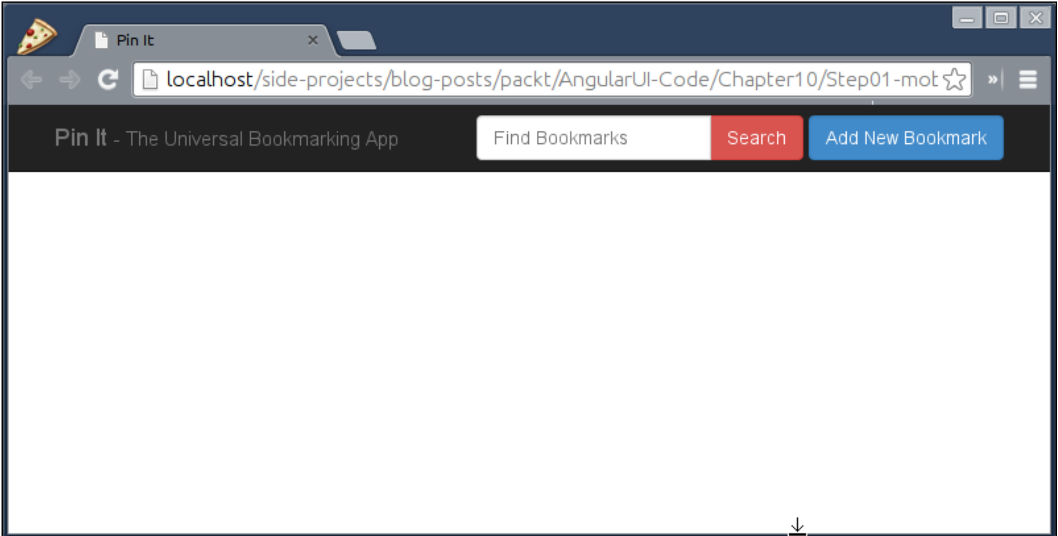
```
<link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
</head>
<body ng-app="myApp">
<div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
<div class="container">
<div class="navbar-header">
<button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">Pin It <small>- The Universal
Bookmarking App</small></a>
</div>
<div class="navbar-collapse collapse">
<form class="navbar-form navbar-right" role="form">
<div class="input-group">
<input type="text" class="form-control" placeholder="Find Bookmarks">
<span class="input-group-btn">
<button class="btn btn-danger">Search</button>
</span>
</div>
<button class='btn btn-primary form-control' data-toggle="modal" data-
target="#addBookmark">Add New Bookmark</button>
</form>
</div>
</div>
</div>

<!-- A list of bookmarks with pagination go here -->

<!-- Add Bookmark Modal goes here -->

<script src="bower_components/jquery/dist/jquery.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/bootstrap/dist/js/bootstrap.js"></
script>
<script src="js/app.js"></script>
</body>
</html>
```

First, we replace the Foundation style sheet used earlier with Twitter Bootstrap. Then, customize the top bar to have a collapsible but responsive header so that it turns into a nice hamburger icon that you can expand to get to the search box and add the bookmark option on small-scale devices. This works pretty much same as Foundation with slight differences in the CSS naming conventions. Finally, we replace the Foundation JavaScript reference with `bootstrap.js`.

Then, update `app.js` with:

```
angular.module('myApp', []).run(function() { });
```

Here, we simply removed the constructor call used for Foundation before as Bootstrap widgets work out of the box by just injecting the respective JavaScript in the page. You can now take a look at it in a browser:



It's time to add some dummy bookmarks now. Just replace the `<!-- A list of bookmarks with pagination go here -->` comment added earlier with the following markup:

```
<div class="container-fluid" style="padding-top: 70px;">
<div class="row">
<div class="col-xs-3 col-sm-3 col-md-2 col-lg-2 text-right">
<a href="#"><img src="img/angular20.jpg"></a>
</div>
<div class="col-xs-9 col-sm-9 col-md-10 col-lg-10">
<b>AngularJS 2.0 is a framework for mobile apps</b>
<p>
```

```
<a href="http://blog.angularjs.org/2014/03/angular-20.html" target='_
blank'>http://blog.angularjs.org/2014/03/angular-20.html</a>
</p>
<span class="label label-primary">angular</span>
<span class="label label-default">angular advanced</span>
<span class="label label-danger">mvc</span>
</div>
</div>
<hr>
<div class="row">
<div class="col-xs-3 col-sm-3 col-md-2 col-lg-2 text-right">
<a href="#"><img src="img/durandal.jpg" /></a>
</div>
<div class="col-xs-9 col-sm-9 col-md-10 col-lg-10">
<b>Angular and Durandal Converge</b>
<p>
<a href="http://blog.angularjs.org/2014/04/angular-and-durandal-
converge.html" target='_blank'>http://blog.angularjs.org/2014/04/
angular-and-durandal-converge.html</a>
</p>
<span class="label label-default">angular</span>
<span class="label label-primary">durandal</span>
</div>
</div>
<hr>

<div class='text-center'>
<ul class="pagination">
<li><a href="#">&laquo;</a></li>
<li><a href="#">1</a></li>
<li><a href="#">2</a></li>
<li><a href="#">3</a></li>
<li><a href="#">4</a></li>
<li><a href="#">5</a></li>
<li><a href="#">&raquo;</a></li>
</ul>
</div>
```

This will simply render a bookmark with a thumbnail next to it, quite similar to the Foundation layout. The only thing to notice here is that we are using Bootstrap's grid system to have a responsive layout in order to adapt varying sizes of screen resolutions and form factors. If you refresh the tab, you will be quite happy to see the exact same application in Twitter Bootstrap now. Woohoo!

Making the application dynamic

In this section, we'll store new bookmarks in `localStorage` in order to make it dynamic. Let's enable an **add new bookmark** button. Replace the `<!-- Add Bookmark Modal goes here -->` comment in `index.html` with the following:

```
<div ng-controller="BookmarkCtrl" class="modal fade" id="addBookmark"
tabindex="-1" role="dialog">
<div class="modal-dialog">
<div class="modal-content">
<div class="modal-header">
<button type="button" class="close">
<span aria-hidden="true">&times;</span>
<span class="sr-only">Close</span></button>
<h4 class="modal-title" id="myModalLabel">New Bookmark</h4>
</div>
<div class="modal-body">
<div class="form-group">
<label for="url">URL</label>
<input type="url" class="form-control" id="url" placeholder="Add
bookmark URL" ng-model="bookmark.url">
</div>
<div class="form-group">
<label for="title">Title</label>
<input type="text" class="form-control" id="title" placeholder="Add
bookmark title" ng-model="bookmark.title">
</div>
<div class="form-group">
<label for="title">Tags</label>
<input type="text" class="form-control" id="title" placeholder="comma
separated tags i.e. angular, jquery, foundation" ng-model="bookmark.
tags">
</div>
</div>
<div class="modal-footer">
<button type="button" class="btn btn-primary btn-block" data-
dismiss="modal"ng-disabled="!bookmark.url || !bookmark.title ||
!bookmark.tags" ng-click="save()">Save Bookmark</button>
</div>
</div>
</div>
</div>
```

A few points to note are as follows:

- Remember we did not call a Bootstrap construction function in `app.js` unlike Foundation. This is because the `data-*` attribute makes all the difference. In order to open a modal, we simply added `data-target="#addBookmark"` to tell the button to find an element with the same ID, which in this case is `addBookmark`, to open up as a modal when clicked on.
- We are going to have a separate controller named `BookmarkCtrl` to isolate it from the other parts of the application to avoid accidentally overriding any scoped variables. Also, the controller helps you make the application modular and manageable when it grows.
- We have three form fields, `url`, `title`, and `tags`, binded to the `$scope` object of the controller with `ngModel`. The `ngModel` directive stores the form control's state or value in its respective model so that we can access it using, say, `$scope.url`, instead of `$('url').val()` in jQuery.
- Then, we keep the **Submit** button disabled until the form is filled as expected. We can store a new bookmark then.

To make it work, add `controllers.js` and `services.js` in `index.html` after `app.js` as follows:

```
<script src="js/controllers.js"></script>
<script src="js/services.js"></script>
```

Create `controllers.js` as follows:

```
angular.module('myApp.controllers', [])
.controller('BookmarkCtrl', function($scope, BookmarkStore,
$rootScope, $element) {
    $scope.save = function() {
      var arrBookmarks = BookmarkStore.get();

      arrBookmarks.push({
        url     : $scope.bookmark.url,
        title   : $scope.bookmark.title,
        tags    : $scope.bookmark.tags.split(','),
        created : new Date().getTime()
      });

      BookmarkStore.put(arrBookmarks);
      $scope.bookmark = null;
$rootScope.$broadcast('mainCtrl:loadBookmarks', {});
      $element.modal('hide');
    };
  })
```

We just read the form values via the `$scope` object and store them in `localStorage` using a custom service named `BookmarkStore`, which we are yet to write. Then, we broadcast an event named `mainCtrl:loadBookmarks` towards `MainCtrl` to inform it to refresh the list. Remember the `$broadcasted` event goes downwards from `$rootScope` to all of its child scopes and their children. The `$element` parameter refers to the element on which we applied `BookmarkCtrl` in `bookmarks.html`.

Let's create it as `js/services.js`:

```
angular.module('myApp.services', [])
  .factory('BookmarkStore', function() {
    return {
      searchCriteria: '',

      get: function(sorted) {
        var bookmarks = JSON.parse(localStorage.getItem('pin-
bookmarks') || '[]');
        return sorted ? bookmarks.sort(function(a, b) { return
a.created <b.created; }) : bookmarks;
      },

      put: function(bookmarks) {
        localStorage.setItem('pin-bookmarks', JSON.
stringify(bookmarks));
      }
    };
  });
```

The `factory` function is the easiest way to create a service that returns a literal object. In our case, it consists of two methods, `get` and `put`, to fetch stored bookmarks and store new bookmarks respectively in `localStorage`.

Then, quickly update `app.js` to reference the newly created controllers and services as follows:

```
angular.module('myApp', ['myApp.controllers', 'myApp.services'])
```

You can now go ahead and try adding couple of bookmarks.

You might notice that a list of bookmarks does not update, and this is because it's static. Let's make it dynamic too. We'll basically use the same `BookmarkStore` service to fetch all the stored bookmarks using the `.get()` method call and render all of them with the `ngRepeat` directive in AngularJS. Replace the previously added the `div.container-fluid` block with the following:

```
<div ng-controller="MainCtrl" class="container-fluid" style="padding-
top: 70px;">
```

```
<div class="row" ng-repeat-start="bookmark in pageBookmarks">
<div class="col-xs-3 col-sm-3 col-md-2 col-lg-2 text-right">
<a ng-href="{{bookmark.url}}" target='_blank'><img src="http://
lorempixel.com/random/80/80"></a>
</div>
<div class="col-xs-9 col-sm-9 col-md-10 col-lg-10">
<b ng-bind="bookmark.title"></b>
<p>
<a ng-href="{{bookmark.url}}" target='_blank' ng-bind="bookmark.
url"></a>
</p>
<span style="margin-right: 5px;" class="label" ng-class="{0:'label-
default', 1: 'label-primary', 2: 'label-success', 3: 'label-info', 4:
'label-warning', 5: 'label-danger'}[$index % 6]" ng-repeat="tag in
bookmark.tags" ng-bind="tag"></span>
</div>
</div>
<hr ng-repeat-end>

<div class='text-center'>
<ul class="pagination">
<li ng-class="{'disabled': isFirst()}">
<span ng-if="isFirst()">&laquo;</span>
<a ng-if="!isFirst()" ng-href="#/page/{{currentPage - 1}}">&laquo;</a>
</li>
<li ng-repeat="page in pages" ng-class="{'active': page ==
currentPage}">
<a ng-href="#/page/{{$index + 1}}" ng-bind="page"></a>
</li>
<li ng-class="{'disabled': isLast()}">
<span ng-if="isLast()">&raquo;</span>
<a ng-if="!isLast()" ng-href="#/page/{{currentPage + 1}}">&raquo;</a>
</li>
</ul>
</div>
</div>
```

Note the following points:

- The `ngRepeatStart` and `ngRepeatEnd` directives are variations of the `ngRepeat` directive to repeat over multiple elements without any root node. The `ngRepeat` directive only works with the root node. In our case, we repeat both the `div.row` and `hr` elements together using `ngRepeatStart` and `ngRepeatEnd`.

- We then render the bookmark title and URL using `ngBind`, which is a replacement for double curly base syntax in AngularJS for data binding. The benefit of using the directive over a curly notation is that it avoids flash of unstyled content and shows a value after it is parsed.
- Later, we loop through a list of tags to display them with `ngRepeat` and apply different styles to distinguish them.
- At the end, we render the dynamic pagination based on the number of bookmarks we list.

Now, add `MainCtrl` in `controllers.js` as follows:

```
angular.module('myApp.controllers', [])
  .controller('MainCtrl', function($scope, BookmarkStore, $location) {
loadBookmarks();

    $scope.$on('mainCtrl:loadBookmarks', loadBookmarks);

    function loadBookmarks() {
      $scope.bookmarks = BookmarkStore.get(true);
      $scope.perPage = 10;
      $scope.totalPages = Math.ceil($scope.bookmarks.length / $scope.
perPage);
      $scope.currentPage = 1;
      if ($scope.currentPage > $scope.totalPages) {
        $location.path('/page/1');
      } else {
        $scope.pageBookmarks = $scope.bookmarks.slice(($scope.
currentPage - 1) * $scope.perPage, $scope.currentPage * $scope.
perPage);
        $scope.pages = [];
        for (var i = 0; i < $scope.totalPages; i++) {
          $scope.pages.push(i + 1);
        }
      }
    }

    $scope.isFirst = function() {
      return $scope.currentPage === 1;
    };

$scope.isLast = function() {
      return $scope.currentPage === $scope.totalPages;
    };
  })
```

Note the following:

- Again, we use a separate controller named `MainCtrl` for isolation from other parts of the application.
- We fetch all stored bookmarks and calculate the total number of pages to be shown by the pagination widget. For the record, we want to show 10 bookmarks per page.
- We also catch an event broadcasted by `BookmarkCtrl` earlier to refresh the list of bookmarks whenever a new bookmark is saved.

You are now free to take a look at the updates. Do not forget to add dozens of bookmarks to see the pagination widget in action.

However, you might notice that the pagination does not work even though the URL changes. This is because we are yet to use the AngularJS route module to maintain browser history for pagination. Let's install `angular-route.js` with the following command in the terminal:

```
bower install --save angular-route
```

Also, reference `angular-route.js` in `index.html` as follows:

```
<script src="bower_components/angular-route/angular-route.js"></
script>
```

Then, reference its module in `app.js` to access the `$routeProvider` service:

```
angular.module('myApp', ['ngRoute', 'myApp.controllers', 'myApp.
services'])
  .config(function($routeProvider){
    $routeProvider.when('/page/:pageNumber', {templateUrl: 'views/
bookmarks.html'});
    $routeProvider.otherwise({redirectTo: '/page/1'});
  });
```

The `ngRoute` module allows us to access `$routeProvider` in the configuration phase in order to set up custom routes for the application. This route simply tells AngularJS that if `pageNumber` is provided in the URL, then fetch `views/bookmark.html`, which is a view for the route. Okay, so where will it be rendered? To solve this problem, AngularJS has a special type of directive named `ng-view` that allows us to render the template into it.

So, create `views/bookmark.html` and move the `div.container-fluid` and `bookmark` modal markups into it. Add the following in `index.html`, replacing the same:

```
<div ng-view></div>
```

Now, we need to tell `MainCtrl` to render a new set of bookmarks when the route changes. To do this, we need to read `pageNumber` from the route object. So, update `controllers.js` as follows:

```
angular.module('myApp.controllers', [])
  .controller('MainCtrl', function($scope, BookmarkStore, $route,
$location) {
    loadBookmarks();

    function loadBookmarks() {
      $scope.bookmarks = BookmarkStore.get(true);
      $scope.perPage = 10;
      $scope.totalPages = Math.ceil($scope.bookmarks.length / $scope.
perPage);
      $scope.currentPage = parseInt($route.current.params.pageNumber,
false) || 1;
      if ($scope.currentPage > $scope.totalPages) {
        $location.path('/page/1');
      } else {
        $scope.pageBookmarks = $scope.bookmarks.slice(($scope.
currentPage - 1) * $scope.perPage, $scope.currentPage * $scope.
perPage);
        $scope.pages = [];
        for (var i = 0; i < $scope.totalPages; i++) {
          $scope.pages.push(i + 1);
        }
      }
    }

    $scope.isFirst = function() {
      return $scope.currentPage === 1;
    };

    $scope.isLast = function() {
      return $scope.currentPage === $scope.totalPages;
    };
  })
```

Give it a shot! The pagination should be working for you now.

Allowing users to search through bookmarks

In this section, we'll allow users to search bookmarks by title or tag to quickly filter out. Update the `.navbar` markup added at the beginning of the chapter with the following one in `index.html`:

```
<div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
<div class="container">
<div class="navbar-header">
<button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</button>
<a class="navbar-brand" href="#">Pin It <small>- The Universal
Bookmarking App</small></a>
</div>
<div class="navbar-collapse collapse">
<form class="navbar-form navbar-right" role="form">
<div class="input-group" ng-controller="SearchCtrl">
<input type="text" class="form-control" placeholder="Find Bookmarks"
ng-model="searchCriteria">
<span class="input-group-btn">
<button class="btn btn-danger" ng-click="searchIt(searchCriteria)">Se
arch</button>
</span>
</div>
<button class='btn btn-primary form-control' data-toggle="modal" data-
target="#addBookmark">Add New Bookmark</button>
</form>
</div><!--/.navbar-collapse -->
</div>
</div>
```

Here, we have a separate controller named `SearchCtrl` that encompasses a model named `searchCriteria` to read the search term. Then, we call the `searchIt()` method to trigger a search call within bookmarks. Add the `SearchCtrl` controller in `controllers.js` after `BookmarkCtrl` as follows:

```
.controller('SearchCtrl', function($scope, BookmarkStore, $location,
$route) {
    $scope.searchIt = function(searchCriteria) {
BookmarkStore.searchCriteria = searchCriteria;
      $location.path('/page/1');
      $route.reload();
    };
  });
```

This is fairly simple. We are simply storing the search term in `BookmarkStore.searchCriteria` and taking the user to the first pagination post search. The `$location` object is a wrapper around the `window.location` object in AngularJS that allows you to modify or read an URL and `querystrings` out of the box.

Finally, update `MainCtrl` in `controllers.js` to read the `searchCriteria` model to filter bookmarks upon searching as follows:

```
angular.module('myApp.controllers', [])
  .controller('MainCtrl', function($scope, BookmarkStore, $route,
$location, $filter) {
    loadBookmarks();

    function loadBookmarks() {
      $scope.bookmarks = $filter('filter')(BookmarkStore.get(true),
BookmarkStore.searchCriteria);
      $scope.perPage = 10;
$scope.totalPages = Math.ceil($scope.bookmarks.length / $scope.
perPage);
      $scope.currentPage = parseInt($route.current.params.pageNumber,
false) || 1;
      if ($scope.currentPage > $scope.totalPages) {
        $location.path('/page/1');
      } else {
        $scope.pageBookmarks = $scope.bookmarks.slice(($scope.
currentPage - 1) * $scope.perPage, $scope.currentPage * $scope.
perPage);
        $scope.pages = [];
        for (var i = 0; i < $scope.totalPages; i++) {
          $scope.pages.push(i + 1);
      }
      }
    }

    $scope.isFirst = function() {
      return $scope.currentPage === 1;
    };

    $scope.isLast = function() {
      return $scope.currentPage === $scope.totalPages;
    };
  })
```

The `$filter` filter takes a collection to search through and a search term itself. Try searching!

Crafting the application for mobile devices

The pagination widget that we used to navigate a list of bookmarks is not really useful in terms of usability in mind for touch devices. People often prefer swipe gestures to navigate to anything quickly. In AngularJS, adding touch support is easier than ever.

First, we need to install the `angular-touch` module separately as it does not come with the core:

```
bower install --save angular-touch
```

Reference `angular-touch.js` in `index.html` after `angular-route.js`:

```
<script src="bower_components/angular-touch/angular-touch.js"></
script>
```

Then, add its module as a dependency in `app.js`:

```
angular.module('myApp', ['ngRoute', 'ngTouch', 'myApp.controllers',
'myApp.services'])
  .config(function($routeProvider){
    $routeProvider.when('/page/:pageNumber', {templateUrl: 'views/
bookmarks.html'});
    $routeProvider.otherwise({redirectTo: '/page/1'});
  })
```

The AngularJS touch module provides touch events and other helpers for touch-enabled devices. This supports a special set of directives to specify custom behavior when any touch action is triggered. There is a 300 ms delay between `touch` and `click` event so that browsers would wait after tap-and-release action before firing the `click` event. However, this module handles it automatically with `ngClick` directive that works out of the box without modifying the underling code. So apply the `ngSwipeLeft` and `ngSwipeRight` directives on `div.container-fluid` in `bookmarks.html` as follows:

```
<div ng-controller="MainCtrl" class="container-fluid" style="padding-
top: 70px;" ng-swipe-left="paginate('forward')" ng-swipe-
right="paginate('backward')">
```

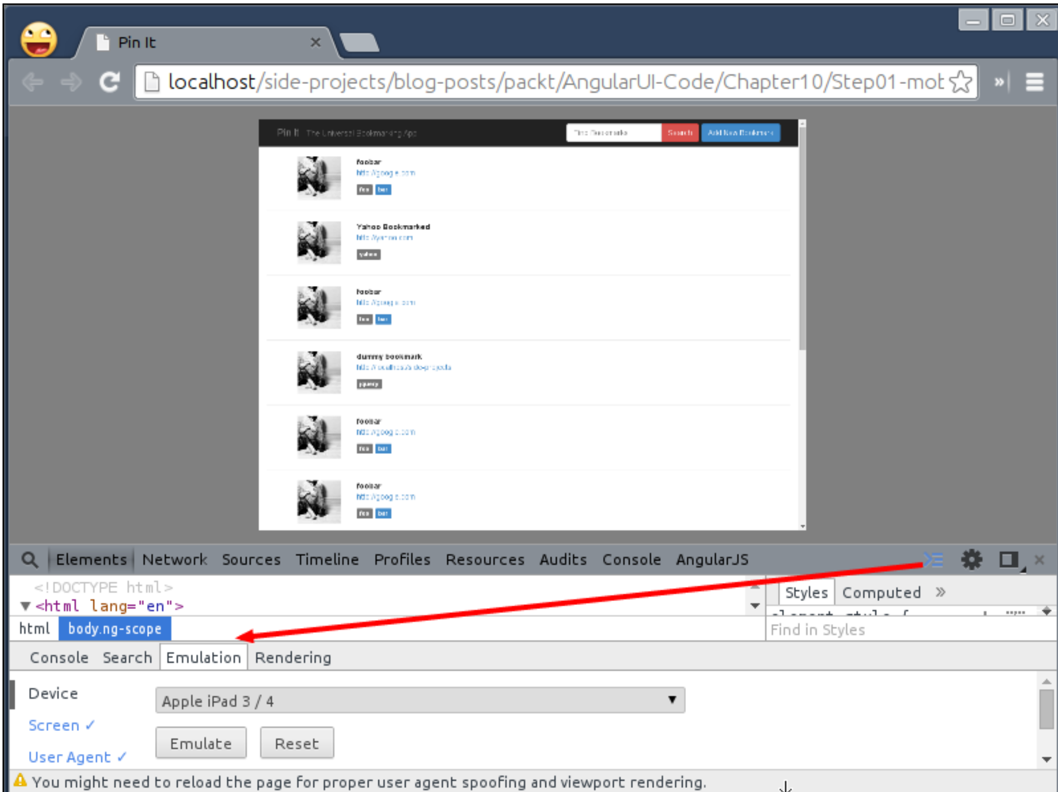
Going forward with this, any swipe gesture (left/right) will be captured and the appropriate method will be called thereafter. We now have to define a method named `paginate` under `MainCtrl` in `controllers.js`. First, inject `$rootScope` in `MainCtrl` as follows:

```
angular.module('myApp.controllers', [])
  .controller('MainCtrl', function($scope, $rootScope, BookmarkStore,
$route, $location, $filter) {
```

Add the `paginate()` method in the same controller:

```
$scope.paginate = function(mode) {
  if ((mode === 'backward' && $scope.isFirst()) || (mode === 'forward'
&& $scope.isLast())) return;

$rootScope.slideDirection = mode;
$location.path('/page/' + ( mode === 'forward' ? $scope.currentPage +
1 : $scope.currentPage - 1 ));
};
```

Here, the mode that is `forward` or `backward`, which is passed to the function, helps us to find the next or previous page to jump to, based on `currentPage`. Then, we bind the mode itself on the `$rootScope` object that we'll see soon. At this time, swiping should be working fine for you. You can either check the application in a tablet if you have one or use the **Chrome DevTool** emulation as shown in the following screenshot:



You can click on the third drawer icon on the top-right corner in DevTool and go to the **Emulation** tab. Then, select any device and click on the **Emulate** button. Now the touch events will work on desktop too. Try swiping left or right to check whether the list updates. How about adding a transition when the list changes?

Animation for better user experience

As AngularJS handles all sort of operations, including the updating of views and automatic binding of events, it is quite difficult to add animation in between that requires imperative DOM manipulation. The AngularJS team addressed the problem early on and came up with an `angular-animate` module to easily add animation in no time. You already learned about animations in *Chapter 5*, *Learning Animation*, so we'll go straight into installing it. Run the following commands in a terminal:

```
bower install --save angular-animate
bower install --save modernizr
```

Add `modernizr.js` before jQuery in `index.html` and `angular-animate.js` after `angular-touch.js`. Then, inject the `ngAnimate` module in `app.js` for use as follows:

```
angular.module('myApp', ['ngRoute', 'ngAnimate', 'ngTouch', 'myApp.
controllers', 'myApp.services'])
  .config(function($routeProvider){
    $routeProvider.when('/page/:pageNumber', {templateUrl: 'views/
bookmarks.html'});
    $routeProvider.otherwise({redirectTo: '/page/1'});
  })
```

The `ngClass` directive allows us to dynamically apply the `forward` or `backward` CSS class upon swiping. That's why we bound the same as the `slideDirection` model in `MainCtrl` earlier. We can now apply it on the `div[ng-view]` element in `index.html`:

```
<div ng-view class="view-animation" ng-class="slideDirection"></div>
```

When you swipe to the left, the `slideDirection` model value changes to `forward` and on swiping to the right, it changes to `backward`. The same class gets applied on the `div` element as shown previously. This is useful to have a different animation when you swipe to the left or right, based on the CSS class it has. We want the content to flow from right to left when we swipe to the left and left to right when we swipe to the right. Let's add the CSS styles in `css/style.css` and then reference it in `index.html` under the head tag as follows:

```
html.touch .view-animation.backward.ng-enter {
  position: relative;
  -webkit-animation: enter_backward 0.5s linear;
```

```
      -moz-animation: enter_backward 0.5s linear;
        -o-animation: enter_backward 0.5s linear;
           animation: enter_backward 0.5s linear;
}
@-webkit-keyframes enter_backward {
  from { left: -100%; }
  to   { left: 0%; }
}
@-moz-keyframes enter_backward {
  from { left: -100%; }
  to   { left: 0%; }
}
@-o-keyframes enter_backward {
  from { left: -100%; }
  to   { left: 0%; }
}
@keyframes enter_backward {
  from { left: -100%; }
  to   { left: 0%; }
}

html.touch .view-animation.forward.ng-enter {
  position: relative;
  -webkit-animation: enter_forward 0.5s linear;
      -moz-animation: enter_forward 0.5s linear;
        -o-animation: enter_forward 0.5s linear;
           animation: enter_forward 0.5s linear;
}
@-webkit-keyframes enter_forward {
  from { left: 100%; }
  to   { left: 0%; }
}
@-moz-keyframes enter_forward {
  from { left: 100%; }
  to   { left: 0%; }
}
@-o-keyframes enter_forward {
  from { left: 100%; }
  to   { left: 0%; }
}
@keyframes enter_forward {
  from { left: 100%; }
  to   { left: 0%; }
}
```

As you learned in *Chapter 5*, *Learning Animation*, we are just targeting the `ng-enter` event here so that a new list of bookmarks will be animating in either from the left or right. You can have it animating out when the old list of bookmarks wipe out but I'll leave that to you as an exercise. Also, notice that we want this transition on touch devices only, that's why we installed Modernizr earlier. Modernizr adds the `touch` CSS class on the HTML tag, which we have used to target touch-enabled devices only.

> Note that CSS3 keyframe animation requires vendor prefixes to support different browsers such as `-webkit` for Safari and Chrome, `-moz` for Firefox, and so on. You can check out `http://caniuse.com/#search=keyframe` for more details.

Now we have introduced swipe gestures to navigate through a list of bookmarks, it's time to get rid of the pagination widget on touch devices. Add the following `ng-if` directive to show the widget only on the desktop:

```
<div class='text-center' ng-if="isDesktop()">
<ul class="pagination">
<li ng-class="{'disabled': isFirst()}">
<span ng-if="isFirst()">&laquo;</span>
<a ng-if="!isFirst()" ng-href="#/page/{{currentPage - 1}}">&laquo;</a>
</li>
<li ng-repeat="page in pages" ng-class="{'active': page ==
currentPage}">
<a ng-href="#/page/{{$index + 1}}" ng-bind="page"></a>
</li>
<li ng-class="{'disabled': isLast()}">
<span ng-if="isLast()">&raquo;</span>
<a ng-if="!isLast()" ng-href="#/page/{{currentPage + 1}}">&raquo;</a>
</li>
</ul>
</div>
```

Add the following in `MainCtrl` of `controllers.js`:

```
$scope.isDesktop = function() {
   return !Modernizr.touch && $scope.pages;
 };
```

Take a look at the application now in both desktop and touch device (or in emulation using Chrome DevTools) to see different but notable behaviors.

Mobile optimization for a better user experience

In this section, we are going to optimize the whole application that we built in the previous section to perform well on low-end mobile devices. Sometimes, you need to cut down a few features (that are not important in mobile scope) to do so like other mobile applications such as Facebook, Twitter, or Google+ mobile apps. In our case, there is no extraneous feature or functionality that we can remove, so we can skip this step of optimization for PinIt. There are other things that we can improve upon to make our application rock.

Periodic delay for tap events

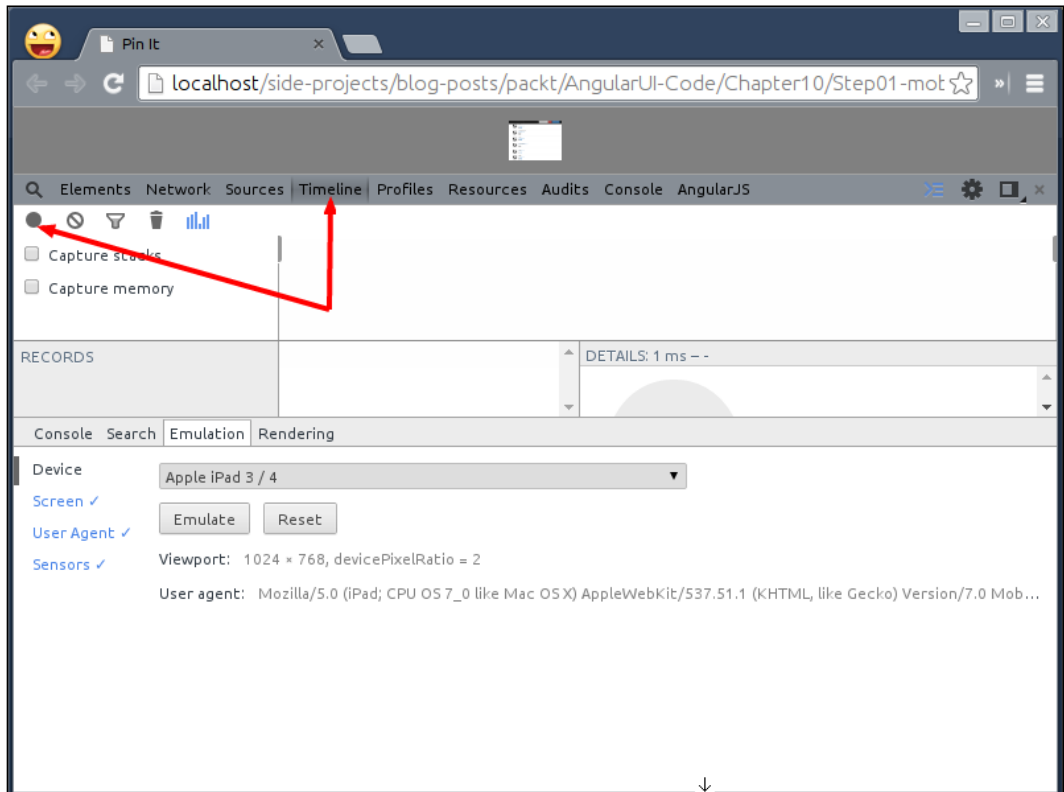
All mobile devices wait approximately 300 milliseconds from the time that you tap the button to fire the `click` event. The reason for this is that the browser is waiting to see whether you are actually performing a double tap. However, this check often leads to a laggy delay while performing user actions and sometimes gives a bad impression to your users. To eliminate this problem, we often have to use an external JavaScript library, such as `FastButton`, `FastClick`, and so on, to convert all the `click` events to `touch` on mobile.

The good thing is that AngularJS itself takes care of this with the `ngTouch` module that we enabled earlier. For touch-enabled devices, `ngClick` converts the `click` events to `touch` events under the hood by excluding the 300 ms delay so that your application feels native to end users.

Accelerated transitions and animations

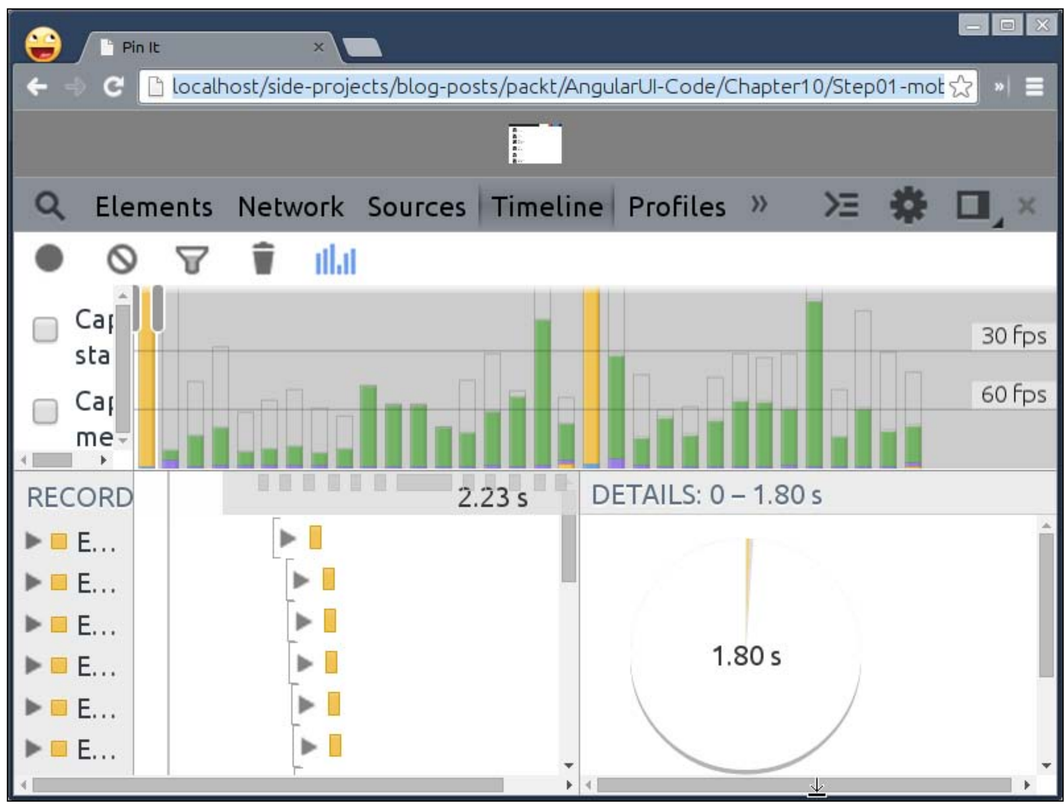
Knowing the fact that animations and transitions really make your application user-friendly and intuitive but it might be disastrous on mobile devices if you do not pay attention to it. For this, we need to know what happens when animation happens in the browser. The mantra of silky smooth animation is 60 **frames per second** (**FPS**), that is, render everything within 16 ms. Chrome DevTool's timeline feature is extremely useful to understand how page transition works in our application. Let's take a look at it.

Open our PinIt app in Google Chrome and press *F12* to open up **devtool** as shown in the following screenshot:



As highlighted in the screenshot, you need to emulate any mobile device (it's Apple iPad in this case) and click on the **Timeline** tab on the top. The gray circle highlighted on the left is to capture the timeline when the animation is running.

Now, go ahead and click on that gray circle and swipe through a few bookmarks. You will notice that many green bars almost cross the 60 FPS limit, as shown as follows:



This gives us an insight into how badly our application can perform on low-end mobile devices, which is not a good thing. The green lines represent the browser repaint events in which the browser has to repaint pixels for every animation frame. The more it repaints, the more janky the animation will become.

The problem is with the CSS property, `left`, that we use to animate a list. It is recommended to use the `transform` CSS property to animate cheaply. The reason is that `transform` uses hardware acceleration to animate stuff. This means we want the mobile device CPU to set up the initial animation and then have the GPU responsible for composing different layers during the animation process. The native application can access the device's GPU to perform animation smoothly, which was not possible until the CSS transform standardized in HTML. So, now we can leverage the same to have jank-free animations at a 60 FPS scale.

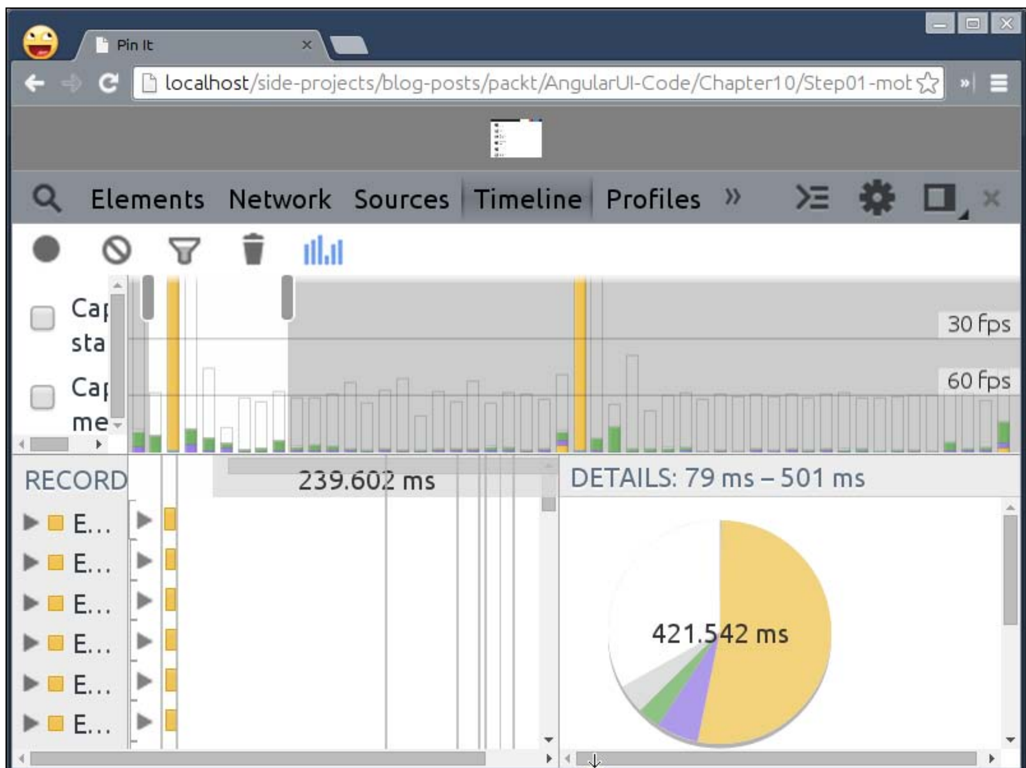
Let's hardware accelerate the page transition in `style.css` as follows:

```
html.touch .view-animation.backward.ng-enter {
  -webkit-animation: enter_backward 0.5s linear;
     -moz-animation: enter_backward 0.5s linear;
       -o-animation: enter_backward 0.5s linear;
          animation: enter_backward 0.5s linear;
}
@-webkit-keyframes enter_backward {
  from { -webkit-transform: translate(-100%, 0); }
  to   { -webkit-transform: translate(0%, 0); }
}
@-moz-keyframes enter_backward {
  from { -moz-transform: translate(-100%, 0); }
  to   { -moz-transform: translate(0%, 0); }
}
@-o-keyframes enter_backward {
  from { -o-transform: translate(-100%, 0); }
  to   { -o-transform: translate(0%, 0); }
}
@keyframes enter_backward {
  from { transform: translate(-100%, 0); }
  to   { transform: translate(0%, 0); }
}


html.touch .view-animation.forward.ng-enter {
  -webkit-animation: enter_forward 0.5s linear;
     -moz-animation: enter_forward 0.5s linear;
       -o-animation: enter_forward 0.5s linear;
          animation: enter_forward 0.5s linear;
}
@-webkit-keyframes enter_forward {
  from { -webkit-transform: translate(100%, 0); }
  to   { -webkit-transform: translate(0%, 0); }
}
@-moz-keyframes enter_forward {
  from { -moz-transform: translate(100%, 0); }
  to   { -moz-transform: translate(0%, 0); }
}
@-o-keyframes enter_forward {
  from { -o-transform: translate(100%, 0); }
  to   { -o-transform: translate(0%, 0); }
}
```

```
@keyframes enter_forward {
  from { transform: translate(100%, 0); }
  to   { transform: translate(0%, 0); }
}
```

The only difference here is that we have replaced all the instances of the CSS property named `left` with `translate`. The `translate` property takes two parameters as the *x* and *y* axes. We are only considering the *x* axis to animate horizontally.

Refresh the browser and record the timeline again to see the difference in painting as shown as follows:



As you can see, there are almost no green bars, which means the animation will perform extremely smoothly without any performance overhead on low-end devices. We have barely scratched the surface here when it comes to performance optimization. Heavy JavaScript can also cause performance issues but we do not have it in our application and hence it can not be addressed. Visit `http://jankfree.org` for cool stuff on performance optimization.

Improving initial page load

We often have a lot of crazy things on a development server that might or might not require on production so that the application should be extremely lightweight to load quickly for users. Let's create a build for PinIt, which we already covered in *Chapter 1*, *Setting Up the Environment*. So, our strategy to reduce the initial page load time is to concatenate all JavaScript and CSS files to reduce multiple requests. This boils down further to add a subset of JavaScript or CSS libraries, which are relevant for the application by excluding unnecessary baggage. In this section, we'll extend what we learned in the first chapter and install additional Grunt modules to reduce the initial page load time.

First, create `package.json` in the `Chapter10/Step01-mobile/` directory with the following code:

```
{
  "name": "pin-it",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.1",
    "grunt-contrib-copy": "~0.4.1",
    "grunt-contrib-concat": "~0.3.0",
    "grunt-targethtml": "~0.2.6"
  }
}
```

Then, open up a terminal to install the previous Grunt packages and run the following command:

```
cd Chapter10/Step01-mobile
npm install
```

Now, write a Grunt script in `Gruntfile.js` in `Chapter10/Step01-mobile` as follows:

```
module.exports = function(grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    concat: {
      options: { separator: ';' },
      dist: {
        src: [
          'src/bower_components/modernizr/modernizr.js',
          'src/bower_components/jquery/dist/jquery.js',
          'src/bower_components/angular/angular.js',
```

```
            'src/bower_components/angular-route/angular-route.js',
            'src/bower_components/angular-touch/angular-touch.js',
            'src/bower_components/angular-animate/angular-animate.js',
            'src/bower_components/bootstrap/dist/js/bootstrap.js',
            'src/js/app.js',
            'src/js/controllers.js',
      'src/js/services.js'
          ],
          dest: 'dist/js/<%= pkg.name %>.js'
        }
      },
      copy: {
        main: {
          files: [
            { expand: true, cwd: 'src/css/', src: ['**'], dest: 'dist/
css/' },
            { expand: true, cwd: 'src/bower_components/bootstrap/dist/
css/', src: ['bootstrap.css'], dest: 'dist/bower_components/bootstrap/
dist/css' },
            { expand: true, cwd: 'src/views/', src: ['**'], dest: 'dist/
views/' }
          ]
        }
      },
      targethtml: {dist:{files:{'dist/index.html': 'src/index.html'}}},
    });
    grunt.loadNpmTasks('grunt-contrib-concat');
    grunt.loadNpmTasks('grunt-contrib-copy');
    grunt.loadNpmTasks('grunt-targethtml');
    grunt.registerTask('dist', ['concat', 'targethtml', 'copy']);
};
```

This is the exact `Gruntfile.js` file that we created in *Chapter 1, Setting Up the Environment*, we just added some extra JavaScript and HTML files to copy to the `dist/` directory when the build command is run.

Finally, wrap all the scripts tags in `index.html` into a comment block to reference the concatenated `pin-it.js` file:

```
<!--(if target dev)><!-->
<script src="bower_components/modernizr/modernizr.js"></script>
<script src="bower_components/jquery/dist/jquery.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-route/angular-route.js"></
script>
```

```
<script src="bower_components/angular-touch/angular-touch.js"></
script>
<script src="bower_components/angular-animate/angular-animate.js"></
script>
<script src="bower_components/bootstrap/dist/js/bootstrap.js"></
script>
<script src="js/app.js"></script>
<script src="js/controllers.js"></script>
<script src="js/services.js"></script>
<!--<!(endif)-->
<!--(if target dist)>
<script src="js/pin-it.js"></script>
<!(endif)-->
```

The Grunt plugin named `targetHtml` will replace all the JS files with `pin-it.js` during the build process. Now, everything is up and running. We can go ahead and create our first build for PinIt; run the following commands in a terminal:

```
cd Chapter10/Step01-mobile
grunt dist
```

You should see the **Done, without errors.** message in green in a terminal by now. Fire up a browser and open `dist/index.html` to see the production PinIt application working as expected.

The one thing that worries us is the size of the production app, which is almost 1.4 MB; a bad thing for our users on mobile devices with low bandwidth. Let's fix it.

First, modify `package.json` to add couple of Grunt modules as follows:

```
{
  "name": "pin-it",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.1",
"grunt-contrib-clean": "~0.4.0",
    "grunt-contrib-copy": "~0.4.1",
    "grunt-contrib-concat": "~0.3.0",
 "grunt-contrib-cssmin": "~0.5.0",
"grunt-contrib-uglify": "~0.2.0",
    "grunt-ng-annotate": "^0.3.0",
    "grunt-contrib-htmlmin": "^0.3.0",
    "grunt-targethtml": "~0.2.6"
  }
}
```

There are four new Grunt modules, namely, `clean`, `cssmin`, `ngAnnotate`, and `uglify`. Run the following command to install the remaining Grunt modules that we just added in `package.json`:

```
cd Chapter10/Step01-mobile
npm install
```

This will install new Grunt modules in the `node_modules/` directory, so we can use them in `Gruntfile.js` to fix the build process. Replace the whole `Gruntfile.js` file with the following code:

```
module.exports = function (grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
clean: { dist: { files: [{ dot: true, src: ['.tmp', 'dist/*']}]},
post: { files: [{ dot: true, src: ['dist/.tmp']}]} },
ngAnnotate: {
      options: { singleQuotes: true },
      app: { files: {
        'dist/.tmp/js/app.js': ['src/js/app.js'],
        'dist/.tmp/js/controllers.js': ['src/js/controllers.js'],
        'dist/.tmp/js/services.js': ['src/js/services.js']
      }}
    },
copy: { main: { files: [ { expand: true, cwd: 'src/views/', src:
['**'], dest: 'dist/views/' }] } },
 cssmin: { dist: { files: { 'dist/css/style.css': [
      'src/bower_components/bootstrap/dist/css/bootstrap.css',
      'src/css/style.css'
    ]}}},
concat: {
      options: { separator: ';' },
      dist: {
        src: [
          'src/bower_components/modernizr/modernizr.js',
          'src/bower_components/jquery/dist/jquery.js',
          'src/bower_components/angular/angular.js',
          'src/bower_components/angular-route/angular-route.js',
          'src/bower_components/angular-touch/angular-touch.js',
          'src/bower_components/angular-animate/angular-animate.js',
          'src/bower_components/bootstrap/dist/js/bootstrap.js',
'dist/.tmp/js/app.js',
          'dist/.tmp/js/controllers.js',
          'dist/.tmp/js/services.js'
        ],
```

```
        dest: 'dist/js/<%= pkg.name %>.js'
      }
    },
   uglify: {dist:{files:{'dist/js/<%= pkg.name %>.js': ['dist/js/<%=
  pkg.name %>.js']}}},
  targethtml: {dist:{files:{'dist/index.html': 'src/index.html'}}},
  htmlmin: {
      dist: {
        options: {
          removeCommentsFromCDATA: true,
          collapseWhitespace: true,
          collapseBooleanAttributes: true,
          removeAttributeQuotes: true,
          removeRedundantAttributes: true,
  useShortDoctype: true,
          removeEmptyAttributes: true,
          removeOptionalTags: true
        },
        files: [
          { expand: true, cwd: 'dist', src: ['index.html'], dest:
  'dist' },
          { expand: true, cwd: 'dist/views', src: ['bookmarks.html'],
  dest: 'dist/views' }
        ]
      }
    }
   });
   grunt.loadNpmTasks('grunt-contrib-clean');
   grunt.loadNpmTasks('grunt-contrib-copy');
   grunt.loadNpmTasks('grunt-contrib-cssmin');
   grunt.loadNpmTasks('grunt-contrib-concat');
   grunt.loadNpmTasks('grunt-contrib-uglify');
   grunt.loadNpmTasks('grunt-ng-annotate');
   grunt.loadNpmTasks('grunt-targethtml');
   grunt.loadNpmTasks('grunt-contrib-htmlmin');
   grunt.registerTask('dist', ['clean', 'ngAnnotate', 'copy', 'cssmin',
  'concat', 'uglify', 'targethtml', 'htmlmin', 'clean:post']);
  };
```

The following are a few points to remember:

- The `clean:dist` task removes all the files from the previous build.
- The `ngAnnotate` task annotates dependencies in `app.js`, `controllers.js`, and `services.js` so that it does not break when minified and gzipped. The annotated files are kept in the `dist/.tmp/` directory.

- The `copy` task moves the `index.html` and `bookmarks.html` files as is in the `dist/` directory.
- The `cssmin` task combines `bootstrap.css` and `style.css` into the `dist/css/style.css` file.
- The `concat` task merges all JS files into `pin-it.js`. Notice that it takes `app.js`, `controllers.js`, and `services.js` from the `dist/.tmp` directory instead of the `src/` directory because they are annotated by the `ngAnnotate` task earlier.
- The `uglify` task takes `pin-it.js` from the `dist/js/` directory and then minifies and gzips it.
- The `targethtml` task updates the references of the newly created `pin-it.js` and `style.css` files in `index.html` with the help of the `<!--(if target dist)>` comment block.
- The `htmlmin` task removes unnecessary spaces and HTML tags to reduce the size further.
- At the end, the `clean:post` task removes the `dist/.tmp` directory created by the `ngAnnotate` task earlier.

Then, modify the head tag in `index.html` to use the minified `style.css` file in production as follows:

```
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, user-scalable=no">
<title>Pin It</title>
<!--(if target dev)><!-->
<link rel="stylesheet" href="bower_components/bootstrap/dist/css/
bootstrap.css">
<link rel="stylesheet" type="text/css" href="css/style.css">
<!--<!(endif)-->
<!--(if target dist)>
<link rel="stylesheet" type="text/css" href="css/style.css">
<!(endif)-->
</head>
```

It's time to create a new build again. Run the `grunt dist` command again in a terminal to *gruntify*, I mean gratify yourself to know that the overall size of the application reduces to merely 400 KB from 1.4 MB. That's incredible!

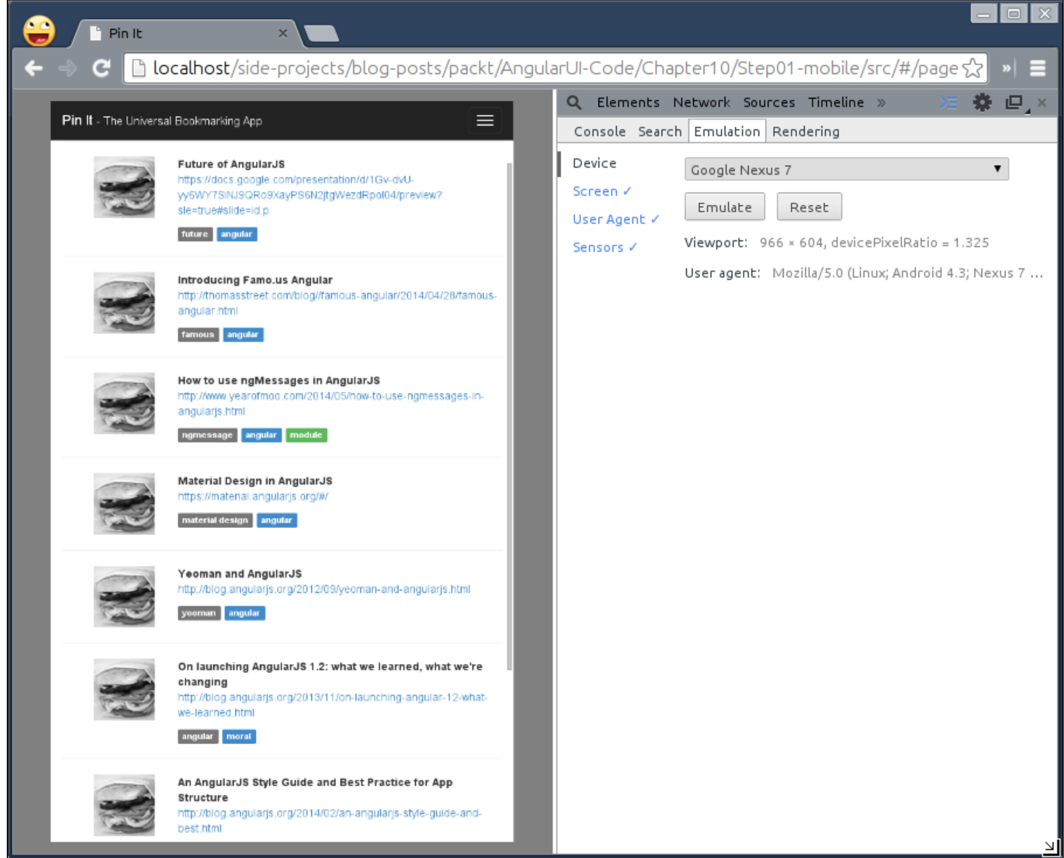
Again, we included the entire Bootstrap framework in `index.html` but only using the modal widget. Why not use just `modal.js` and `collapse.js` instead? Go ahead and replace the reference of `bootstrap.js` in `index.html` with the following code:

```
<script src="bower_components/bootstrap/js/modal.js"></script>
<script src="bower_components/bootstrap/js/collapse.js"></script>
```

Then, update the `concat` task in `Gruntfile.js` as follows:

```
concat: {
      options: { separator: ';' },
      dist: {
        src: [
          'src/bower_components/modernizr/modernizr.js',
          'src/bower_components/jquery/dist/jquery.js',
          'src/bower_components/angular/angular.js',
          'src/bower_components/angular-route/angular-route.js',
          'src/bower_components/angular-touch/angular-touch.js',
          'src/bower_components/angular-animate/angular-animate.js',
          'src/bower_components/bootstrap/js/modal.js',
          'src/bower_components/bootstrap/js/collapse.js',
          'dist/.tmp/js/app.js',
          'dist/.tmp/js/controllers.js',
          'dist/.tmp/js/services.js'
        ],
        dest: 'dist/js/<%= pkg.name %>.js'
      }
    },
```

This will reduce our build size down to 350 KB, which is super impressive! In addition to this, we can even use a customized `bootstrap.css` file to reduce its size further but I'll leave that to you as an exercise. On a final note, here is **Pin It** in the Google Nexus 7 device:



You might have noticed that the thumbnails we show next to each bookmark are common for all because we have not yet fetched them from the bookmark URLs, which can only be done using a server-side programming language, such as Node.js, which is out of the scope of this book. Alternatively, you can use an online API (`http://grabz.it`) to do so as an exercise for you.

Summary

In this chapter, you learned a new term called mobile first to build anything on the Web. To understand it, we built a simple bookmarking application with mobile users in mind and made it work on the desktop PC at the same time. We then leverage the browser's internal storage API to have a persistent layer on top. Towards the end of the chapter, we were bugged by a few performance issues on low-end devices. We figured that the `transform` property in CSS offloads heavy animations on the GPU to be performed without any jank. We also tried to reduce the overall size of the application to mere KBs from MBs earlier, which helped us load the application in a snap on low bandwidth. All in all, we built a really useful application that we can take to the next level by adding extra features. I'm really excited to see what you will build next. Keep learning. All the best!

Index

Symbols

$animate service object 95
$filter filter 216
$http service 131
$location object 216
$scope.$apply function 73
$scope.blurCallback function 48
$scope.dayClick function 72
$scope.eventSources array 72
$scope.getModel function 43
$scope object 12
$scope.remove function 72
$templateCache
  external template, loading via 189
$timeout function 48
.gitignore file
  modifying 65
@media 139, 140
@media expression 141
@time-scale variable 106

A

accordion UI
  reference link 161
add new issue form 174
alert 176
angular-animate
  installing 94
Angular Google chart tools
  URL 124
  using 124-126
AngularJS
  about 53
  bootstrapping 58
  service, creating 81, 82
  URL 9
angular-seed project
  URL 19
angular-touch module
  installing 217
AngularUI
  about 38
  calendar component 65
  downloading 38
  Google Maps component 57
AngularUI Bootstrap
  about 158
  accordion, creating 161-164
  buttons, utilizing 174
  carousel, using 177-180
  content, hiding with collapse 169-172
  customizing 187
  pagination widget, customizing 190-195
  PMA, building 158, 159
  priorities, converting in form of ratings 175
  progress bar, displaying status of issue 181
  tabs, creating 164-169
  tab widget, extending 196-202
  timelines, setting with datepicker 172, 173
  users, notifying with alert messages 176
AngularUI-Utils
  building 38
  integrating, into project 40-42
  URL 38
animate.css
  URL 106
  using 106-108
animations
  creating 95-100
  scaling, LESS used 104-106

staggering 108-110
application dependencies
managing, with Bower 63, 64
application foundation
designing 148-154
application-specific menus
common housing, with dropdown 183, 184
Arshaw FullCalendar
URL 65

B

bar chart
converting, into widget 122
creating 116, 117
bar chart data driven
creating 118-121
bar-chart directive
creating 122-124
Bezier curves
applying 103, 104
bookmarking app
animations, installing 219-221
building, with mobile first approach 204-207
crafting for mobile devices 217-219
creating, as dynamic 208-214
search, enabling 215, 216
Bootstrap
download link 42
Bower
client-side dependencies, managing with 16-18
URL 16
bower.json file
about 64
setting up, manually 64

C

calendar component
about 65-67
application, building 76
controller, adapting 71-73
controller, testing 74, 75
date formatting, filter used 67-70
filter, testing 75
styling 70
tests, adding 73
callback function 58
charts
about 116
bar chart, creating 116, 117
bar chart data driven, creating 118-121
importance 116
Chrome DevTool emulation 218
clean:dist task 231
clean:post task 232
client-side dependencies
managing, with Bower 16-18
concat task 53, 232
Content Delivery Network (CDN) 9
copy task 232
CouchDB 185
cssmin task 232
cubic-bezier function 103

D

D3.js
URL 127
dashboard
building, GitHub REST API used 127-131
extending 132-134
data service
creating 81, 82
datepicker widget
about 172
used, for setting timelines 172
declarative approach 88
directive
building 13
dist folder 41

E

elements
animating 101, 102
Event Binder component 46-49
external template
about 187, 188
loading, via $templateCache 189
loading, via script tag 188
using 189, 190

F

filters.spec.js 75
Foundation 148

G

Git
about 17
installing, on Linux (Ubuntu) 33
installing, on OS X 33
installing, on Windows 33
source code, managing with 33-35
GitHub
about 38, 127
URL 127
GitHub REST API
used, for building dashboard 127-131
Glyphicons 169
Google Maps
about 57
embedding 57-60
event binding 62, 63
markers 61
Grunt
about 28
Protractor, running from 31-33
gruntFile.js
clean:rm_tmp task 39
concat:modules task 39
concat:tmp task 39
uglify task 39
grunt task concat 41

H

Hello World application
about 7-11
building 28-31
directive, building 13, 14
end-to-end tests, with Protractor 24-28
integration tests, with Protractor 24-28
objects, using instead of primitives 11, 12
testing 19
unit tests, performing 19, 20
Homebrew
reference link 15
htmlmin task 232

J

Jasmine
about 20
installing 20-23
reference link 20
Java runtime
reference link 25
JavaScript-defined animations
performing 110-113
jQLite 53
jQuery 53
jQuery Passthrough component 52-55
JsHint
about 28
URL 28
JSON.stringify function 44

K

Karma
about 20
installing 20-23
reference link 20
Keypress component 50-52

L

Leaner CSS (LESS)
used, for scaling entire animations 104-106
Linux (Ubuntu)
Git, installing on 33
Node.js, installing on 16
NPM, installing on 16

M

master/details view
using 88
media queries
@media 139
@media expression 140
expression 139
overview 139
URL 142
media type
about 139
URL 139

mobile application
designing 204
mobile optimization
accelerated animations 222-226
accelerated transitions 222-226
initial page load, improving 227-234
performing 222
periodic delay, for tap events 222
Modernizr
about 112, 221
installing 112
moment.js file
URL 68
MongoDB 185
MySQL 185

N

ngAnnotate task 231
ng-grid component
URL 79
ng-grid project
grid, grouping 86, 87
master/details view, using 88-90
setting up 80
simple grid view 82-86
ngHide directive 95
ng-repeat directive 120
ngRepeatEnd directive 211
ngRepeatStart directive 211
ngRoute module 213
ng-show attribute 51
ng-style attribute 47
Node.js
about 15
installing, on Linux (Ubuntu) 16
installing, on OS X 15
installing, on Windows 15
reference link 15
NPM
about 15
installing, on Linux (Ubuntu) 16
installing, on OS X 15
installing, on Windows 15
NVD3
URL 127

O

OS X
Git, installing on 33
Node.js, installing on 15
NPM, installing on 15

P

progress bar
issue status, displaying 181
project
reference link 38
setting up 94
Project Management Application (PMA)
building 158, 159
Protractor
about 24, 28
reference link 24
running, from Grunt 31-33

R

responsive design
evolution 138
URL 138

S

SASS 104
script tag
external template, loading via 188
service singleton 81
setContainer() method 121
setDetails() method 121
ShipIt 158, 196, 199
simple grid view 82-86
Single Page Application 159
source code
managing, with Git 33-35
staggering
animations 108-110
working 110

T

targethtml task 232
test-driven development (TDD) 73
tooltip() function 55
Twitter Bootstrap
  about 142
  URL 142
  using 143-147
typeahead directive
  using 182, 183
types, @media
  all 139
  print 139
  screen 139

U

uglify task 232
uiMask directive component 42-46
unit tests, Hello World application
  performing 19, 20

V

Version Control System (VCS) 33

W

W3C (World Wide Web Consortium) 138
widget
  bar chart, converting into 122
Windows
  Git, installing on 33
  Node.js, installing on 15
  NPM, installing on 15

Y

Yeoman
  URL 7

Thank you for buying
AngularJS UI Development

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

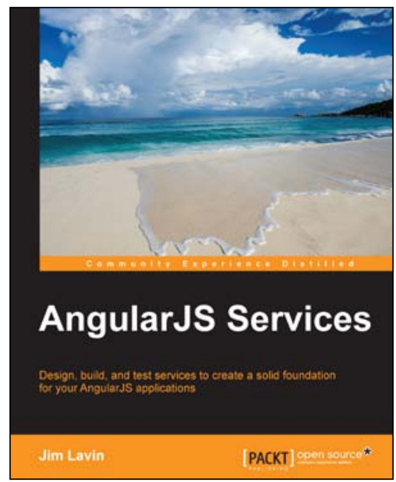
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

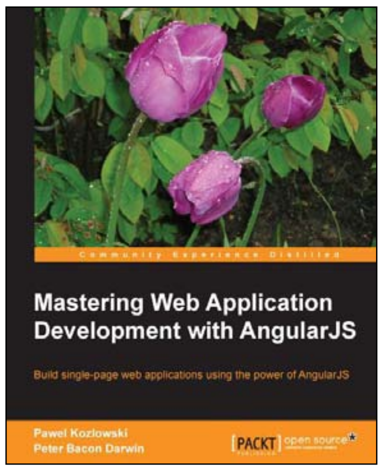


AngularJS Services

ISBN: 978-1-78398-356-8 Paperback: 152 pages

Design, build, and test services to create a solid foundation for your AngularJS applications

1. Understand how services are a vital component of the AngularJS framework and how leveraging services can benefit your application.
2. Design and structure your AngularJS services and learn the best practices used in designing AngularJS services.
3. Effectively write, test, and finally deploy your application.



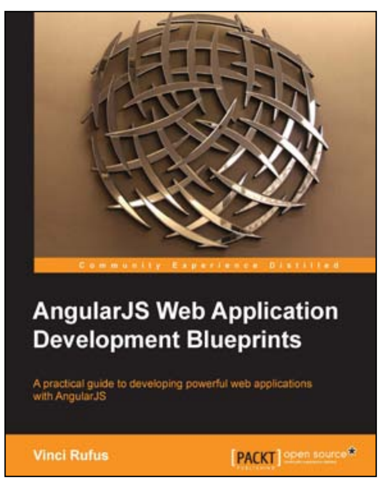
Mastering Web Application Development with AngularJS

ISBN: 978-1-78216-182-0 Paperback: 372 pages

Build single-page web applications using the power of AngularJS

1. Make the most out of AngularJS by understanding the AngularJS philosophy and applying it to real-life development tasks.
2. Effectively structure, write, test, and finally deploy your application.
3. Add security and optimization features to your AngularJS applications.

Please check **www.PacktPub.com** for information on our titles

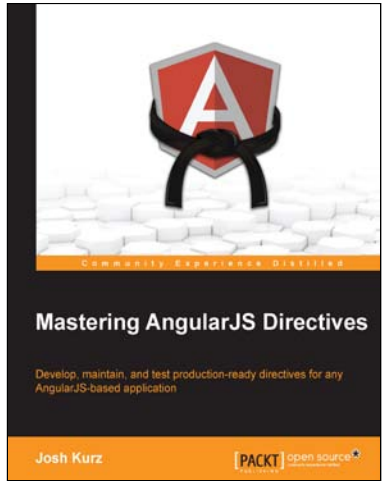


AngularJS Web Application Development Blueprints

ISBN: 978-1-78328-561-7 Paperback: 300 pages

A practical guide to developing powerful web applications with AngularJS

1. Get to grips with AngularJS and the development of single-page web applications.
2. Develop rapid prototypes with ease using Bootstraps Grid system.
3. Complete and in depth tutorials covering many applications.



Mastering AngularJS Directives

ISBN: 978-1-78398-158-8 Paperback: 210 pages

Develop, maintain, and test production-ready directives for any AngularJS-based application

1. Explore the options available for creating directives, by reviewing detailed explanations and real-world examples.
2. Dissect the life cycle of a directive and understand why they are the base of the AngularJS framework.
3. Discover how to create structured, maintainable, and testable directives through a step-by-step, hands-on approach to AngularJS.

Please check **www.PacktPub.com** for information on our titles